МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет

Факультет електроніки та інформаційних технологій

Кафедра комп'ютерних наук

«До захисту допущено»

В.о. завідувача кафедри

Ігор ШЕЛЕХОВ

_____

(підпис)

червня 202_ р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавр

зі спеціальності 122 Комп'ютерні науки,

освітньо-професійної програми «Інформатика»

на тему: «Мобільний застосунок для мониторингу якості телекомунікаційних послуг»

здобувача групи ІН-02 Сіньковського Віктора Андрійовича

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ Віктор СІНЬКОВСЬКИЙ

(підпис)

Керівник, старший викладач кафедри комп'ютерних наук, к.ф.-м.н. Дмитро ВЕЛИКОДНИЙ

_____

(підпис)

Суми – 2024

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ
на здобуття освітнього ступеня бакалавра
зі спеціальності 122 – Комп'ютерних наук, освітньо-професійної програми «Інформатика»
здобувача групи ІН-02 Сіньковського В. А.

1. Тема роботи: «Мобільний застосунок для моніторингу якості телекомунікаційних послуг» затверджую наказом по СумДУ від «22» квітня 2024 р. № 0414-VI
2. Термін здачі здобувачем кваліфікаційної роботи до 01 червня 2024 року
3. Вхідні дані до кваліфікаційної роботи
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
1) Аналіз проблеми предметної області, постановка й формування завдань дослідження. 2) Огляд технологій для розробки мобільного застосунку для моніторингу якості телекомунікаційних послуг. 3) Розробка інформаційної системи застосунку для моніторингу якості телекомунікаційних послуг. 4) Аналіз отриманих результатів. 5) Оформлення пояснювальної записки до кваліфікаційної роботи.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____
6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «_____» _____ 2024 р.

Завдання прийняв до виконання _____ Керівник _____
(підпис) (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	Аналіз проблеми предметної області, постановка й формування завдань дослідження	06.05-10.05	
2	Огляд технологій для розробки мобільного застосунку для моніторингу якості телекомунікаційних послуг	10.05-12.05	
3	Розробка інформаційної системи застосунку для моніторингу якості телекомунікаційних послуг	13.05-27.05	
4	Аналіз отриманих результатів	28.05	
5	Оформлення пояснювальної записки до кваліфікаційної роботи	29.05-01.06	

Здобувач вищої освіти _____ Керівник _____

(підпис) (підпис)

АНОТАЦІЯ

Записка: **97** стор., 18 рис., 2 додаток, 54 джерел.

Обґрунтування актуальності теми роботи – Тема кваліфікаційної роботи належить до актуальних напрямків розвитку інформаційних технологій, оскільки в сучасному світі велике значення набувають мобільні застосунки для моніторингу якості телекомунікаційних послуг. Проблема забезпечення високоякісного зв'язку та доступу до інтернету є важливою як для споживачів, так і для операторів зв'язку.

Об'єкт дослідження – Процес моніторингу та оцінки якості телекомунікаційних послуг за допомогою мобільних застосунків.

Мета роботи – Розробка мобільного застосунку для автоматизованого моніторингу якості телекомунікаційних послуг. Ця система має на меті забезпечити зручний та ефективний процес збирання, аналізу та візуалізації даних про якість зв'язку та доступу до інтернету.

Методи дослідження – У роботі використовуються методи аналізу та розробки програмного забезпечення, зокрема використання інструментів для збору даних, аналізу продуктивності мережі, моніторингу та візуалізації даних, а також принципів мобільної розробки.

Результати – Розроблено мобільний застосунок, який дозволяє автоматизувати процес моніторингу якості телекомунікаційних послуг. Застосунок включає в себе інструменти для ефективного збирання та аналізу даних, що дозволяє забезпечити надійну та швидку оцінку якості зв'язку для користувачів та операторів телекомунікаційних послуг.

ІНФОРМАЦІЙНА СИСТЕМА, REACT NATIVE, TYPESCRIPT/JAVASCRIPT, REST API, NODE JS, EXPRESS JS, TRACEROUTE, ХМАРНІ СЕРВІСИ

ЗМІСТ

ВСТУП

Актуальність. У сучасному світі, де телекомунікаційні послуги стають все більш розгалуженими та складними, якість зв'язку визначає зручність та задоволеність користувачів. Враховуючи величезну кількість мобільних пристроїв і їх незамінну роль у повсякденному житті, моніторинг якості телекомунікаційних послуг стає ключовим аспектом для забезпечення стабільного зв'язку. Розробка мобільного застосунку, який здатен ефективно моніторити ці послуги, допомагає операторам виявляти та виправляти проблеми в реальному часі, підвищуючи рівень задоволеності споживачів.

Об'єкт дослідження. Процес моніторингу якості телекомунікаційних послуг за допомогою мобільного застосунку.

Предмет дослідження. Методи та засоби для збору, аналізу та візуалізації даних про якість телекомунікаційних послуг у мобільному застосунку.

Гіпотеза. Ефективний моніторинг та аналіз якості телекомунікаційних послуг можливий завдяки розробці мобільного застосунку, який використовує сучасні методи збору даних, їх обробки та візуалізації, дозволяючи операторам швидко реагувати на виклики якості обслуговування.

Наукова новизна. У порівнянні з існуючими рішеннями, запропонований мобільний застосунок забезпечує більш детальний та комплексний аналіз якості послуг, використовуючи інноваційні методики збору та обробки даних, а також ефективну візуалізацію результатів для кращого розуміння проблемних зон у мережі.

Структура. Робота складатиметься з вступу, інформаційного огляду, вибору методів для розв’язання задачі, детального огляду програмного та інформаційного забезпечення системи, аналізу результатів та висновків, а також списку використаних джерел і додатків.

1. ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Значення моніторингу в телекомунікаціях

Моніторинг якості телекомунікаційних послуг [1] є надзвичайно важливим аспектом сучасних телекомунікаційних систем, оскільки він дозволяє забезпечити стабільність та високу якість обслуговування користувачів. У зв'язку з постійним зростанням обсягу даних та складністю мережевої інфраструктури, необхідність ефективного моніторингу стає все більш актуальною.

Моніторинг у телекомунікаціях включає в себе постійний збір та аналіз даних про стан мережі, якість зв'язку, навантаження на мережеві ресурси та інші важливі параметри. Це дозволяє операторам вчасно виявляти та вирішувати проблеми, що можуть виникнути в мережі, таким чином забезпечуючи безперервність і якість телекомунікаційних послуг.

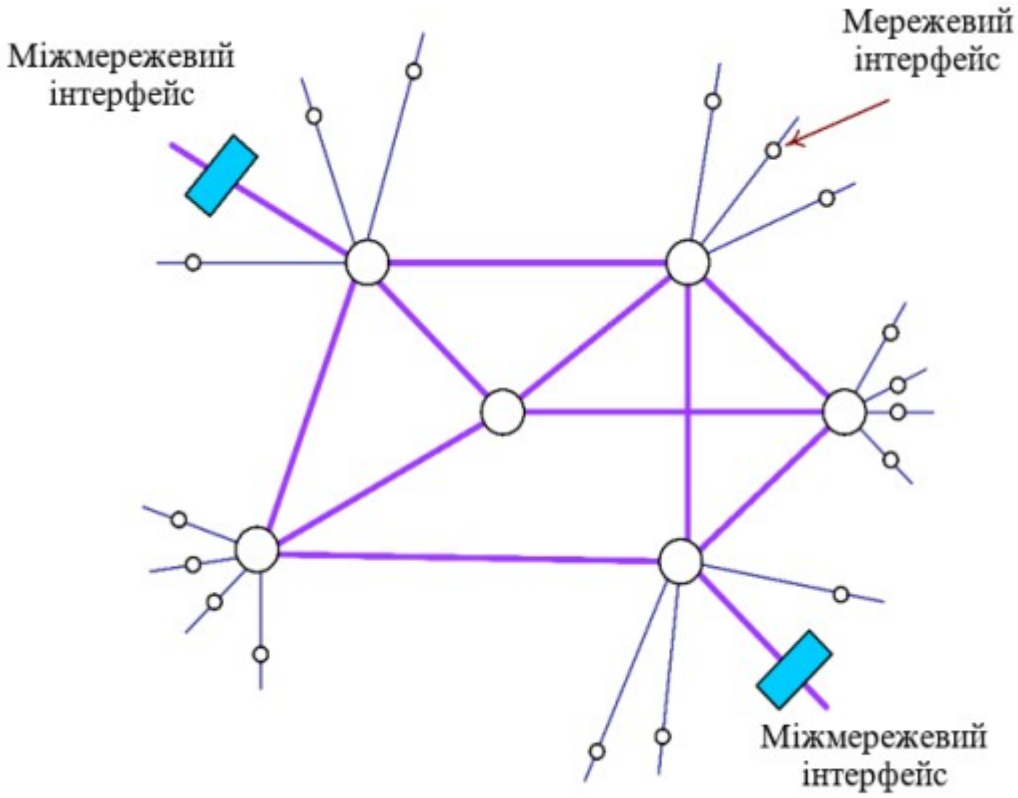


Рисунок 1.1 – Телекомунікаційна мережа [1]

Виявлення та діагностика збоїв є одними з найважливіших аспектів моніторингу телекомунікаційних мереж. Цей процес допомагає операторам та інженерам швидко реагувати на неполадки, мінімізувати час простою та забезпечити високу якість обслуговування для користувачів.

Основні компоненти виявлення та діагностики збоїв включать:

Автоматичне виявлення збоїв:

– системи моніторингу можуть автоматично виявляти відхилення від норми в роботі мережі, такі як підвищена затримка, втрати пакетів або надмірне навантаження на сервери.
– використання штучного інтелекту та машинного навчання для розпізнавання складних збоїв, які можуть не бути очевидними за допомогою традиційних методів.

Інструменти діагностики:

– розширені діагностичні інструменти дозволяють детально аналізувати проблеми, виявлені під час моніторингу. Це може включати в себе аналіз трафіку, перевірку конфігурацій обладнання та перевірку логів.
– використання автоматичних скриптів та команд для швидкого виявлення причини проблеми.

Інтеграція з системами управління мережею:

– діагностичні засоби часто інтегровані з більш широкими системами управління мережею (NMS), що дозволяє централізовано управляти всіма збоями та неполадками.
– посилання між системами моніторингу та системами керування забезпечує збір великих обсягів даних для більш точної діагностики.

Проактивне втручання та виправлення:

– виправлення помилок часто може бути автоматизовано, що зменшує ручну роботу та скорочує час відновлення після збою.

системи можуть бути налаштовані для автоматичного переконфігурування або перезавантаження обладнання при виявленні збою.

Звітність та аналіз після інциденту:

– звіти про інциденти та аналіз збоїв забезпечують знання, які можуть бути використані для покращення процесів та техніки.
– освіта та тренінги, базовані на аналізі збоїв, допомагають персоналу більш ефективно реагувати на майбутні проблеми.

Ефективна виявлення та діагностика збоїв дозволяє телекомунікаційним компаніям підтримувати надійність мережі на високому рівні та забезпечувати безперебійне обслуговування клієнтів.

За допомогою моніторингу можна вчасно прогнозувати зміни в навантаженні на мережу, що дозволяє оптимізувати розподіл ресурсів і запобігти перевантаженням. Моніторинг допомагає операторам мережі адаптуватися до змінних умов трафіку, попередньо розподіляючи необхідні ресурси та здійснюючи відповідні налаштування, що мінімізує затримки та покращує загальний досвід користувачів.

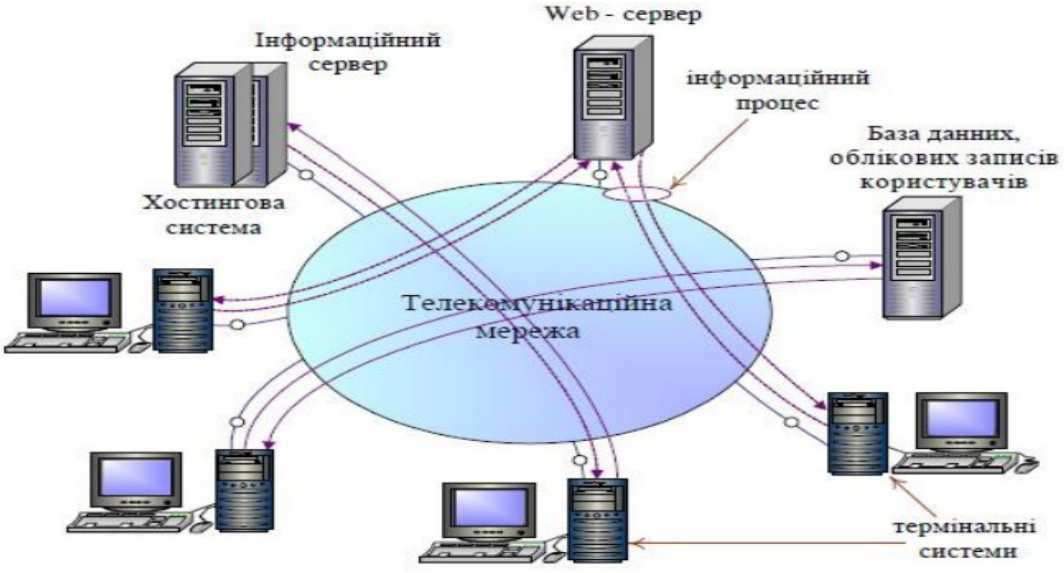


Рисунок 1.2 – Інформаційна мережа [2]

Моніторинг відіграє ключову роль у забезпеченні якості обслуговування (QoS), [3] що дозволяє підтримувати обіцяні стандарти продуктивності та надійності телекомунікаційних сервісів. Через систематичний нагляд за мережею, можна виявляти та виправляти проблеми до того, як вони вплинуть на кінцевих користувачів, що сприяє підвищенню загальної задоволеності користувачів і довіри до провайдера. Застосування сучасних інструментів моніторингу також допомагає точно вимірювати і контролювати різні параметри QoS, які включають затримку, пропускну спроможність, доступність та втрати пакетів, забезпечуючи таким чином високу якість телекомунікаційних послуг.

1.2 Стандарти якості телекомунікаційних послуг

Стандарти якості телекомунікаційних послуг встановлюють вимоги до надання телекомунікаційних послуг та забезпечення високого рівня задоволеності користувачів. Ці стандарти визначають параметри, які мають бути досягнуті і підтримувані операторами, щоб забезпечити надійність, швидкість та доступність зв'язку.

Основні міжнародні стандарти

ETSI [4] (Європейський інститут стандартів телекомунікацій):

- ETSI EG 202 009-1: Рекомендації щодо якості обслуговування (QoS) для IP-мереж.
- ETSI TS 102 250: Вимірювання якості обслуговування мобільних мереж.

ITU [5] (Міжнародний телекомунікаційний союз):

- ITU-T E.800: Визначення та метрики якості обслуговування для телекомунікаційних послуг.
- ITU-T G.1010: Вимоги до якості обслуговування користувачів для різних застосувань, включаючи голос, відео та дані.

ISO [6] (Міжнародна організація зі стандартизації):

– ISO/IEC 25010: Модель якості систем та програмного забезпечення, яка визначає характеристики та підхарактеристики якості.

Основні параметри якості телекомунікаційних послуг

Затримка (Latency)[7]:

– Час, необхідний для передачі пакету даних від джерела до призначення. Низька затримка є критичною для таких послуг, як VoIP та відеоконференції.

Пропускна здатність (Bandwidth)[8]:

– Максимальна кількість даних, яка може бути передана через мережу за одиницю часу. Висока пропускна здатність необхідна для потокового відео та інших додатків, які вимагають великої кількості даних.

Втрати пакетів (Packet Loss)[9]:

– Відсоток втрачених пакетів даних під час передачі. Низький рівень втрат є важливим для збереження якості голосового та відеозв'язку.

Варіація затримки (Jitter)[10]:

– Відхилення в затримці доставки пакетів даних. Низький рівень варіації затримки важливий для додатків реального часу, таких як VoIP.

Надійність (Reliability)[11]:

– Здатність мережі забезпечувати безперебійне надання послуг. Висока надійність означає мінімальні збої та швидке відновлення після неполадок.

Вимірювання та оцінка якості [12]

Для забезпечення відповідності стандартам якості телекомунікаційних послуг, використовуються різні методи вимірювання та оцінки:

– активний моніторинг: включає генерування трафіку та вимірювання його параметрів у реальному часі. Це дозволяє виявляти проблеми в мережі оперативно.

- пасивний моніторинг: збирає дані з реального трафіку, аналізуючи вже існуючі з'єднання. Це дозволяє оцінити якість послуг на основі фактичного використання.
- опитування користувачів: включає збір зворотного зв'язку від користувачів щодо їхнього досвіду використання телекомунікаційних послуг. Це допомагає визначити суб'єктивні аспекти якості обслуговування.

Впровадження стандартів якості

Для впровадження та підтримки стандартів якості телекомунікаційних послуг необхідно:

- розробити політики та процедури для моніторингу та управління якістю послуг.
- використовувати сучасні технології для автоматизації процесів моніторингу та аналізу даних.
- забезпечити навчання персоналу для ефективного використання інструментів моніторингу та управління.
- регулярно проводити аудит та оцінку відповідності стандартам якості, вносячи необхідні корективи для поліпшення обслуговування.

Таким чином, стандарти якості телекомунікаційних послуг є основою для забезпечення високої якості зв'язку та задоволеності користувачів. Використання сучасних технологій та методів моніторингу дозволяє операторам підтримувати належний рівень обслуговування та оперативно реагувати на зміни в мережі.

1.3 Підходи до моніторингу та аналізу даних

Моніторинг та аналіз даних [13] є критично важливими для забезпечення високої якості телекомунікаційних послуг. Вони дозволяють виявляти про-

блеми, оптимізувати роботу мережі та забезпечувати відповідність стандартам якості. Існує кілька основних підходів до моніторингу та аналізу даних у телекомунікаціях:

Активний моніторинг передбачає генерування трафіку спеціально для вимірювання параметрів мережі. Це дозволяє отримувати детальну інформацію про продуктивність та якість зв'язку в режимі реального часу.

Переваги такого моніторингу в тому, що можливо виявити проблему у реальному часі. Висока точність вимірювань. Здатність моделювати різні сценарії використання.

Недоліки полягають в створенні додаткового навантаження на мережу. Вимагання спеціального обладнання та програмного забезпечення.

Пасивний моніторинг базується на зборі та аналізі даних з уже існуючого трафіку в мережі. Це дозволяє оцінити реальну продуктивність та якість зв'язку без створення додаткового навантаження.

Переваги в тому, що не створює додаткового трафіку. Дозволяє аналізувати фактичне використання мережі. Менш витратний, ніж активний моніторинг.

Недоліки: Менш точний у порівнянні з активним моніторингом. Виявлення проблем може займати більше часу.

Гібридний підхід комбінує елементи активного та пасивного моніторингу, дозволяючи використовувати переваги обох методів. Це забезпечує більш повне розуміння продуктивності та якості мережі.

Переваги: поєднує точність активного моніторингу з реалістичністю пасивного. Забезпечує більш повну картину стану мережі. Гнучкість у налаштуванні та використанні.

Недоліки: вимагає більшої складності в налаштуванні та управлінні. Може бути дорожчим у реалізації.

Аналіз даних є невід'ємною частиною процесу моніторингу. Він дозволяє перетворювати зібрані дані на корисну інформацію для прийняття рішень. Основні підходи до аналізу даних включають:

Дескриптивний аналіз: описує поточний стан мережі на основі зібраних даних. Використовується для виявлення тенденцій та аномалій.

Діагностичний аналіз: визначає причини проблем, виявлених під час моніторингу. Допомагає розробляти заходи щодо усунення виявлених недоліків.

Предиктивний аналіз: використовує історичні дані для прогнозування майбутніх проблем та навантажень на мережу. Допомагає планувати розвиток мережі та запобігати можливим збоям.

Прескриптивний аналіз: пропонує конкретні дії для оптимізації роботи мережі на основі прогнозів та поточних даних. Використовується для автоматизації процесів управління та оптимізації.

Інструменти для моніторингу та аналізу даних

Для ефективного моніторингу та аналізу даних у телекомунікаційних мережах використовуються різні програмні та апаратні засоби:

- системи управління мережею (NMS) [14]: Забезпечують централізоване управління та моніторинг мережевих компонентів.
- програмні агенти: збирають дані з різних точок мережі та передають їх на центральний сервер для аналізу.
- аналітичні платформи: використовують алгоритми машинного навчання та штучного інтелекту для обробки великих обсягів даних та виявлення прихованих закономірностей.

Таким чином, підходи до моніторингу та аналізу даних відіграють ключову роль у забезпеченні високої якості телекомунікаційних послуг. Використання сучасних методів та інструментів дозволяє операторам своєчасно виявляти та усувати проблеми, забезпечуючи стабільну та надійну роботу мережі.

1.4 Використання хмарних сервісів для моніторингу

Використання хмарних сервісів [15] для моніторингу телекомунікаційних мереж стає все більш поширеним завдяки їх численним перевагам. Хмарні сервіси надають можливість централізованого управління, масштабування та аналізу даних, що робить їх ефективним рішенням для моніторингу якості телекомунікаційних послуг.

Переваги використання хмарних сервісів

Масштабованість:

– хмарні платформи дозволяють легко масштабувати обчислювальні ресурси відповідно до потреб моніторингу. Це особливо важливо для великих телекомунікаційних мереж, де обсяг даних може суттєво змінюватись.

Гнучкість:

– хмарні сервіси забезпечують гнучкість у налаштуванні та управлінні інфраструктурою моніторингу. Вони дозволяють швидко адаптуватись до змін у мережі та вимог бізнесу.

Централізоване управління:

– використання хмарних сервісів дозволяє централізувати управління та моніторинг мережевих ресурсів. Це спрощує процес управління та забезпечує більш ефективний контроль за якістю послуг.

Зниження витрат:

– хмарні сервіси дозволяють знизити капітальні витрати на обладнання та інфраструктуру. Оплата за використання ресурсів на основі підписки або фактичного використання допомагає оптимізувати витрати.

Доступність та мобільність:

– дані та аналітичні інструменти доступні з будь-якого місця та пристрою, що забезпечує мобільність і зручність для адміністраторів мережі.

Основні хмарні платформи для моніторингу

Amazon Web Services (AWS) [16]:

– Amazon CloudWatch: забезпечує моніторинг ресурсів AWS та додатків, які на них працюють. Надає метрики, журнали та тривоги в реальному часі.
– AWS X-Ray: дозволяє аналізувати та налагоджувати розподілені додатки, візуалізуючи взаємодію між компонентами.

Microsoft Azure [17]:

– Azure Monitor: пропонує комплексні рішення для моніторингу додатків, інфраструктури та мережі. Надає можливості аналітики та попередження про проблеми.
– Azure Log Analytics: збирає та аналізує журнали з різних джерел для виявлення та усунення проблем.

Google Cloud Platform (GCP) [18]:

– Google Stackdriver: надає моніторинг, логування та діагностику для додатків, що працюють у GCP та AWS. Забезпечує централізований огляд стану системи.

IBM Cloud [19]:

– IBM Cloud Monitoring with Sysdig: пропонує моніторинг продуктивності та безпеки контейнерів, додатків та інфраструктури. Включає можливості аналізу даних в реальному часі.

DigitalOcean [20]:

– Monitoring and Alerts: DigitalOcean пропонує вбудовані засоби для моніторингу використання ресурсів, таких як CPU, пам'ять, та дискове використання, із можливістю налаштування сповіщень.

Salesforce [21]:

– Einstein Analytics: сервіс для глибокого аналізу даних, що включає моніторинг використання Salesforce та аналіз клієнтської взаємодії для підвищення ефективності бізнесу.

SAP Cloud Platform [22]:

– SAP Cloud ALM: інструмент для управління життєвим циклом додатків, який забезпечує моніторинг та аналіз середовища SAP, включаючи продуктивність застосунків і ресурси.

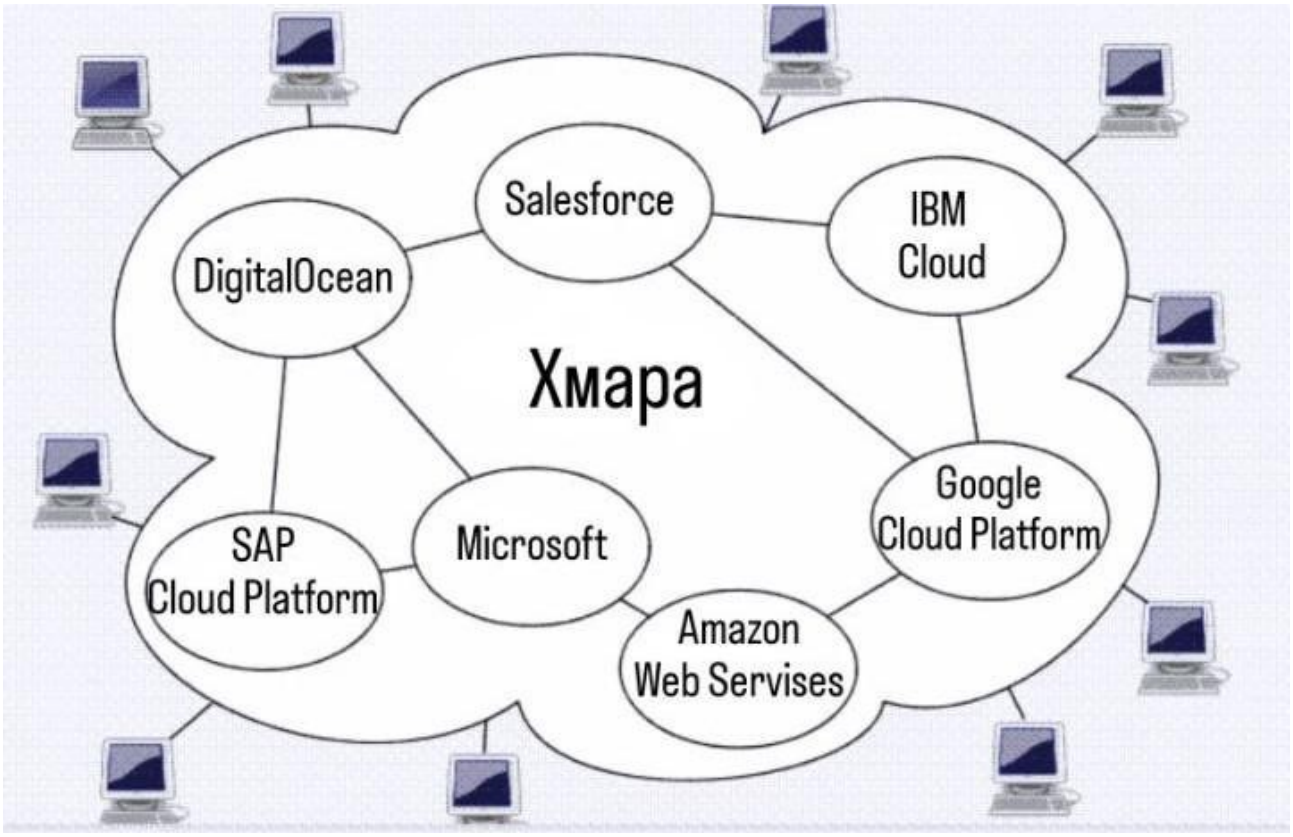


Рисунок1.3 – Хмарні платформи для моніторину

Впровадження хмарних сервісів для моніторингу вимагає ретельного планування та підготовки. Основні кроки включають:

1. Оцінка потреб:

– визначення вимог до моніторингу, обсягу даних та необхідних ресурсів.

– аналіз існуючої інфраструктури та можливостей її інтеграції з хмарними сервісами.

2. Вибір платформи:

– оцінка різних хмарних платформ з урахуванням їх можливостей, вартості та сумісності з існуючою інфраструктурою.

3. Налаштування та конфігурація:

– налаштування обраної хмарної платформи для збору, зберігання та аналізу даних.

– конфігурація метрик, журналів та тривог для забезпечення ефективного моніторингу.

4. Тестування та оптимізація:

– проведення тестування налаштованої системи моніторингу.

– оптимізація конфігурації на основі отриманих результатів та зворотного зв’язку.

5. Навчання персоналу:

– забезпечення навчання для адміністраторів та інженерів, відповідальних за управління системою моніторингу.

– підтримка постійного навчання для адаптації до нових технологій та методів.

Виклики та перспективи

Використання хмарних сервісів для моніторингу телекомунікаційних мереж також має свої виклики:

Безпека та конфіденційність:

– забезпечення захисту даних в хмарному середовищі.

– дотримання нормативних вимог та стандартів безпеки.

Залежність від провайдера:

– вразливість до проблем, які можуть виникати у хмарного провайдера.

– необхідність планування на випадок відмови або зміни провайдера.

Інтеграція з існуючими системами:

– забезпечення сумісності та безперебійної роботи з наявними інструментами та інфраструктурою.

Проте, перспективи використання хмарних сервісів для моніторингу є надзвичайно позитивними завдяки їхнім перевагам у гнучкості, масштабованості та ефективності. Впровадження цих технологій дозволяє операторам забезпечити високу якість телекомунікаційних послуг та оперативно реагувати на зміни в мережі.

1.5 Методи збору та аналізу телеметрії в мобільних застосунках

Телеметрія [23] є критичним компонентом для забезпечення високої якості послуг мобільних застосунків. Вона дозволяє збирати та аналізувати дані про стан системи, продуктивність та поведінку користувачів у реальному часі. Це дозволяє операторам мобільних мереж та розробникам застосунків швидко виявляти та вирішувати проблеми, а також покращувати загальну якість обслуговування. Методи збору телеметрії

Локальне збирання даних:

Вбудовані логи: Збір логів безпосередньо на пристрої користувача. Це можуть бути журнали подій, помилки та інша діагностична інформація.

API [24] збору даних: Використання спеціальних API для збору метрик з мобільного застосунку, таких як час завантаження сторінок, частота виникнення помилок, тривалість сесій користувачів тощо.

Збір даних в реальному часі:

Аналітичні інструменти: інтеграція з аналітичними платформами, такими як Google Analytics [25], Firebase [26], які надають можливість збору та аналізу даних у реальному часі.

Системи моніторингу продуктивності: використання інструментів, таких як New Relic [27], AppDynamics [28] для збору даних про продуктивність застосунків в реальному часі.

Телеметричні сервери:

Обробка подій: надсилання зібраних даних на сервери для подальшої обробки та аналізу. Сервери можуть використовуватися для агрегації та зберігання телеметричних даних.

Системи зберігання великих даних: використання хмарних платформ для зберігання та обробки великих обсягів телеметричних даних, наприклад, AWS S3[29], Google Cloud Storage [30].

Методи аналізу телеметричних даних

Аналіз продуктивності:

Метрики продуктивності: аналіз часу відгуку, частоти помилок, використання пам'яті та процесора для виявлення вузьких місць у роботі застосунку.

Трасування: використання трасування для детального аналізу шляху виконання запитів та виявлення проблемних місць.

Аналіз поведінки користувачів:

Поведінковий аналіз: вивчення моделей поведінки користувачів, таких як частота використання функцій, відсоток завершених дій, точки виходу з застосунку.

Когортний аналіз: аналіз груп користувачів, об'єднаних за певними характеристиками, для розуміння змін в поведінці з часом.

Аналіз подій:

подієвий аналіз: відстеження та аналіз ключових подій у застосунку, таких як натискання кнопок, завершення транзакцій, виникнення помилок.

кореляційний аналіз: виявлення взаємозв'язків між різними подіями та метриками для визначення причин проблем.

Прогнозна аналітика:

Машинне навчання: використання алгоритмів машинного навчання для прогнозування майбутніх проблем та поведінки користувачів на основі історичних даних.

Аналіз трендів: визначення трендів у використанні застосунку та продуктивності для прийняття обґрунтованих рішень.

Виклики та рішення

Забезпечення приватності даних:

Анонімізація: використання методів анонімізації для захисту особистих даних користувачів під час збору та аналізу телеметричних даних.

Політики конфіденційності: впровадження чітких політик конфіденційності та забезпечення їх дотримання.

Інтеграція з існуючими системами:

Сумісність: забезпечення сумісності з існуючими інструментами та системами для безперебійної інтеграції.

Стандартизація: використання стандартних протоколів та форматів даних для полегшення інтеграції.

Обробка великих обсягів даних:

Масштабовані системи: використання хмарних платформ та розподілених систем для обробки великих обсягів телеметричних даних.

Ефективні алгоритми: розробка та використання ефективних алгоритмів для аналізу даних у реальному часі.

Методи збору та аналізу телеметрії в мобільних застосунках є критичними для забезпечення високої якості телекомунікаційних послуг. Використання сучасних інструментів та хмарних сервісів дозволяє ефективно збирати, обробляти та аналізувати дані, що сприяє підвищенню продуктивності та задоволеності користувачів.

1.6 Постановка задачі

Метою роботи є розробка мобільного застосунку для моніторингу якості телекомунікаційних послуг для підрозділу Сумського державного університету. Ця робота входить у склад практичного застосування технологій мобільної розробки, аналізу даних та відповідає вимогам студентів бакалавратури зі спеціальності "Комп'ютерні науки".

Застосунок має забезпечувати наступні функціональні можливості:

- збір даних про якість телекомунікаційних послуг у реальному часі.
- моніторинг основних показників якості, таких як швидкість з'єднання, затримка, втрата пакетів та стабільність з'єднання.
- аналіз зібраних даних для виявлення проблем та генерація звітів.
- надання користувачам інструментів для перегляду історії якості послуг та отримання рекомендацій щодо поліпшення.
- для досягнення поставлених цілей необхідно виконати наступні завдання:
- визначити оптимальну архітектуру для розробки мобільного застосунку з функціями моніторингу.
- обрати технології для збору, обробки та аналізу даних про якість телекомунікаційних послуг.
- реалізувати мобільний застосунок, використовуючи сучасні інструменти розробки мобільних додатків.
- протестувати систему збору та аналізу даних, а також її здатність до моніторингу телекомунікаційних послуг у реальному часі.

Предметом дослідження є методи та інструменти збору, аналізу та моніторингу даних про якість телекомунікаційних послуг у мобільному застосунку.

Наукова новизна полягає в розробці ефективної системи моніторингу якості телекомунікаційних послуг у мобільному середовищі, що використовує сучасні методи аналізу даних та забезпечує реальний час зворотного зв’язку.

Практичне значення полягає в підвищенні якості телекомунікаційних послуг для користувачів завдяки можливості швидкого виявлення та усунення проблем, що забезпечує кращу задоволеність користувачів та покращення роботи телекомунікаційних мереж.

2. ВИБІР МЕТОДІВ РОЗВ'ЯЗАННЯ ЗАДАЧІ

2.1 Інструменти для збору даних моніторингу

Збір даних моніторингу є ключовим етапом для забезпечення ефективного контролю за якістю телекомунікаційних послуг. У цьому розділі розглянемо основні інструменти, які можуть бути використані для збору даних моніторингу в мобільному застосунку.

1. NetTest:
   - Опис: NetTest [31] – це популярний інструмент для вимірювання швидкості інтернет-з'єднання, який може бути інтегрований у мобільні додатки.
   - Функціональність: збір даних про швидкість завантаження та вивантаження, затримку, втрата пакетів.
   - Переваги: простота інтеграції, точні результати вимірювань.
2. OpenSignal:
   - Опис: OpenSignal [32]– це інструмент для аналізу покриття мобільної мережі та вимірювання якості з'єднання.
   - Функціональність: відстеження якості сигналу, швидкість з'єднання, розташування веж стільникового зв'язку.
   - Переваги: велика база даних про якість сигналу в різних регіонах, можливість порівняння з іншими користувачами.
3. Network Diagnostic Tool (NDT):
   - Опис: NDT [33] – це інструмент для діагностики мереж, що дозволяє вимірювати різні параметри інтернет-з'єднання.
   - Функціональність: аналіз пропускної здатності, затримки, втрата пакетів, тривалість з'єднання.
   - Переваги: глибокий аналіз мережі, зручний інтерфейс для розробників.

4. Speedtest by Ookla:

– Опис: Speedtest by Ookla [34] – широко відомий інструмент для вимірювання швидкості інтернет-з'єднання.
– Функціональність: вимірювання швидкості завантаження та вивантаження, затримки.
– Переваги: висока точність, велика кількість серверів для тестування по всьому світу.

5. TelephonyManager (Android):

– Опис: TelephonyManager [35] – це стандартний інструмент Android SDK для отримання інформації про стан мобільної мережі.
– Функціональність: інформація про мережу, сигнал, ідентифікатори мережі, стан з'єднання.
– Переваги: безпосередній доступ до даних мережі, підтримка багатьох параметрів.

6. NetworkInfo (iOS):

– Опис: NetworkInfo [36] – це інструмент для iOS, який надає інформацію про стан мережі.
– Функціональність: інформація про підключення, тип мережі, стан сигналу.
– Переваги: легка інтеграція з iOS-додатками, доступ до основних параметрів мережі.

3. Prometheus & Grafana:

– Опис: Prometheus [37] – це система моніторингу та оповіщення з відкритим кодом, а Grafana – інструмент для візуалізації метрик.
– Функціональність: збір та зберігання телеметрії, створення дашбордів для візуалізації даних.
– Переваги: гнучкість, розширюваність, інтеграція з багатьма іншими системами.

Для досягнення оптимальних результатів у розробці мобільного застосунку для моніторингу якості телекомунікаційних послуг, рекомендується використовувати комбінацію кількох інструментів. Це дозволить забезпечити більш повне та точне збирання даних про якість з'єднання, що сприятиме підвищенню якості телекомунікаційних послуг для кінцевих користувачів.

2.2 Інструменти аналізу телеметрії

Аналіз телеметрії є важливим етапом у процесі моніторингу якості телекомунікаційних послуг. Цей процес включає збір, обробку та аналіз великих обсягів даних для виявлення проблем, оцінки продуктивності та забезпечення високої якості обслуговування. У цьому розділі розглянемо основні інструменти, які можуть бути використані для аналізу телеметрії в мобільному застосунку.

1. ELK Stack (Elasticsearch, Logstash, Kibana):

Опис: ELK Stack [38] – це потужна платформа для пошуку, аналізу та візуалізації даних в реальному часі.

Функціональність:

- Elasticsearch: пошук та індексація великих обсягів даних.
- Logstash: збір, обробка та перетворення даних з різних джерел.
- Kibana: візуалізація даних за допомогою дашбордів та графіків.

Переваги: гнучкість, масштабованість, підтримка різноманітних форматів даних.

2. Prometheus:

Опис: Prometheus – це система моніторингу та оповіщення з відкритим кодом, що спеціалізується на збиранні та аналізі метрик.

Функціональність:

– Збір телеметричних даних за допомогою клієнтських бібліотек та експортерів.
– Зберігання даних у вигляді часових рядів.
– Підтримка запитів на мові PromQL для аналізу даних.

Переваги: висока продуктивність, підтримка контейнеризованих середовищ, гнучкість у налаштуванні.

3. Grafana:

Опис: Grafana – це інструмент для візуалізації метрик та побудови інтерактивних дашбордів.

Функціональність:

– Підключення до різних джерел даних, включаючи Prometheus, Elasticsearch, InfluxDB.
– Створення кастомних дашбордів для моніторингу та аналізу.
– Підтримка оповіщень на основі заданих умов.

Переваги: інтуїтивно зрозумілий інтерфейс, багаті можливості для візуалізації, інтеграція з багатьма системами.

4. Splunk:

Опис: Splunk – це комерційна платформа для моніторингу, пошуку та аналізу машинних даних.

Функціональність:

– Збір та індексація телеметричних даних.
– Потужні інструменти для пошуку та аналізу.
– Інтерактивні дашборди та звіти.

Переваги: висока продуктивність, багатий функціонал, підтримка великих обсягів даних.

5. Apache Kafka:

Опис: Apache Kafka – це платформа потокової обробки даних, що дозволяє збирати, обробляти та аналізувати дані в реальному часі.

Функціональність:

- Підтримка високопродуктивного обміну повідомленнями.
- Потокова обробка даних за допомогою Kafka Streams та інших інструментів.
- Інтеграція з різними джерелами та сховищами даних.

Переваги: висока масштабованість, низька затримка, можливість обробки даних у реальному часі.

6. Fluentd:

Опис: Fluentd – це інструмент для збору, обробки та передачі логів і телеметричних даних.

Функціональність:

- Збір даних з різних джерел.
- Обробка та перетворення даних перед їх передачею до сховищ.
- Підтримка численних плагінів для інтеграції з іншими системами.

Переваги: гнучкість, масштабованість, проста інтеграція з різними джерелами даних.

Для досягнення найкращих результатів у аналізі телеметрії мобільного застосунку рекомендується використовувати комбінацію кількох інструментів. Це дозволить забезпечити більш точний і повний аналіз даних, що сприятиме підвищенню якості телекомунікаційних послуг для користувачів.

2.3 Платформи для обробки даних у реальному часі

Обробка даних у реальному часі є критично важливою для мобільного застосунку, що моніторить якість телекомунікаційних послуг. Ця обробка дозволяє оперативно реагувати на зміни у мережевій інфраструктурі, виявляти проблеми і забезпечувати високий рівень обслуговування. У цьому розділі розглянемо основні платформи, які можна використовувати для обробки даних у реальному часі.

1. Apache Kafka:

Опис: Apache Kafka [39] – це розподілена стрімінгова платформа, що дозволяє збирати, обробляти та зберігати великі обсяги даних у режимі реального часу.

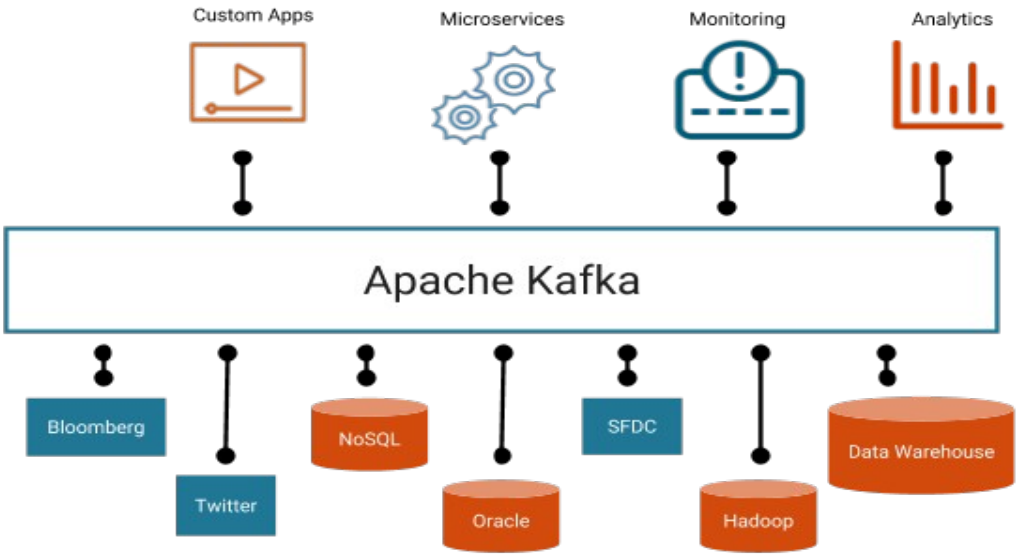


Рисунок 2.1 – Apache Kafka Architecture [40]

Функціональність:

– Надійне зберігання та передача повідомлень.
– Високопродуктивна обробка даних у реальному часі за допомогою Kafka Streams.
– Масштабованість для обробки великих обсягів даних.

Переваги: висока пропускна здатність, низька затримка, підтримка інтеграції з іншими інструментами.

2. Apache Flink:

Опис: Apache Flink [41] – це платформа для потокової обробки даних, що забезпечує високий рівень продуктивності та низьку затримку.

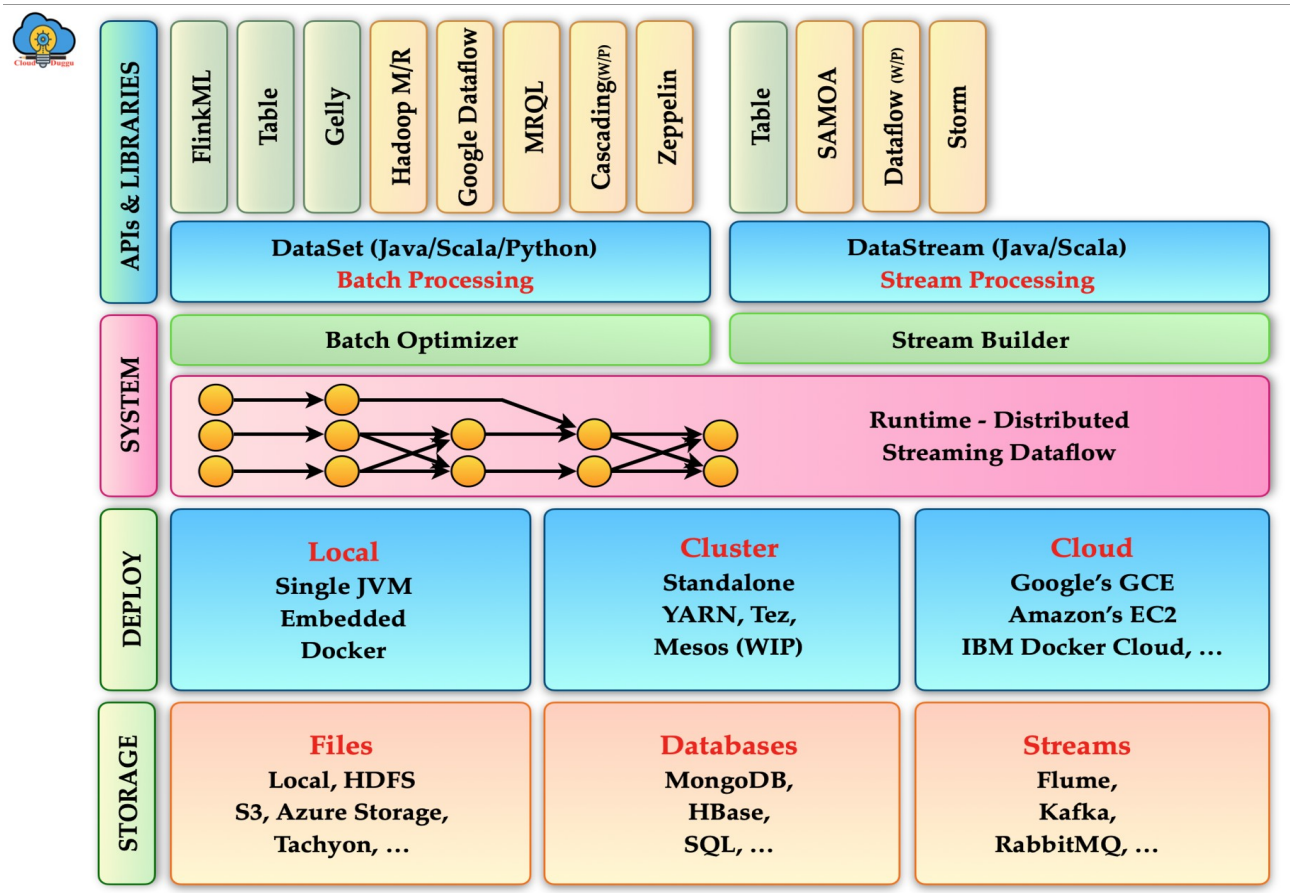


Рисунок 2.2 – Apache Flink Architecture [42]

Функціональність:

– Підтримка обробки даних у реальному часі та пакетної обробки.
– Складні аналітичні можливості для обробки потоків даних.
– Висока надійність та масштабованість.

Переваги: потужні інструменти для аналізу даних, гнучкість у налаштуванні, інтеграція з багатьма джерелами даних.

3. Apache Storm:

Опис: Apache Storm [43] – це розподілена система для обробки потоків даних у реальному часі, що забезпечує високу швидкість та надійність.

Функціональність:

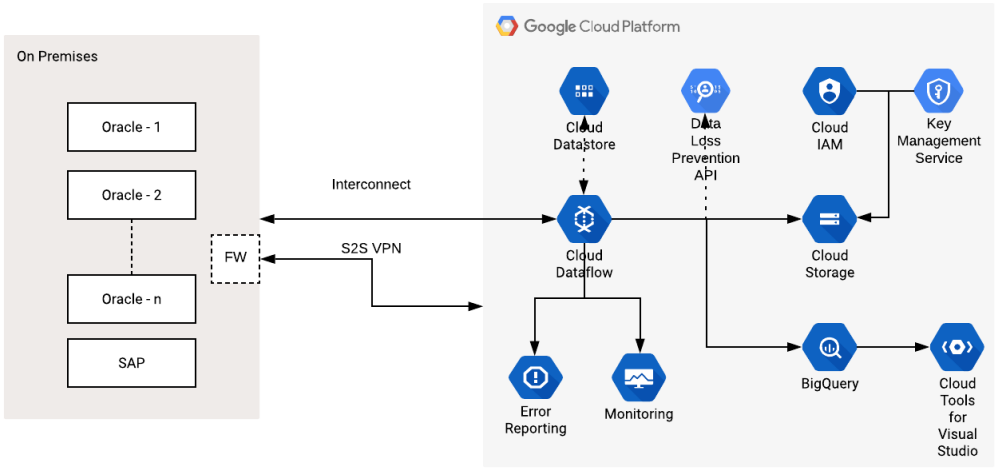
– Обробка даних у реальному часі з низькою затримкою.
– Підтримка складних обчислювальних завдань на потоках даних.

– Гнучкість у налаштуванні та масштабованість.

Переваги: висока продуктивність, низька затримка, надійність у роботі з великими обсягами даних.

4. Google Cloud Dataflow:

Опис: Google Cloud Dataflow [44] – це хмарна платформа для потокової та пакетної обробки даних, що забезпечує автоматичне масштабування та високу продуктивність.



Рисуноку 2.3 – Understanding the Google Cloud Dataflow Model [45]

Функціональність:

– Обробка даних у режимі реального часу та пакетної обробки.

– Підтримка складних обчислювальних завдань.

– Інтеграція з іншими сервісами Google Cloud.

Переваги: автоматичне масштабування, висока надійність, простота у використанні.

5. AWS Kinesis:

Опис: AWS Kinesis [46] – це сервіс для потокової обробки великих обсягів даних у реальному часі, що входить до складу Amazon Web Services.

Функціональність:

– Надійне збирання та обробка потокових даних.
– Підтримка складних аналітичних завдань у режимі реального часу.
– Масштабованість для обробки великих обсягів даних.

Переваги: висока продуктивність, інтеграція з іншими сервісами AWS, гнучкість у налаштуванні.

6. Microsoft Azure Stream Analytics:

Опис: Microsoft Azure Stream Analytics [47] – це сервіс для обробки потокових даних у реальному часі, що надає широкий спектр аналітичних можливостей.

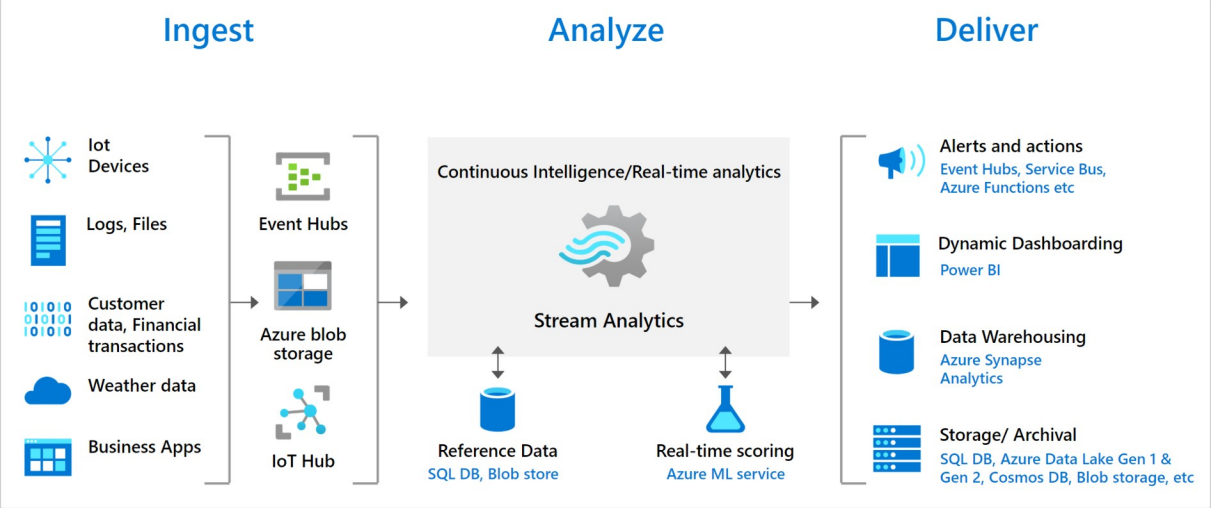


Рисунок 2.4 – Azure Stream Analytics [48]

Функціональність:

– Обробка даних у реальному часі.
– Підтримка складних аналітичних запитів.
– Інтеграція з іншими сервісами Azure.

Переваги: простота у використанні, висока надійність, інтеграція з екосистемою Azure.

Вибір платформи для обробки даних у реальному часі залежить від конкретних потреб та вимог проекту. Всі розглянуті платформи мають свої

переваги та можуть бути використані для забезпечення ефективного моніторингу та аналізу якості телекомунікаційних послуг у мобільному застосунку.

2.4 Інструменти для забезпечення якості передачі даних

Забезпечення якості передачі даних є критично важливим для мобільного застосунку, що моніторить якість телекомунікаційних послуг. Якість передачі даних визначає, наскільки ефективно та надійно користувачі можуть отримувати та передавати інформацію через телекомунікаційні мережі.

1. Wireshark:

Опис: Wireshark [49] – це популярний аналізатор мережевого трафіку, що дозволяє захоплювати та аналізувати пакети даних у реальному часі.

Функціональність:

- Захоплення та інспекція пакетів даних на різних рівнях мережевої моделі.
- Аналіз протоколів і виявлення проблем у передачі даних.
- Деталізовані звіти та візуалізація мережевого трафіку.

Переваги: безкоштовний, потужний інструмент з широкими аналітичними можливостями, підтримка великої кількості протоколів.

2. iPerf:

Опис: iPerf [50] – це інструмент для вимірювання пропускної здатності мережі, що дозволяє оцінити якість з'єднання між двома кінцевими точками.

Функціональність:

- Тестування пропускної здатності TCP, UDP та SCTP з'єднань.
- Налаштування параметрів тестування, таких як тривалість тесту, розмір буфера та інше.
- Виведення детальних результатів про пропускну здатність та затримки.

Переваги: простий у використанні, точні вимірювання, підтримка різних типів з'єднань.

3. PingPlotter:

Опис: PingPlotter [51] – це інструмент для діагностики мережевих проблем, що використовує команди ping та traceroute для аналізу з'єднань.

Функціональність:

– Вимірювання часу відгуку та втрачених пакетів на кожному етапі маршруту.
– Візуалізація результатів у вигляді графіків та діаграм.
– Ідентифікація вузьких місць і проблем у мережевій інфраструктурі.

Переваги: інтуїтивний інтерфейс, наочні візуалізації, корисні для швидкого виявлення проблем.

4. NetFlow Analyzer:

Опис: NetFlow Analyzer [52] – це інструмент для аналізу мережевого трафіку, що дозволяє детально моніторити потоки даних у мережі.

Функціональність:

– Збір і аналіз даних про трафік з підтримкою протоколів NetFlow, sFlow, IPFIX тощо.
– Моніторинг пропускної здатності, визначення джерел навантаження та аналіз аномалій.
– Генерація звітів про мережеву активність і виявлення проблем.

Переваги: детальний аналіз трафіку, можливість інтеграції з іншими мережевими інструментами, підтримка різних протоколів.

5. SolarWinds Network Performance Monitor (NPM):

Опис: SolarWinds NPM [53] – це комплексний інструмент для моніторингу продуктивності мережі, що забезпечує повний огляд стану мережевої інфраструктури.

Функціональність:

– Моніторинг часу відгуку, доступності та продуктивності мережевих пристроїв.

– Виявлення проблем у реальному часі та сповіщення про них.
– Візуалізація мережевих карт і створення детальних звітів.

Переваги: широкі можливості моніторингу, інтеграція з іншими продуктами SolarWinds, потужні інструменти для аналізу продуктивності.

6. Nagios:

Опис: Nagios [54] – це система для моніторингу мережевих сервісів, що дозволяє відслідковувати стан і продуктивність мережевих ресурсів.

Функціональність:

– Моніторинг доступності серверів, мережевих пристроїв і сервісів.
– Сповіщення про проблеми та їх візуалізація.
– Генерація звітів про стан мережевої інфраструктури.

Переваги: відкрите програмне забезпечення, гнучкість у налаштуванні, велика кількість плагінів.

Вибір інструментів для забезпечення якості передачі даних залежить від конкретних потреб і вимог проекту. Всі розглянуті інструменти мають свої особливості та переваги, що дозволяють ефективно моніторити та забезпечувати високу якість телекомунікаційних послуг у мобільному застосунку.

2.5 Огляд хмарних провайдерів для моніторингу телекомунікацій

Використання хмарних сервісів для моніторингу телекомунікаційних послуг надає ряд переваг, таких як масштабованість, надійність та гнучкість. Розглянемо провідних хмарних провайдерів, які пропонують інструменти для моніторингу телекомунікаційних мереж і сервісів.

1. Amazon Web Services (AWS):

Опис: AWS – один з найбільших хмарних провайдерів, який пропонує широкий спектр сервісів для моніторингу та управління телекомунікаційними мережами.

Основні інструменти:

- Amazon CloudWatch: інструмент для моніторингу і управління ресурсами та додатками, що працюють в AWS. Підтримує збір метрик, логів та подій.
- AWS X-Ray: сервіс для аналізу та налагодження розподілених додатків, який допомагає відслідковувати запити і визначати вузькі місця у продуктивності.

Переваги: інтеграція з іншими сервісами AWS, масштабованість, високий рівень безпеки.

2. Microsoft Azure:

Опис: Azure – хмарна платформа від Microsoft, яка пропонує комплексні рішення для моніторингу телекомунікаційних мереж.

Основні інструменти:

- Azure Monitor: сервіс для збору та аналізу даних про продуктивність і використання ресурсів. Підтримує створення дашбордів та опові-щень.
- Azure Network Watcher: Інструмент для моніторингу, діагностики та візуалізації мережевої інфраструктури в Azure.

Переваги: глибока інтеграція з Windows Server та іншими продуктами Microsoft, висока масштабованість, підтримка гібридних рішень.

3. Google Cloud Platform (GCP):

Опис: GCP – хмарна платформа від Google, яка надає інструменти для моніторингу та управління телекомунікаційними мережами.

Основні інструменти:

- Google Cloud Monitoring: сервіс для збору метрик, логів та подій з додатків і ресурсів, що працюють в GCP та інших середовищах.
- Google Cloud Trace: інструмент для аналізу продуктивності додатків та визначення затримок у виконанні запитів.

Переваги: інтеграція з іншими сервісами Google, потужні аналітичні інструменти, висока продуктивність.

4. IBM Cloud:

Опис: IBM Cloud пропонує різноманітні інструменти для моніторингу телекомунікаційних мереж та сервісів.

Основні інструменти:

- IBM Cloud Monitoring: сервіс для моніторингу продуктивності додатків і ресурсів, що працюють в IBM Cloud.
- IBM Cloud Pak for Watson AIOps: інструмент, який використовує штучний інтелект для автоматизації операцій та моніторингу IT-інфраструктури.

Переваги: підтримка гібридних та мультихмарних рішень, використання AI для автоматизації, висока надійність.

5. Oracle Cloud:

Опис: Oracle Cloud надає комплексні рішення для моніторингу та управління телекомунікаційними мережами.

Основні інструменти:

- Oracle Cloud Infrastructure (OCI) Monitoring: сервіс для моніторингу метрик та створення оповіщень для ресурсів, що працюють в OCI.
- Oracle Management Cloud: платформа для моніторингу, аналізу логів та управління продуктивністю додатків.

Переваги: інтеграція з базами даних Oracle, висока безпека, потужні аналітичні можливості.

Вибір хмарного провайдера для моніторингу телекомунікаційних послуг залежить від конкретних потреб проекту, таких як вимоги до продуктивності, інтеграції з існуючими системами, бюджету та інших факторів. Кожен з розглянутих провайдерів пропонує свої унікальні інструменти та переваги, які до-

зволяють ефективно забезпечувати моніторинг і управління телекомунікаційними мережами.

3. ІНФОРМАЦІЙНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

3.1 Технології розробки мобільного застосунку для моніторингу

Розробка мобільного застосунку для моніторингу якості телекомунікаційних послуг має на меті забезпечити користувачів інструментом для ефективного відстеження та аналізу стану послуг у режимі реального часу. Застосунок має інтуїтивно зрозумілий інтерфейс, що дозволяє користувачам легко навігувати та отримувати необхідні дані.

Розробка включає використання кількох ключових технологій, які забезпечують високу ефективність та надійність системи.

React Native є фреймворком, який дозволяє створювати кросплатформені мобільні застосунки з використанням JavaScript та React. Він дозволяє розробникам писати код один раз і запускати його на обох популярних мобільних платформах - iOS та Android. React Native використовується для створення інтерфейсу користувача застосунку, що забезпечує зручність використання та інтуїтивно зрозумілу навігацію. Це дозволяє користувачам легко перевіряти якість своїх телекомунікаційних послуг, переглядати результати тестів та отримувати рекомендації.

JavaScript є мовою програмування, яка використовується для створення динамічних і інтерактивних веб-сторінок. У застосунку JavaScript використовується для логіки клієнтської частини, обробки подій та взаємодії з сервером. Використання JavaScript дозволяє забезпечити швидке реагування застосунку на дії користувачів та обробку великих обсягів даних.

TypeScript є надмножиною JavaScript, яка додає статичну типізацію до мови. Це допомагає розробникам виявляти помилки ще на етапі розробки, що робить код більш надійним і легким у підтримці. У нашому проекті TypeScript використовується для написання більш структурованого та зрозумілого коду, особливо для великих компонентів та модулів.

Express.js є мінімалістичним веб-фреймворком для Node.js, який полегшує створення серверних додатків та API. Express.js використовується для створення серверної частини застосунку, яка обробляє запити від клієнтської частини, взаємодіє з базою даних та зовнішніми сервісами. Це забезпечує стабільну роботу серверу та швидку обробку запитів.

Node.js є середовищем виконання JavaScript на стороні сервера, яке дозволяє запускати JavaScript-код поза браузером. Node.js використовується для серверної логіки, обробки запитів, управління базами даних та інших серверних операцій. Використання Node.js дозволяє забезпечити високу продуктивність та масштабованість серверної частини застосунку.

RESTful API є архітектурним стилем для створення веб-сервісів, який використовує HTTP-запити для доступу та маніпулювання даними. RESTful API використовується для забезпечення взаємодії між клієнтською та серверною частинами. Це дозволяє клієнтському застосунку отримувати дані про якість телекомунікаційних послуг, відправляти результати тестів та отримувати рекомендації у стандартизованому форматі.

Traceroute є мережею утилітою для діагностики, яка використовується для визначення маршруту та вимірювання часу передачі пакетів між комп'ютером та цільовим сервером. Traceroute використовується для аналізу маршрутів передачі даних та виявлення проблем на мережевому шляху. Це дозволяє користувачам отримувати детальну інформацію про маршрути, що допомагає визначити можливі проблеми з мережею та підвищити якість телекомунікаційних послуг.

Загалом, використання цих технологій дозволило створити ефективний та надійний мобільний застосунок для моніторингу якості телекомунікаційних послуг, який забезпечує зручність використання, високу продуктивність та масштабованість.

3.2 Структура мобільного застосунку для моніторингу якості телекомунікаційних послуг

Структура мобільного застосунку визначає організацію файлів, каталогів та конфігурацій, які використовуються для розробки, тестування, компіляції та розгортання додатку. Вона впливає на зручність розробки, масштабованість, підтримку та подальший розвиток застосунку. Основними складовими структури мобільного застосунку є вихідний код, ресурси, конфігураційні файли, залежності та тестові файли.

Структура мобільного застосунку призначена для моніторингу якості телекомунікаційних послуг. Вона містить всі необхідні компоненти для розробки, налаштування, тестування та розгортання додатку на платформах Android та iOS.

Опис структури

1. tests: каталог для тестових файлів, що використовуються для перевірки функціональності компонентів додатку.
   - Використання: автоматичне тестування компонентів для забезпечення стабільності та якості коду.
2. .bundle: каталог, який може містити зібрані файли та залежності для збірки проекту.
   - Використання: містить метадані для бандлерів, що використовуються під час компіляції.
3. .vscode: конфігураційний каталог для редактора Visual Studio Code.
   - Використання: налаштування редактора коду, такі як розширення, налаштування форматування коду.
4. android: каталог для Android-платформи.
   - Використання: містить файли та конфігурації для розробки та компіляції додатку під Android.
5. ios: каталог для iOS-платформи.

– Використання: містить файли та конфігурації для розробки та компіляції додатку під iOS.

6. node_modules: каталог з встановленими Node.js модулями.

– Використання: зберігає всі залежності проекту, що встановлені через npm.

7. src: основний каталог з вихідним кодом проекту.

– Використання: містить всі файли з реалізацією основного функціоналу додатку.

8. vendor: каталог для сторонніх бібліотек та залежностей.

– Використання: зберігає сторонні залежності, які не встановлюються через менеджери пакетів.

9. .eslintrc.js: файл конфігурації ESLint.

– Використання: налаштування для автоматичного аналізу коду на наявність проблем.

10. .gitignore: файл для налаштування ігнорування файлів та каталогів у системі контролю версій Git.

– Використання: вказує, які файли не потрібно додавати до репозиторію.

11. .prettierrc.js: файл конфігурації Prettier.

– Використання: налаштування для автоматичного форматування коду.

12. .watchmanconfig: файл конфігурації Watchman.

– Використання: моніторинг змін у файлах для виконання відповідних дій.

13. app.json: файл конфігурації проекту.

– Використання: містить метадані та налаштування додатку.

14. App.tsx: основний файл додатку, написаний з використанням TypeScript та JSX.

– Використання: реалізація головного компонента користувацького інтерфейсу.

15. babel.config.js: файл конфігурації Babel.

– Використання: налаштування для транспіляції сучасного JavaScript коду у зворотньо сумісний формат.

16. Gemfile та Gemfile.lock: файли конфігурації для керування залежностями Ruby через Bundler.

– Використання: вказують залежності та їх версії для Ruby компонентів проекту.

17. index.js: початковий файл додатку.

– Використання: містить основну логіку запуску додатку.

18. jest.config.js: файл конфігурації Jest.

– Використання: налаштування для фреймворку тестування JavaScript коду.

19. metro.config.js: файл конфігурації Metro.

– Використання: налаштування для бандлера Metro, який використовується у React Native додатках.

20. package.json: файл конфігурації Node.js проекту.

– Використання: містить метадані проекту, список залежностей та скрипти для виконання задач.

21. README.md: файл документації проекту.

– Використання: містить інформацію про проект, інструкції для встановлення та використання.

22. tsconfig.json: файл конфігурації TypeScript.

– Використання: містить налаштування для компілятора TypeScript.

23. yarn.lock: файл блокування залежностей Yarn.

– Використання: гарантує встановлення всіх залежностей у тій самій версії, що й у автора проекту.

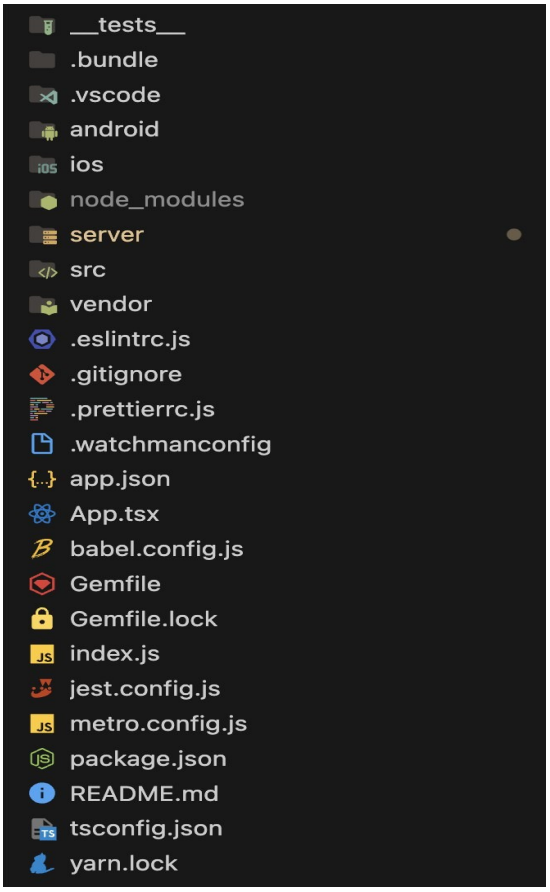


Рисунок 3.1 – Загальна структура проекту.

Структура мобільного застосунку для моніторингу якості телекомунікаційних послуг забезпечує ефективну організацію розробки, тестування та підтримки проекту. Кожен компонент структури має своє призначення, що сприяє досягненню високої якості коду та стабільної роботи додатку на різних платформах.

Загальна структура React Native проекту включає в себе організацію файлів та каталогів, що дозволяє ефективно розробляти, тестувати та підтримувати мобільний додаток. React Native використовує JavaScript та React для створення крос-платформенних мобільних додатків, що можуть працювати на iOS та Android.

Структура проекту визначає, як різні частини коду та ресурси організовані у проекті, щоб забезпечити зручну навігацію, повторне використання коду та легкість у підтримці.

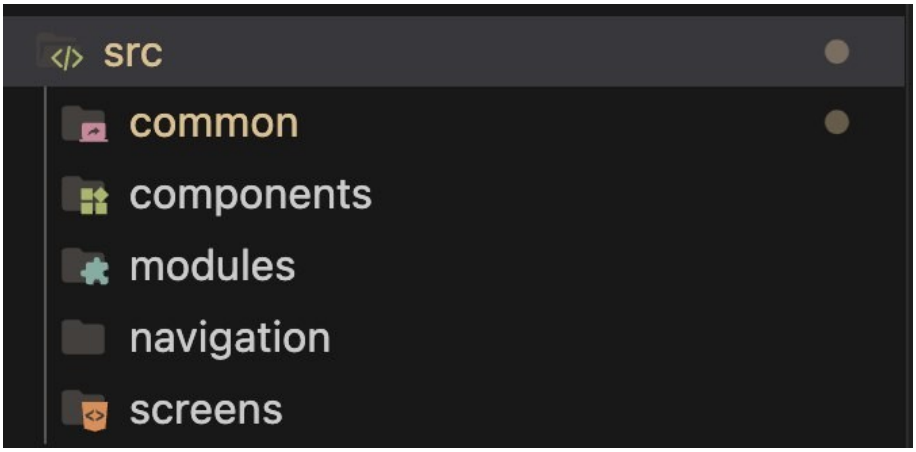


Рисунок 3.2 – Загальна структура React native

На рисунку 3.2 представлена структура каталогу src, що містить основний вихідний код React Native проекту. Каталог src містить наступні підкаталоги:

1. common: каталог, що містить загальні утиліти, стилі та інші спільні ресурси, які використовуються у різних частинах проекту.
   - Утиліти (utilities): функції та методи, які можуть бути повторно використані в різних модулях додатку.
   - Стилі (styles): загальні стилі, що використовуються у компонентах, для підтримки єдиного стилю додатку.
   - Константи (constants): глобальні змінні та налаштування, які використовуються у різних частинах додатку.
2. components: каталог для компонентів, які є будівельними блоками інтерфейсу користувача.
   - Презентаційні компоненти (presentational components): компоненти, які відповідають лише за відображення інтерфейсу та не містять логіки.

– Контейнерні компоненти (container components): компоненти, які містять логіку та взаємодіють зі станом додатку, передаючи дані до презентаційних компонентів.

3. modules: каталог, що містить функціональні модулі додатку.

– Логіка бізнесу (business logic): модулі, які реалізують конкретну функціональність додатку, наприклад, автентифікацію, роботу з API або управління користувачами.
– Дії (actions): операції, які ініціюються користувачем або системою і впливають на стан додатку.
– Редюсери (reducers): функції, які змінюють стан додатку на основі виконаних дій.

4. navigation: каталог для налаштувань навігації додатку.

– Маршрути (routes): визначення шляхів переходу між різними екранами додатку.
– Навігаційні компоненти (navigation components): компоненти, які реалізують логіку переходу між екранами, такі як стеки навігації або вкладки.

5. screens: каталог для екранних компонентів, які відповідають за відображення основних розділів додатку.

Використання:

– Екрани (screens): компоненти, що представляють повноцінні сторінки додатку, такі як головна сторінка, профіль користувача, налаштування тощо.
– Сторінки (pages): можуть бути більш детальними частинами екранних компонентів, поділяючи великі екрани на підкомпоненти.

Використання структурних елементів у проекті

– common використовується для організації спільних ресурсів, що сприяє зменшенню дублювання коду та полегшує підтримку.

- components містить повторно використовувані елементи інтерфейсу, що дозволяє легко складати нові сторінки з існуючих блоків.
- modules забезпечує розподіл логіки додатку на окремі функціональні частини, що покращує зрозумілість та модульність коду.
- navigation визначає структуру навігації між різними частинами додатку, забезпечуючи зручний та інтуїтивний користувацький досвід.
- screens організує основні розділи додатку, кожен з яких відповідає за окремий функціональний аспект, що полегшує навігацію та розвиток додатку.

Ця структура сприяє організованій, модульній та масштабованій розробці мобільного застосунку, що є критично важливим для підтримки та подальшого розширення функціональності проекту.

Запуск програми React Native включає декілька кроків, які забезпечують налаштування середовища розробки, встановлення необхідних залежностей та запуск програми на різних платформах. Цей процес включає використання різних команд для управління пакетами та налаштування платформ.

Встановлення загальних модулів для React Native проекту

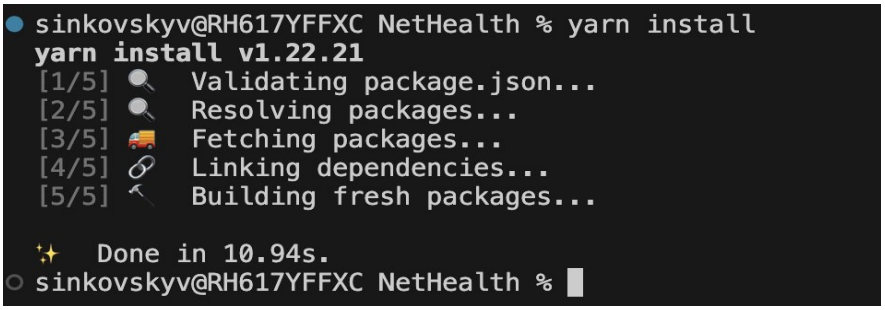


Рисунок 3.3 – Yarn install

На рисунку 3.3 зображено виконання команди yarn install у терміналі:

Команда yarn install: ця команда використовується для встановлення всіх залежностей, зазначених у файлі package.json.

Кроки:

- Validating package.json: перевірка файлу package.json на наявність та коректність.
- Resolving packages: визначення залежностей та їх версій.
- Fetching packages: завантаження необхідних пакетів з реєстру npm.
- Linking dependencies: зв’язування встановлених пакетів з проектом.
- Building fresh packages: збирання нових пакетів.
- Встановлення Pods для iOS

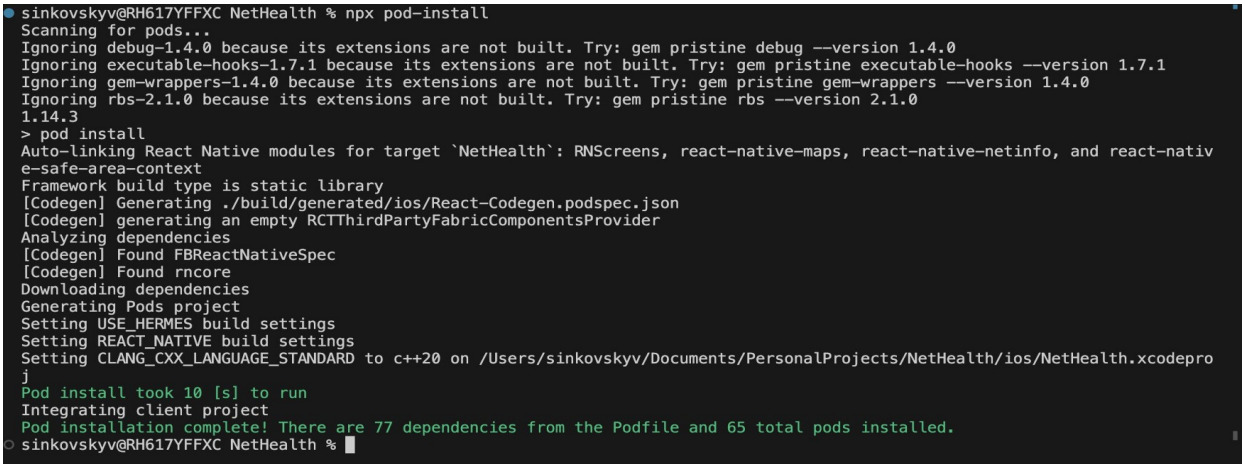


Рисунок 3.4 – npx pod-install

Для iOS платформи додатково необхідно встановити CocoaPods, що забезпечують управління залежностями iOS проекту:

Команда npx pod-install: використовується для встановлення CocoaPods залежностей, які визначені у файлі Podfile.

Кроки:

- Перехід у каталог iOS проекту: підготовка до встановлення Pods.
- Виконання команди pod install: завантаження та встановлення залежностей для iOS.
- Запуск програми на iOS та Android

Після встановлення всіх залежностей можна запускати програму на різних платформах.

– Запуск на iOS
– Команда: yarn ios
– Опис: запускає додаток на iOS симуляторі.
– Кроки:
– Підготовка середовища: Перевірка наявності всіх необхідних залежностей та налаштувань.
– Запуск Metro Bundler: інструмент для бандлінгу JavaScript коду.
– Запуск Xcode: виконується компіляція та запуск додатку у симуляторі iOS.
– Запуск на Android
– Команда: yarn android: запускає додаток на Android емуляторі або підключеному пристрої.
– Кроки:
– Підготовка середовища: перевірка наявності всіх необхідних залежностей та налаштувань.
– Запуск Metro Bundler: інструмент для бандлінгу JavaScript коду.
– Запуск Android Studio: виконується компіляція та запуск додатку у емуляторі Android або на фізичному пристрої.

Запуск програми React Native включає кілька важливих кроків, які починаються з встановлення залежностей за допомогою yarn install і npx pod-install, після чого програму можна запускати на iOS та Android платформах за допомогою відповідних команд. Цей процес забезпечує належну підготовку середовища розробки та гарантує коректне функціонування додатку на різних пристроях.

На рисунку 3.4 зображено виконання команди npx pod-install у терміналі, що використовується для встановлення залежностей iOS проекту. Команда ви-

конує кілька важливих кроків, необхідних для налаштування середовища розробки.

Команда: npx pod-install: виконує встановлення залежностей CocoaPods, які визначені у файлі Podfile в iOS проекті.

- Scanning for pods: сканування проекту для визначення необхідних Pods.
- Ignoring debug-1.4.0...: повідомлення про ігнорування певних залежностей, які не є критичними для проекту.

pod install: власне команда встановлення Pods, яка виконує наступні кроки:

- Auto-linking React Native modules for target NetHealth: автоматичне зв’язування модулів React Native для цілі NetHealth.
- Framework build type is static library: визначення типу збірки фреймворку як статичної бібліотеки.
- [Codegen] Generating...: генерація файлів конфігурації та специфікацій для модулів.
- Setting USE_HERMES build settings: налаштування для використання Hermes, JavaScript-двигуна для React Native.
- Setting REACT_NATIVE build settings: налаштування збірки для React Native.
- Pod install took 10 [s] to run: час, витрачений на виконання команди pod install.
- Pod installation complete!: повідомлення про успішне завершення встановлення Pods, яке включає:
- 77 dependencies from the Podfile and 65 total pods installed: встановлення 77 залежностей з Podfile та загалом 65 Pods.

Виконання команди npx pod-install є важливим кроком у підготовці iOS частини React Native проекту до запуску. Ця команда автоматизує процес

встановлення та зв’язування необхідних залежностей, забезпечуючи правильне налаштування середовища розробки.

3.3 Сервіси мобільного застосунку для якості телекомунікаційних послуг

Даний мобільний застосунок, під назвою "NetHealth", призначений для моніторингу та аналізу якості телекомунікаційних послуг. Головний екран додатку надає користувачу доступ до трьох основних сервісів: "Моя IP адреса", "Пошук по IP", та "Traceroute". Кожен з цих сервісів виконує певні функції та надає користувачу детальну інформацію про стан мережі.

На головному екрані розміщено три основні кнопки для переходу до відповідних сервісів. Користувач може обрати необхідний сервіс, натиснувши на відповідну кнопку: Моя IP адреса; Пошук по IP; Traceroute.

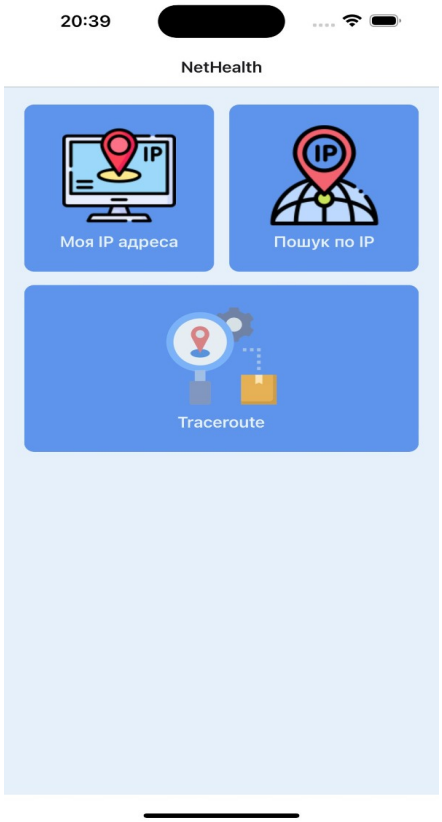


Рисунок 3.5 – Головний екран застосунку

Сервіс "Моя IP адреса" надає користувачу інформацію про його поточну IP адресу та деталі мережі. При використанні цього сервісу користувач отримує такі дані:

– IP адреса: поточна IP адреса користувача (наприклад, 46.173.97.169).
– Тип мережі: тип підключення до мережі (наприклад, WiFi).
– ISP (Internet Service Provider): постачальник інтернет-послуг (наприклад, Kolumbus PE).
– Місто: місто, де знаходиться користувач (наприклад, Chervonohrad).
– Регіон: область або регіон (наприклад, Lvivska oblast).
– Країна: країна (наприклад, Ukraine).

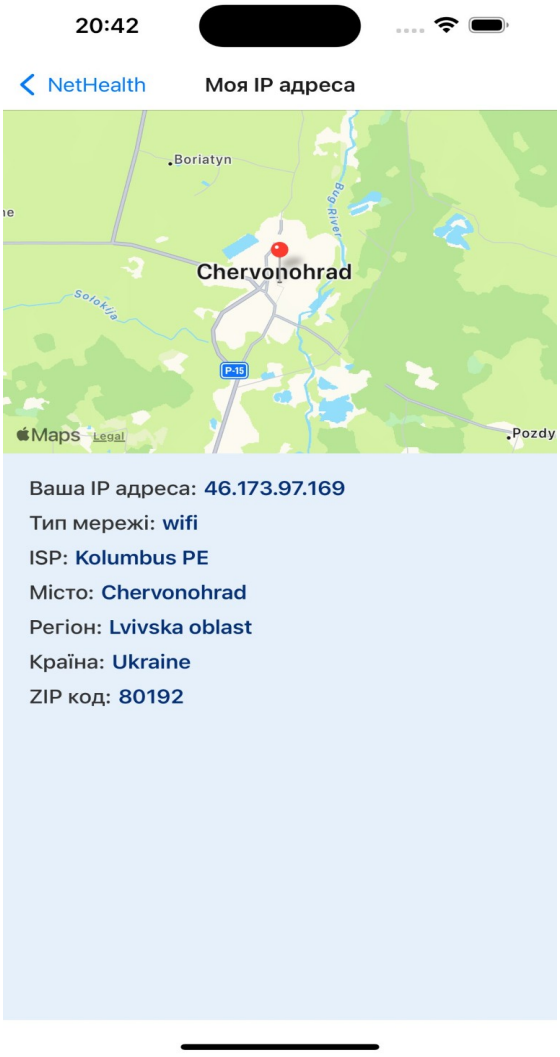


Рисунок 3.6 – Сервіс "Моя IP адреса"

Для реалізації функціональності цього сервісу використовуються декілька важливих технологій та етапів обробки даних:

Використання бібліотек для карт та мережевої інформації

- У мобільному додатку інтегровані бібліотеки для роботи з картами (наприклад, react-native-maps) та для отримання інформації про мережу (наприклад, react-native-netinfo).
- Бібліотека react-native-maps дозволяє відображати карту та геолокаційні дані на ній.
- Бібліотека react-native-netinfo використовується для отримання поточної IP адреси користувача.

Запит на бекенд

- Після отримання поточної IP адреси з допомогою react-native-netinfo, мобільний додаток відправляє запит на бекенд.
- Запит містить інформацію про поточну IP адресу користувача.
- Обробка на бекенді
- Бекенд, отримавши запит від мобільного додатку, відправляє запит до стороннього API сервісу для отримання детальної інформації про IP адресу.
- Використовується API сервісу https://api.ip2location.io, який надає безкоштовний доступ до великої бази даних IP адрес.
- Бекенд відправляє запит до https://api.ip2location.io, включаючи поточну IP адресу користувача.

Отримання та обробка відповіді

- API сервіс ip2location повертає детальну інформацію про IP адресу, включаючи:
- Географічні дані (місто, регіон, країна, поштовий індекс).
- Інформацію про інтернет-провайдера (ISP).

– Тип мережі (WiFi або мобільний інтернет).
– Бекенд обробляє отримані дані та формує відповідь для мобільного додатку.

Відображення інформації в додатку

– Мобільний додаток отримує відповідь від бекенду, яка містить всю необхідну інформацію про IP адресу.
– На екрані сервісу "Моя IP адреса" відображається:
– Поточна IP адреса користувача.
– Тип мережі.
– Інформація про інтернет-провайдера.
– Місто, регіон, країна, поштовий індекс.
– Геолокація IP адреси відображається на карті за допомогою бібліотеки react-native-maps.

Сервіс "Пошук по IP" дозволяє користувачу ввести конкретну IP адресу та отримати детальну інформацію про неї. Інформація включає:

– IP адреса: введена користувачем IP адреса (наприклад, 1.1.1.1).
– ISP: постачальник інтернет-послуг для вказаної IP адреси (наприклад, CloudFlare Inc.).
– Місто: місто, де знаходиться IP адреса (наприклад, Denver).
– Регіон: область або регіон (наприклад, Colorado).
– Країна: країна (наприклад, United States of America).
– ZIP код: поштовий індекс (наприклад, 80002).

Сервіс "Пошук по IP" в мобільному додатку "NetHealth" дозволяє користувачам отримати детальну інформацію про будь-яку IP адресу, введену вручну. Цей сервіс також відображає геолокацію введеної IP адреси на карті.

Як працює сервіс "Пошук по IP"?

Реалізація функціональності сервісу "Пошук по IP" аналогічна до сервісу "Моя IP адреса", але з можливістю введення довільної IP адреси.

Сервіс "Пошук по IP" в мобільному додатку "NetHealth" дозволяє користувачам отримати детальну інформацію про будь-яку IP адресу, введену вручну. Цей сервіс також відображає геолокацію введеної IP адреси на карті.

Користувач також може переглянути інформацію на карті, натиснувши на маркер, що відображає розташування IP адреси.

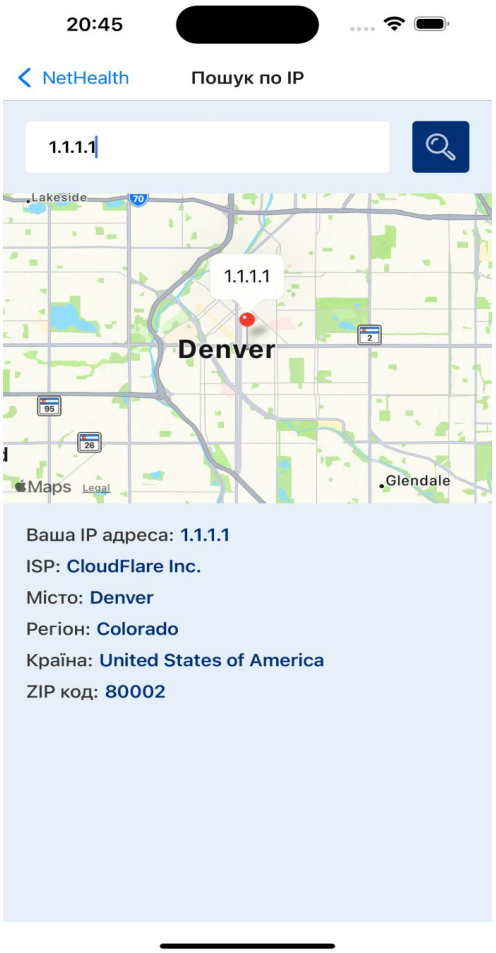


Рисунок 3.6 – Сервіс "Пошук по IP"

Сервіс "Пошук по IP" в мобільному додатку "NetHealth" дозволяє користувачам отримати детальну інформацію про будь-яку IP адресу, введену вручну. Цей сервіс також відображає геолокацію введеної IP адреси на карті.

Як працює сервіс "Пошук по IP"? Реалізація функціональності сервісу "Пошук по IP" аналогічна до сервісу "Моя IP адреса", але з можливістю введен-

ня довільної IP адреси. Для цього використовуються наступні етапи та технології:

Використання бібліотек для карт та мережевої інформації

– У мобільному додатку інтегровані бібліотеки для роботи з картами (наприклад, react-native-maps) та для отримання інформації про мережу (наприклад, react-native-netinfo).
– Бібліотека react-native-maps дозволяє відображати карту та геолокаційні дані на ній.

Введення IP адреси

– Користувач вводить бажану IP адресу у спеціальне поле в інтерфейсі мобільного додатку.

Запит на бекенд

– Після введення IP адреси, мобільний додаток відправляє запит на бекенд з введеною IP адресою.

Обробка на бекенді

– Бекенд, отримавши запит від мобільного додатку, відправляє запит до стороннього API сервісу для отримання детальної інформації про введену IP адресу.
– Використовується API сервісу https://api.ip2location.io, який надає безкоштовний доступ до великої бази даних IP адрес.
– Бекенд відправляє запит до https://api.ip2location.io, включаючи введену IP адресу.

Отримання та обробка відповіді

API сервіс ip2location повертає детальну інформацію про введену IP адресу, включаючи:

– Географічні дані (місто, регіон, країна, поштовий індекс).
– Інформацію про інтернет-провайдера (ISP).

– Бекенд обробляє отримані дані та формує відповідь для мобільного додатку.

Відображення інформації в додатку

– мобільний додаток отримує відповідь від бекенду, яка містить всю необхідну інформацію про введену IP адресу.
– на екрані сервісу "Пошук по IP" відображається:
– введена IP адреса користувача.
– інформація про інтернет-провайдера.
– місто, регіон, країна, поштовий індекс.
– геолокація введеної IP адреси відображається на карті за допомогою бібліотеки react-native-maps.

Сервіс "Traceroute" надає користувачу можливість виконати трасування маршруту мережі до вказаної IP адреси. Цей сервіс показує послідовність проходження пакетів через різні вузли мережі та час затримки на кожному з них.

Інформація, що відображається, включає:

– IP адреси та вузли: IP адреси вузлів, через які проходить трафік.
– час затримки: Час затримки (в мілісекундах) для кожного вузла.

Сервіс "Traceroute" в мобільному додатку "NetHealth" дозволяє користувачам отримати трасування маршруту для будь-якої IP адреси. Цей сервіс відображає послідовність проходження пакетів через різні вузли мережі та час затримки на кожному з них, а також геолокацію кожного вузла на карті.

Для реалізації функціональності сервісу "Traceroute" використовуються декілька важливих етапів та технологій:

Введення IP адреси

Користувач вводить бажану IP адресу у спеціальне поле в інтерфейсі мобільного додатку.

– Запит на бекенд
– Після введення IP адреси, мобільний додаток відправляє запит на бекенд з введеною IP адресою.
– Запуск утиліти traceroute на бекенді
– Бекенд, отримавши запит від мобільного додатку, запускає утиліту traceroute для зазначеної IP адреси.
– Утиліта traceroute виконує трасування маршруту до зазначеної IP адреси, визначаючи всі проміжні вузли та час затримки до кожного з них.

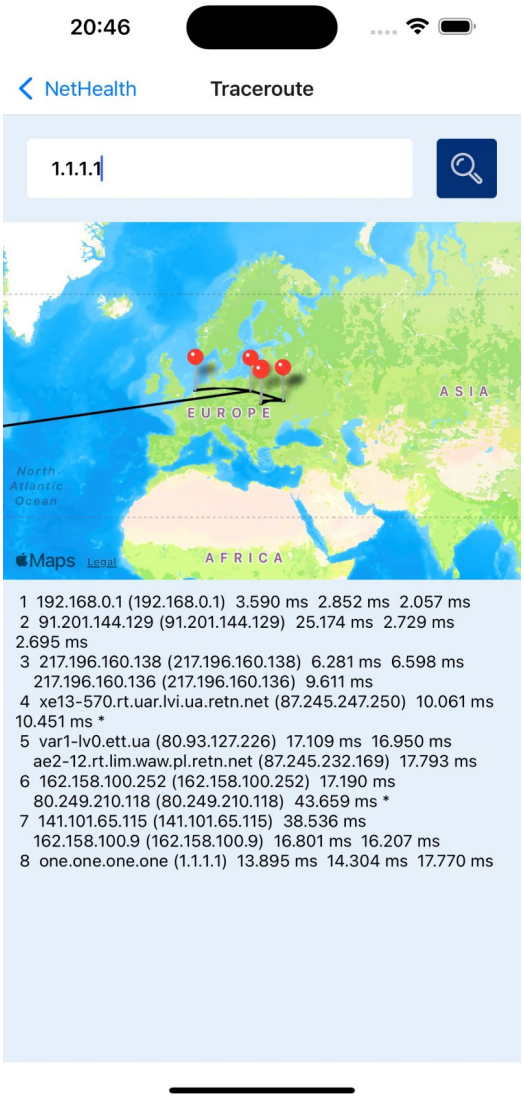


Рисунок 3.7 – Сервіс "Traceroute"

Отримання геолокації кожного вузла

- Для кожного проміжного вузла (роута), отриманого від утиліти traceroute, бекенд відправляє запит до API сервісу https://api.ip2location.io для отримання геолокаційних даних.
- API сервіс ip2location повертає детальну інформацію про кожен вузол, включаючи географічні дані (місто, регіон, країна) та інформацію про інтернет-провайдера (ISP).
- Формування відповіді та відправка даних до мобільного додатку
- Бекенд формує відповідь, що містить дані про трасування маршруту, включаючи геолокацію кожного вузла та час затримки.
- Мобільний додаток отримує дані з серверу.
- Відображення інформації в додатку
- На екрані сервісу "Traceroute" відображаються:
- Список всіх вузлів маршруту з їх IP адресами та часом затримки до кожного вузла.
- Геолокація кожного вузла відображається на карті за допомогою бібліотеки react-native-maps.
- Користувач може бачити візуальний маршрут на карті, що показує шлях проходження пакетів через різні вузли.

На карті відображається маршрут проходження пакетів, що дозволяє візуально оцінити шлях даних від користувача до кінцевої точки.

Даний мобільний застосунок "NetHealth" надає користувачам зручний інтерфейс для отримання детальної інформації про стан їх мережі та якість телекомунікаційних послуг. Використовуючи сервіси "Моя IP адреса", "Пошук по IP", та "Traceroute", користувачі можуть легко моніторити свої мережеві підключення, отримувати важливі дані про свою IP адресу та аналізувати маршрути трафіку в реальному часі. Ці інструменти є корисними як для звичайних користувачів, так і для фахівців з мережевих технологій.

3.4 Веб API для мобільного застосунку

Для забезпечення функціональності мобільного застосунку "NetHealth", було створено веб API на основі Node.js та Express.js. Це необхідно через обмеження мобільних платформ щодо підтримки команд traceroute та безпечного зберігання токенів для API. Використання веб API дозволяє централізовано обробляти запити та забезпечувати безпечний доступ до необхідних даних та сервісів.

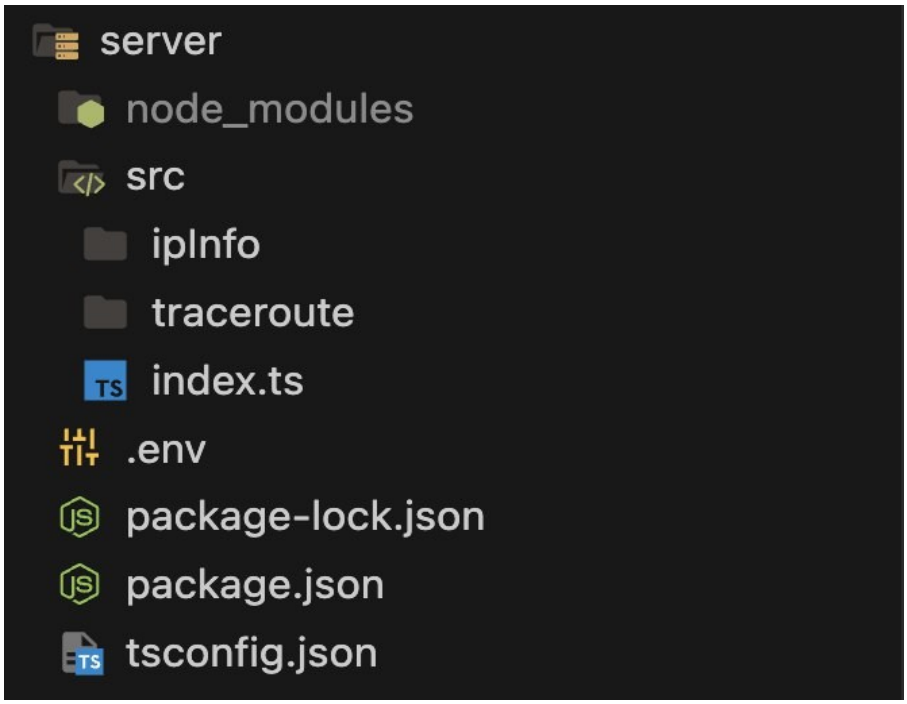


Рисунок 3.8 – Структура проекту Веб API

Структура проекту:

1. server: головний каталог проекту, який містить всі необхідні файли та каталоги для роботи бекенду.

2. node_modules: каталог, який містить всі встановлені пакети та залежності Node.js, необхідні для роботи проекту. Ці залежності вказані у файлі package.json.

3. src: основний каталог з вихідним кодом проекту.

– Містить підкаталоги та файли, що реалізують функціональність веб API.

4. ipInfo: каталог, що містить код для обробки запитів, пов'язаних з отриманням інформації про IP адреси.

– Використання: тут знаходяться функції та модулі, які обробляють запити користувачів для отримання інформації про IP адресу через сторонні сервіси, такі як ip2location.

5. traceroute: каталог, що містить код для виконання команди traceroute.

– Використання: тут реалізовані функції для запуску утиліти traceroute на сервері, обробки результатів та надсилання запитів до сторонніх API для отримання геолокаційних даних про вузли маршруту.

6. index.ts: головний файл входу для веб API.

– Використання: файл містить основну конфігурацію та налаштування сервера Express.js. Він відповідає за ініціалізацію сервера, підключення до необхідних маршрутів та запуск веб сервісу.

7. .env: файл конфігурації середовища.

– Використання: містить змінні середовища, такі як API ключі та інші конфіденційні дані, що використовуються у проекті. Використовується для безпечного зберігання та доступу до налаштувань середовища.

8. package-lock.json: файл блокування версій залежностей.

– Використання: гарантує, що всі залежності встановлюються у тих самих версіях, що були використані під час розробки. Це допомагає уникнути проблем сумісності.

9. package.json: файл конфігурації проекту.

– Використання: містить метадані проекту, включаючи список залежностей, скрипти для запуску та інші налаштування проекту.

10. tsconfig.json: файл конфігурації TypeScript.

– Використання: містить налаштування для компілятора TypeScript, визначає правила та параметри трансляції TypeScript коду у JavaScript.

Структура проекту веб API для мобільного застосунку "NetHealth" забезпечує організовану та ефективну розробку бекенд частини, яка підтримує основні функції додатку. Використання Node.js та Express.js дозволяє централізовано обробляти запити, зберігати конфіденційні дані безпечно та забезпечувати стабільну роботу додатку на різних платформах.

Для запуску веб API, що підтримує функціональність мобільного застосунку "NetHealth", необхідно встановити всі необхідні залежності та запустити сервер. Веб API було створено з використанням Node.js та Express.js. Далі наводиться покрокова інструкція для запуску проекту.

Встановлення залежностей

Для запуску веб API необхідно спочатку встановити всі залежності, визначені у файлі package.json. Це виконується за допомогою команди yarn install.

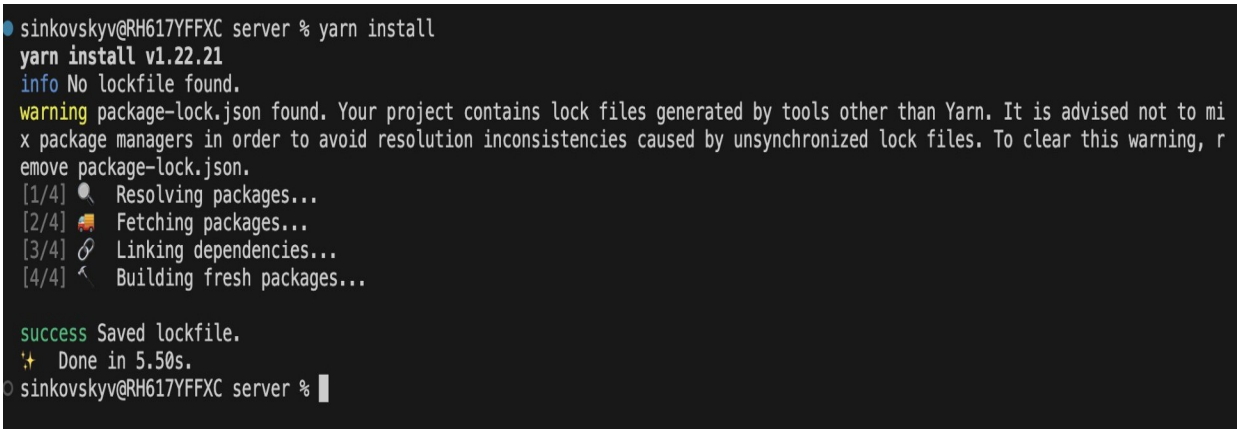


Рисунок 3.9 – Yarn install

Команда yarn install: команда встановлює всі залежності, зазначені у файлі package.json.

Кроки виконання:

– Resolving packages: визначення версій залежностей.

– Fetching packages: завантаження пакетів з реєстру npm.

– Linking dependencies: зв'язування встановлених пакетів з проектом.
– Building fresh packages: збирання нових пакетів.

Результат:

– Після успішного виконання команди, всі необхідні залежності будуть встановлені в каталог node_modules.

Запуск сервера

Після встановлення залежностей можна запустити сервер за допомогою команди yarn start.

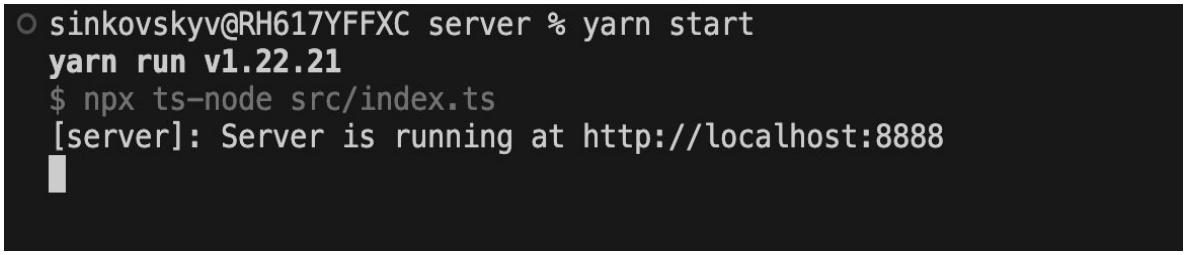


Рисунок 3.10 – Yarn start

Команда yarn start: команда запускає сервер, використовуючи налаштування, визначені в проекті.

Кроки виконання:

– Ініціалізація сервера Express.js.
– Підключення до маршрутів та середовища виконання.
– Запуск сервера та прослуховування вхідних запитів на визначеному порту.

Результат:

– Після успішного виконання команди сервер буде запущено, і веб API буде готове обробляти запити від мобільного застосунку.

Процес запуску веб API для мобільного застосунку "NetHealth" включає два основні кроки: встановлення необхідних залежностей за допомогою команди yarn install та запуск сервера за допомогою команди yarn start. Це забезпечує готовність бекенду обробляти запити від мобільного застосунку та надавати необхідні дані та сервіси.

ВИСНОВКИ

В ході виконання кваліфікаційної роботи бакалавра було проведено аналіз сучасних підходів до розробки мобільних застосунків для моніторингу якості телекомунікаційних послуг. Визначено важливість мобільних технологій у сучасному інформаційному середовищі та розглянуто методи розробки, тестування та впровадження таких застосунків.

У ході виконання кваліфікаційної роботи бакалавра було виконано наступні завдання:

1. Дослідження сучасних підходів та інструментів розробки мобільних застосунків, включаючи мови програмування, фреймворки та платформи.
2. Розроблено та налаштовано прототип мобільного застосунку для моніторингу якості телекомунікаційних послуг, що дозволяє користувачам оцінювати різні параметри, такі як швидкість інтернет-з'єднання, стабільність сигналу та інші показники.
3. Виконано інтеграцію з API телекомунікаційних провайдерів для отримання актуальних даних про якість послуг.
4. Налаштовано систему збору даних та їх обробки для надання користувачам детальної інформації про якість телекомунікаційних послуг у реальному часі.
5. Проведено тестування мобільного застосунку на різних пристроях та в різних умовах, щоб забезпечити його надійність та ефективність.

Отже, можна зробити висновок, що розроблений мобільний застосунок для моніторингу якості телекомунікаційних послуг є ефективним та перспективним рішенням для сучасних інформаційних середовищ.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. "Telecommunications Engineering: Principles and Practice" - [Електронний ресурс] - URL: https://www.worldscientific.com/worldscibooks/10.1142/11053 (дата звернення: 15.05.2024)
2. Telecommunications Engineering: Principles and Practice - [Електронний ресурс] - URL: https://www.worldscientific.com/worldscibooks/10.1142/11053 (дата звернення: 15.05.2024)
3. A Comprehensive Guide to Quality of Service: Demystifying QoS - [Електронний ресурс] - URL: https://lab.wallarm.com/a-comprehensive-guide-to-quality-of-service-demystifying-qos/ (дата звернення: 28.05.2024).
4. ETSI - Telecommunications Standards Updates - [Електронний ресурс] - URL: https://www.etsi.org/technologies (дата звернення: 28.05.2024).
5. ITU Committed to Connecting the World - [Електронний ресурс] - URL: http://epublications.itu.int (дата звернення: 28.05.2024).
6. ISO - ISO Update - [Електронний ресурс] - URL: https://www.iso.org/news/ref2905.html (дата звернення: 28.05.2024).
7. Measuring Latency and Jitter in the Internet with Arduino and Raspberry Pi - [Електронний ресурс] - URL: https://ieeexplore.ieee.org/document/9165121 (дата звернення: 28.05.2024).
8. Bandwidth in Telecom: The Complete Guide - Techopedia - [Електронний ресурс] - URL: https://www.techopedia.com/definition/24689/bandwidth (дата звернення: 28.05.2024).
9. What is Packet Loss? | Twilio - [Електронний ресурс] - URL: https://www.twilio.com/docs/voice/ip-communications/what-is-packet-loss (дата звернення: 28.05.2024).

10. Computer Tech Reviews - What is Jitter? - [Електронний ресурс] - URL: https://www.computertechreviews.com/what-is-jitter-definition-functions-and-features/ (дата звернення: 28.05.2024).
11. Building Resilience in Telecommunications – In Canada and Beyond - [Електронний ресурс] - URL: https://www.ivey.uwo.ca/news/events/2024/05/building-resilience-in-telecommunications-in-canada-and-beyond/ (дата звернення: 28.05.2024).
12. What Standards Apply to the Electronics & Telecom Industry? - [Електронний ресурс] - URL: https://www.nqa.com/en-gb/certification/standards/iso-9001 (дата звернення: 28.05.2024).
13. Splunk. Data Monitoring: Benefits, Best Practices, and Automation Opportunities - [Електронний ресурс] - URL: https://www.splunk.com/en_us/blog/data-monitoring-benefits-best-practices-and-automation-opportunities.html (дата звернення: 29.05.2024).
14. Techwatch.de - "What is NMS (Network Management System) and why should you use it?" - [Електронний ресурс] - URL: https://techwatch.de/what-is-nms/ (дата звернення: 28.05.2024).
15. Best network monitoring tool of 2024 | TechRadar - [Електронний ресурс] - URL: https://www.techradar.com/best/best-network-monitoring-tools (дата звернення: 28.05.2024).
16. AWS Monitoring Best Practices - [Електронний ресурс] - URL: https://aws.amazon.com/cloudwatch/monitoring/ (дата звернення: 28.05.2024).
17. "Azure Network Watcher Overview" - [Електронний ресурс] - URL: https://learn.microsoft.com/en-us/azure/network-watcher/network-watcher-monitoring-overview (дата звернення: 28.05.2024).

18. Google Cloud Platform Monitoring Solutions | Datadog - [Електронний ресурс] - URL: https://www.datadoghq.com/solutions/google-cloud-platform-monitoring/ (дата звернення: 01.06.2024).
19. IBM Cloud Docs: Monitoring Service - [Електронний ресурс] - URL: https://cloud.ibm.com/docs/monitoring (дата звернення: 28.05.2024).
20. DigitalOcean Documentation: Monitoring - [Електронний ресурс] - URL: docs.digitalocean.com (дата звернення: 28.05.2024).
21. "Salesforce Shield" - [Електронний ресурс] - URL: https://trailhead.salesforce.com/en/content/learn/modules/enterprise-security/event-monitoring (дата звернення: 15.05.2024).
22. SAP Blog - Using SAP Cloud ALM for operational monitoring - [Електронний ресурс] - URL: https://blogs.sap.com/using-sap-cloud-alm-for-operational-monitoring (дата звернення: 30.05.2024).
23. Amazon Web Services - "Collecting and Analyzing Mobile App Telemetry with AWS" - [Електронний ресурс] - URL: https://aws.amazon.com/blogs/mobile/collecting-and-analyzing-mobile-app-telemetry-with-aws/ (дата звернення: 28.05.2024).
24. Postman API Documentation Tool - [Електронний ресурс] - URL: https://www.postman.com/api-documentation-tool/ (дата звернення: 28.05.2024).
25. Google Analytics 4 Event Tracking Checklist (2024) - [Електронний ресурс] - URL: https://measureschool.com/google-analytics-4-event-tracking-checklist/ (дата звернення: 27.05.2024).
26. Firebase Documentation - [Електронний ресурс] - URL: Firebase Documentation (дата звернення: 30.05.2024).
27. New Relic Documentation - [Електронний ресурс] - URL: New Relic Documentation (дата звернення: 30.05.2024).
28. Cisco AppDynamics Documentation - [Електронний ресурс] - URL: Cisco AppDynamics Documentation (дата звернення: 30.05.2024).

29. Amazon S3 Documentation - [Електронний ресурс] - URL: Amazon S3 Documentation (дата звернення: 30.05.2024).
30. Google Cloud Storage Documentation - [Електронний ресурс] - URL: Google Cloud Storage Documentation (дата звернення: 30.05.2024).
31. Nettest Package - golang.org - [Електронний ресурс] - URL: nettest package (дата звернення: 30.05.2024).
32. Opensignal Documentation - [Електронний ресурс] - URL: Opensignal Documentation (дата звернення: 30.05.2024).
33. NDT (Network Diagnostic Tool) - M-Lab - [Електронний ресурс] - URL: M-Lab NDT (дата звернення: 30.05.2024).
34. Speedtest by Ookla - The Global Broadband Speed Test - [Електронний ресурс] - URL: Speedtest.net (дата звернення: 30.05.2024).
35. TelephonyManager - Android SDK | Android Developers - [Електронний ресурс] - URL: TelephonyManager - Android Developers (дата звернення: 30.05.2024).
36. CTTelephonyNetworkInfo | Apple Developer Documentation - [Електронний ресурс] - URL: CTTelephonyNetworkInfo (дата звернення: 30.05.2024).
37. "Get started with Grafana and Prometheus" - [Електронний ресурс] - URL: Grafana documentation (дата звернення: 28.05.2024).
38. "The Complete Guide to the ELK Stack" - [Електронний ресурс] - URL: https://logz.io/learn/complete-guide-elk-stack/ (дата звернення: 28.05.2024).
39. "Apache Kafka" - [Електронний ресурс] - URL: https://kafka.apache.org/ (дата звернення: 28.05.2024).
40. "Apache Kafka Architecture: Getting Started with Apache Kafka" - [Електронний ресурс] - URL: https://medium.com/analytics-vidhya/apache-kafka-architecture-getting-started-with-apache-kafka-771d69ac6cef (дата звернення: 31.05.2024).

41. "Apache Flink Documentation" - [Електронний ресурс] - URL: https://flink.apache.org/documentation/ (дата звернення: 31.05.2024).
42. "Apache Flink Architecture" - [Електронний ресурс] - URL: https://www.cloudduggu.com/flink/architecture/ (дата звернення: 28.05.2024).
43. "Apache Storm" - [Електронний ресурс] - URL: https://storm.apache.org/ (дата звернення: 31.05.2024).
44. "Serverless Data Processing with Dataflow: Foundations" - [Електронний ресурс] - URL: https://www.cloudskillsboost.google/paths/16/course_templates/218 (дата звернення: 31.05.2024).
45. "Understanding the Google Cloud Dataflow Model" - [Електронний ресурс] - URL: https://www.analyticsvidhya.com/blog/2022/10/understanding-the-google-cloud-dataflow-model/ (дата звернення: 31.05.2024).
46. "Data Stream Processing - Amazon Kinesis" - [Електронний ресурс] - URL: https://aws.amazon.com/kinesis/ (дата звернення: 28.05.2024).
47. "Azure Stream Analytics Overview" - [Електронний ресурс] - URL: https://learn.microsoft.com/en-us/azure/stream-analytics/stream-analytics-monitoring (дата звернення: 30.05.2024).
48. "Introduction to Azure Stream Analytics" - [Електронний ресурс] - URL: https://learn.microsoft.com/en-us/azure/stream-analytics/stream-analytics-introduction (дата звернення: 31.05.2024).
49. "Wireshark Tutorial: Identifying Hosts and Users" - [Електронний ресурс] - URL: https://unit42.paloaltonetworks.com/wireshark-tutorial-identifying-hosts-and-users/ (дата звернення: 31.05.2024).
50. "iPerf - Download iPerf3 and original iPerf pre-compiled binaries" - [Електронний ресурс] - URL: https://iperf.fr/iperf-download.php (дата звернення: 31.05.2024).
51. "PingPlotter Cloud Manual" - [Електронний ресурс] - URL: https://pingplotter.cloud/docs/cloud_manual.pdf (дата звернення: 28.05.2024).

52. "NetFlow Traffic Analyzer | Real-Time NetFlow Analysis" - [Електронний ресурс] - URL: https://www.manageengine.com/products/netflow/help/introduction.html (дата звернення: 28.05.2024).
53. Network Performance Monitor Getting Started Guide" - [Електронний ресурс] - URL: https://documentation.solarwinds.com/en/Success_Center/NPM/Content/NPM_Getting_Started_Guide.htm (дата звернення: 28.05.2024).
54. "Nagios Open Source" - [Електронний ресурс] - URL: https://www.nagios.org/projects/nagios-core/ (дата звернення: 31.05.2024).

Додаток А. Код клієнтської частини

src\common\assets\index.ts:

```
export const Assets = {
  myIpAddress: require('./MyIpAddress.png'),
  anotherIpAddress: require('./AnotherIpAddress.png'),
  search: require('./search.png'),
  tracking: require('./tracking.png'),
};
```

src\common\config\api.ts:

```
export const APIs = {
  LOCALHOST: {
    baseUrl: 'http://localhost:8888',
  },
};

export const API = APIs.LOCALHOST;
```

src\common\config\config.ts:

```
export {};
```

src\common\config\index.ts:

```
export * from './config';
export * from './api';
```

src\common\conts\color.ts:

```
export enum Colors {
  mainBlue = '#5e94eb',

  backgroundMain = '#e6f0fa',

  mainText = '#333333',
  blueText = '#043078',

  white = 'white',
```

```
}
```

src\common\conts\index.ts:

```
export * from './color';
```

src\common\hooks\index.ts:

```
export * from './useNetInfo';
```

src\common\hooks\useNetInfo\index.ts:

```
export * from './useNetInfo';
```

src\common\hooks\useNetInfo\ useNetInfo.ts:

```
import {useCallback, useEffect, useMemo, useState} from 'react';
import {GetIpInfoResponse} from '../../services/netInfo/types';
import {useNetInfo as useRNCNetInfo} from '@react-native-community/netinfo';
import {NetInfoService} from '../../services';

export const useNetInfo = () => {
 // Net info from mobile module
 const {type, details} = useRNCNetInfo();

 const [netInfo, setNetInfo] = useState<GetIpInfoResponse | undefined>(
  undefined,
 );

 const [publicIpAdress, setPublicIpAdress] = useState<string>();

 // Fetch details info from mobile module
 const {ipAddress: privateIpAdress, subnet} = useMemo(() => {
  if (details) {
   return details as {ipAddress: string; subnet: string};
  }

  return {ipAddress: undefined, subnet: undefined};
 }, [details]);

 // Fetch location from third-part api
 const fetchLocationData = useCallback(async (_ipAddress: string) => {
  const response = await NetInfoService.getIpInfo(_ipAddress!);
```

```
    setNetInfo(response);
  }, []);

  const fetchMyNetInfo = useCallback(() => {
    if (!publicIpAdress) {
      return;
    }
    fetchLocationData(publicIpAdress);
  }, [fetchLocationData, publicIpAdress]);

  useEffect(() => {
    NetInfoService.getPublicIpAdress().then(setPublicIpAdress);
  }, []);

  return {
    fetchLocationData,
    publicIpAdress,
    fetchMyNetInfo,
    privateIpAdress,
    subnet,
    netInfo,
    netType: type,
  };
};
```

src\common\services\index.ts:

```
export * from './netInfo';
```

src\common\services\netInfo\index.ts:

```
export * from './netInfo';
```

src\common\services\netInfo\netInfo.ts:

```
import {API} from '../../config';
import {GetIpInfoResponse, TracerouteHop} from './types';

export class NetInfoService {
  static async getIpInfo(ipAddress: string): Promise<GetIpInfoResponse> {
    return fetch(`${API.baseUrl}/ipinfo?ip=${ipAddress}`).then(response =>
      response.json(),
```

```
    );
  }

  static async getPublicIpAdress(): Promise<string> {
    // TODO: Replace to custom server endpoint
    return fetch('https://api.ipify.org/?format=json')
      .then(response => response.json())
      .then(response => response.ip);
  }

  static async traceroute(
    ip: string,
  ): Promise<{hops: TracerouteHop[]; full: string}> {
    return fetch(`${API.baseUrl}/traceroute?ip=${ip}`).then(response =>
      response.json(),
    );
  }
}
```

src\common\services\netInfo\types.ts:

```
export type GetIpInfoResponse = {
  as: string;
  asn: string;
  city_name: string;
  country_code: string;
  country_name: string;
  ip: string;
  is_proxy: boolean;
  latitude: number;
  longitude: number;
  region_name: string;
  time_zone: string;
  zip_code: string;
};

export type TracerouteHop = {
  ipAddress?: string;
  full: string;
  latitude: number;
  longitude: number;
};
```

src\components\buttons\index.ts:

```
import {Tile} from './Tile';

export const Buttons = {Tile};
```

src\components\buttons\Tile\index.ts

```
export * from './Tile';
```

src\components\buttons\Tile\styles.ts:

```
import {StyleSheet} from 'react-native';
import {Colors} from '../../../common/conts';

export const styles = StyleSheet.create({
  button: {
    backgroundColor: Colors.mainBlue,
    borderRadius: 10,
    alignItems: 'center',
    justifyContent: 'center',

    gap: 8,
  },

  title: {
    color: Colors.backgroundMain,
    fontSize: 17,
    fontWeight: '600',
  },
});
```

src\components\buttons\Tile\Tile.tsx:

```
import React from 'react';
import {
  Image,
  ImageSourcePropType,
  Text,
  TouchableOpacity,
  TouchableOpacityProps,
} from 'react-native';
import {styles} from './styles';
```

```
type TileProps = {
  asset: ImageSourcePropType;
  width?: number;
  height?: number;
  title?: string;
} & TouchableOpacityProps;

const DEFAULT_SIZE = 100;

const IMAGE_RATION = 0.6;

export const Tile = ({
  width = DEFAULT_SIZE,
  height = DEFAULT_SIZE,
  asset,
  style,
  title,
  ...props
}: TileProps) => {
  return (
    <TouchableOpacity
      style={[styles.button, {width, height}, style]}
      {...props}>
      <Image
        source={asset}
        style={{width: height * IMAGE_RATION, height: height * IMAGE_RATION}}
      />

      {!!title && <Text style={styles.title}>{title}</Text>}
    </TouchableOpacity>
  );
};
```

src\components\sections\IpInfo\index.ts:

```
export * from './IpInfo';
```

src\components\sections\IpInfo\ IpInfo.tsx:

```
import React from 'react';
import {Text, View} from 'react-native';
import {styles} from './styles';
```

```
type IpInfoProps = {
  ipAddress: string;
  isp: string;
  city: string;
  region: string;
  country: string;
  zip: string;
  netType?: string;
};

export const IpInfo = ({
  ipAddress,
  isp,
  city,
  country,
  region,
  zip,
  netType,
}: IpInfoProps) => {
  return (
    <View style={styles.container}>
      <View style={styles.infoContainer}>
        <Text style={styles.infoLabel}>Ваша IP адреса:</Text>
        <Text style={styles.infoValue}>{ipAddress}</Text>
      </View>

      {!!netType && (
        <View style={styles.infoContainer}>
          <Text style={styles.infoLabel}>Тип мережі:</Text>
          <Text style={styles.infoValue}>{netType}</Text>
        </View>
      )}

      <View style={styles.infoContainer}>
        <Text style={styles.infoLabel}>ISP:</Text>
        <Text style={styles.infoValue}>{isp}</Text>
      </View>

      <View style={styles.infoContainer}>
        <Text style={styles.infoLabel}>Місто:</Text>
        <Text style={styles.infoValue}>{city}</Text>
      </View>
```

```
      <View style={styles.infoContainer}>
        <Text style={styles.infoLabel}>Регіон:</Text>
        <Text style={styles.infoValue}>{region}</Text>
      </View>

      <View style={styles.infoContainer}>
        <Text style={styles.infoLabel}>Країна:</Text>
        <Text style={styles.infoValue}>{country}</Text>
      </View>

      <View style={styles.infoContainer}>
        <Text style={styles.infoLabel}>ZIP код:</Text>
        <Text style={styles.infoValue}>{zip}</Text>
      </View>
    </View>
  );
};
```

src\components\sections\IpInfo\styles.ts:

```
import {StyleSheet} from 'react-native';
import {Colors} from '../../../common/conts';

export const styles = StyleSheet.create({
  container: {
    flexDirection: 'column',
    gap: 10,
    margin: 20,
  },

  infoContainer: {
    flexDirection: 'row',
    gap: 5,
    alignItems: 'center',
  },

  infoLabel: {
    fontSize: 17,
    fontWeight: '500',
    color: Colors.mainText,
  },

  infoValue: {
```

```
    fontSize: 17,
    fontWeight: '600',
    color: Colors.blueText,
  },
});
```

src\components\sections\IpSearch\index.ts:

```
export * from './IpSearch';
```

src\components\sections\IpSearch\IpSearch.tsx:

```
import React, {useState} from 'react';
import {Image, TextInput, TouchableOpacity, View} from 'react-native';
import {Assets} from '../../../common/assets';
import {styles} from './styles';

type IpSearchProps = {
  onSearch: (search: string) => void;
};

export const IpSearch = ({onSearch}: IpSearchProps) => {
  const [search, setSearchValue] = useState<string>('');

  return (
    <View style={styles.searchContainer}>
      <TextInput
        style={styles.textInput}
        keyboardType="numeric"
        placeholder="Введіть IP адресу"
        onChangeText={setSearchValue}
      />

      <TouchableOpacity
        style={styles.searchButton}
        onPress={() => onSearch(search)}>
        <Image source={Assets.search} style={styles.searchImage} />
      </TouchableOpacity>
    </View>
  );
};
```

src\components\sections\IpSearch\styles.ts:

```
import {StyleSheet} from 'react-native';
import {Colors} from '../../../common/conts';

const SEARCH_BAR_HEIGHT = 50;
const SEARCH_BAR_BORDER_RAIDUS = 4;

export const styles = StyleSheet.create({
 textInput: {
  backgroundColor: Colors.white,
  fontSize: 16,
  fontWeight: '600',
  height: SEARCH_BAR_HEIGHT,
  flex: 1,
  borderRadius: SEARCH_BAR_BORDER_RAIDUS,
  paddingLeft: 20,
 },
 searchContainer: {
  flexDirection: 'row',
  // flex: 1,
  margin: 20,
  gap: 20,
 },

 searchButton: {
  height: SEARCH_BAR_HEIGHT,
  width: SEARCH_BAR_HEIGHT,
  backgroundColor: Colors.blueText,
  borderRadius: SEARCH_BAR_BORDER_RAIDUS,

  alignItems: 'center',
  justifyContent: 'center',
 },

 searchImage: {width: 30, height: 30},
});
```

src\components\sections\index.ts:

```
import {IpInfo} from './IpInfo';

export const Sections = {IpInfo};
```

src\modules\Map\index.ts:

```
export * from './Map';
```

src\modules\Map\ Map.tsx:

```
import React from 'react';
import {DimensionValue} from 'react-native';
import MapView, {
  LatLng,
  MapViewProps,
  Marker,
  Polyline,
} from 'react-native-maps';

type MapProps = {
  width?: DimensionValue;
  height?: DimensionValue;
  coords?: LatLng;
  polyline?: (LatLng & {title: string})[];
  markerTitle?: string;
} & Pick<MapViewProps, 'style'>;

export const Map = ({
  width = '100%',
  height = 300,
  style,
  coords,
  polyline,
  markerTitle,
}: MapProps) => {
  const latitude = coords?.latitude || polyline?.[0]?.latitude;
  const longitude = coords?.longitude || polyline?.[0]?.longitude;

  if (!latitude || !longitude) {
    return null;
  }

  return (
    <>
      <MapView
        style={[{width, height}, style]}
        initialRegion={{
          latitude,
```

```
        longitude,
        latitudeDelta: polyline ? 100 : 0.0922,
        longitudeDelta: polyline ? 100 : 0.0421,
      }}>
      {!polyline && coords && (
        <Marker
          coordinate={{
            latitude: coords.latitude,
            longitude: coords.longitude,
          }}
          title={markerTitle}
        />
      )}

      {polyline?.map((line, index) => (
        <Marker key={index} coordinate={line} title={line.title} />
      ))}

      {!!polyline && <Polyline coordinates={polyline} strokeWidth={2} />}
    </MapView>
  </>
 );
};
```

src\navigation\index.ts:

```
export * from './Navigator';
```

src\navigation\ Navigator.tsx:

```
import React from 'react';
import {NavigationContainer} from '@react-navigation/native';
import {createNativeStackNavigator} from '@react-navigation/native-stack';
import {Screens, StackNavigatorScreens} from './types';
import {MenuScreen} from '../screens/Menu';
import {MyIpInfoScreen} from '../screens/MyIpInfo';
import {FindByIp} from '../screens/FindByIp';
import {TraceRoute} from '../screens/Traceroute';

const Stack = createNativeStackNavigator<StackNavigatorScreens>();

export const Navigator = () => {
```

```
  return (
    <NavigationContainer>
      <Stack.Navigator initialRouteName={Screens.Menu}>
        <Stack.Screen name={Screens.Menu} component={MenuScreen} />
        <Stack.Screen name={Screens.MyIpInfo} component={MyIpInfoScreen} />
        <Stack.Screen name={Screens.FindIpInfo} component={FindByIp} />
        <Stack.Screen name={Screens.Traceroute} component={TraceRoute} />
      </Stack.Navigator>
    </NavigationContainer>
  );
};
```

src\navigation\types.ts:

```
import {FindByIpScreenProps} from '../screens/FindByIp/types';
import {MenuScreenProps} from '../screens/Menu/types';
import {MyIpInfoScreenProps} from '../screens/MyIpInfo/types';
import {SpeedTestScreenProps} from '../screens/SpeedTest/types';
import {TracerouteScreenProps} from '../screens/Traceroute/types';

export enum Screens {
  Menu = 'NetHealth',
  MyIpInfo = 'Моя IP адреса',
  FindIpInfo = 'Пошук по IP',
  SpeedTest = 'SpeedTest',
  Traceroute = 'Traceroute',
}

export type StackNavigatorScreens = {
  [Screens.Menu]: MenuScreenProps;
  [Screens.MyIpInfo]: MyIpInfoScreenProps;
  [Screens.FindIpInfo]: FindByIpScreenProps;
  [Screens.Traceroute]: TracerouteScreenProps;
  [Screens.SpeedTest]: SpeedTestScreenProps;
};
```

src\screens\FindByIp\FindByIp.tsx:

```
import React, {useEffect, useMemo, useState} from 'react';
import {Map} from '../../modules/Map';
import {useNetInfo} from '../../common/hooks';
import {View} from 'react-native';
```

```
import {Sections} from '../../components/sections';
import {styles} from './styles';
import {IpSearch} from '../../components/sections/IpSearch';

export const FindByIp = () => {
 const [searchedIpAddress, setSearchedIpAddress] = useState<
  string | undefined
 >(undefined);

 const {fetchLocationData, netInfo} = useNetInfo();

 const showInfo = useMemo(
  () => !!searchedIpAddress && !!netInfo,
  [netInfo, searchedIpAddress],
 );

 useEffect(() => {
  if (!searchedIpAddress) {
   return;
  }
  fetchLocationData(searchedIpAddress);
 }, [fetchLocationData, searchedIpAddress]);

 return (
  <View style={styles.container}>
   <IpSearch onSearch={setSearchedIpAddress} />

   {showInfo && netInfo && (
    <View>
     <Map
      coords={{
       latitude: netInfo?.latitude,
       longitude: netInfo?.longitude,
      }}
      markerTitle={searchedIpAddress}
     />

     <Sections.IpInfo
      ipAddress={searchedIpAddress!}
      isp={netInfo!.as}
      city={netInfo!.city_name}
      country={netInfo!.country_name}
      region={netInfo!.region_name}
```

```
          zip={netInfo!.zip_code}
        />
      </View>
    )}
  </View>
 );
};
```

src\screens\FindByIp\index.ts:

```
export * from './FindByIp';
```

src\screens\FindByIp\styles.ts:

```
import {StyleSheet} from 'react-native';
import {Colors} from '../../common/conts';

export const styles = StyleSheet.create({
 container: {backgroundColor: Colors.backgroundMain, flex: 1},
});
```

src\screens\FindByIp\types.ts:

```
export type FindByIpScreenProps = undefined;
```

src\screens\Menu\index.ts:

```
export * from './Menu';
```

src\screens\Menu\Menu.tsx:

```
import React from 'react';
import {Dimensions, View} from 'react-native';
import {Buttons} from '../../components/buttons';
import {GAP, PADDING, styles} from './styles';
import {Assets} from '../../common/assets';
import {useNavigation} from '@react-navigation/native';
import {Screens} from '../../navigation/types';

const {width} = Dimensions.get('window');

const MONO_TILE_WIDTH = width - PADDING * 2;
const TILE_SIZE = width / 2 - GAP / 2 - PADDING;
```

```
export const MenuScreen = () => {
 const navigation = useNavigation();

 const onMyIpAddressPress = () => {
  navigation.navigate(Screens.MyIpInfo as never);
 };

 const onFindByIPPress = () => {
  navigation.navigate(Screens.FindIpInfo as never);
 };

 const onTraceroutePress = () => {
  navigation.navigate(Screens.Traceroute as never);
 };

 return (
  <View style={styles.container}>
   <Buttons.Tile
    width={TILE_SIZE}
    height={TILE_SIZE}
    asset={Assets.myIpAddress}
    title="Моя IP адреса"
    onPress={onMyIpAddressPress}
   />
   <Buttons.Tile
    width={TILE_SIZE}
    height={TILE_SIZE}
    asset={Assets.anotherIpAddress}
    title={'Пошук по IP'}
    onPress={onFindByIPPress}
   />

   <Buttons.Tile
    width={MONO_TILE_WIDTH}
    height={TILE_SIZE}
    asset={Assets.tracking}
    title={'Traceroute'}
    onPress={onTraceroutePress}
   />
  </View>
 );
};
```

src\screens\Menu\styles.ts:

```
import {StyleSheet} from 'react-native';
import {Colors} from '../../common/conts';

export const GAP = 15;

export const PADDING = 20;

export const styles = StyleSheet.create({
  container: {
    flex: 1,
    gap: GAP,
    padding: PADDING,
    backgroundColor: Colors.backgroundMain,

    flexDirection: 'row',
    flexWrap: 'wrap',
  },
});
```

src\screens\Menu\types.ts:

```
export type MenuScreenProps = undefined;
```

src\screens\MyIpInfo\index.ts:

```
export * from './MyIpInfo';
```

src\screens\MyIpInfo\MyIpInfo.tsx:

```
import React, {useEffect, useMemo} from 'react';
import {Map} from '../../modules/Map';

import {useNetInfo} from '../../common/hooks';
import {Sections} from '../../components/sections';
import {StyleSheet, View} from 'react-native';
import {Colors} from '../../common/conts';

export const MyIpInfoScreen = () => {
  const {fetchMyNetInfo, netInfo, publicIpAdress, netType} = useNetInfo();

  const showInfo = useMemo(
```

```
  () => !!publicIpAdress && !!netInfo,
  [netInfo, publicIpAdress],
 );

 useEffect(() => {
  fetchMyNetInfo();
 }, [fetchMyNetInfo]);

 return (
  <View style={styles.container}>
   {showInfo && (
    <>
     <Map
      coords={{
       latitude: netInfo?.latitude,
       longitude: netInfo?.longitude,
      }}
      markerTitle={publicIpAdress}
     />

     <Sections.IpInfo
      ipAddress={publicIpAdress!}
      isp={netInfo!.as}
      city={netInfo!.city_name}
      country={netInfo!.country_name}
      region={netInfo!.region_name}
      zip={netInfo!.zip_code}
      netType={netType}
     />
    </>
   )}
  </View>
 );
};

const styles = StyleSheet.create({
 container: {backgroundColor: Colors.backgroundMain, flex: 1},
});
```

src\screens\MyIpInfo\types .ts:

```
export type MyIpInfoScreenProps = undefined;
```

src\screens\Traceroute\index.ts:

```
export * from './Traceroute';
```

src\screens\Traceroute\Traceroute.tsx:

```
import React, {useState} from 'react';
import {TracerouteHop} from '../../common/services/netInfo/types';
import {NetInfoService} from '../../common/services';
import {
  ActivityIndicator,
  ScrollView,
  StyleSheet,
  Text,
  View,
} from 'react-native';
import {Colors} from '../../common/conts';
import {Map} from '../../modules/Map';
import {IpSearch} from '../../components/sections/IpSearch';

export const TraceRoute = () => {
  const [hops, setHops] = useState<TracerouteHop[]>([]);
  const [full, setFull] = useState<string>('');
  const [loading, setLoading] = useState(false);

  const onSearch = async (ip: string) => {
    setLoading(true);
    await NetInfoService.traceroute(ip)
      .then(response => {
        setHops(response.hops);
        setFull(response.full);
      })
      .catch(console.log);
    setLoading(false);
  };

  return (
    <View style={styles.container}>
      <ScrollView>
        {/* Ip search input */}
        <IpSearch onSearch={onSearch} />

        {/* Loader */}
```

```
      {loading && (
        <View style={styles.loadingContainer}>
          <ActivityIndicator size={'large'} color={Colors.mainBlue} />
        </View>
      )}

      {/* Map */}
      {!loading && (
        <>
          <Map
            polyline={hops.map(hop => ({
              latitude: hop.latitude,
              longitude: hop.longitude,
              title: hop.ipAddress || '',
            }))}
          />

          {/* Full info */}
          <Text style={styles.fullInfo}>{full}</Text>
        </>
      )}
    </ScrollView>
  </View>
 );
};

export const styles = StyleSheet.create({
 container: {backgroundColor: Colors.backgroundMain, flex: 1},

 loadingContainer: {flex: 1, justifyContent: 'center'},

 fullInfo: {
  padding: 10,
  gap: 10,
 },
});
```

src\screens\Traceroute\types.ts:

```
export type TracerouteScreenProps = undefined;
```

Додаток Б. Код серверної частини

server\src\ipInfo\index.ts:

```
export * from './ipInfo.service';
```

server\src\ipInfo\ipInfo.service.ts:

```
import {GetIpInfoResponse} from './types';

export class IpInfo {
  private static KEY = '268285F2CA7F69DE0468108B77195CF8';

  static async getInfo(ip?: string): Promise<GetIpInfoResponse | null> {
    if (!ip) {
      throw Error('Not coorect ip');
    }

    return fetch(`https://api.ip2location.io/?key=${this.KEY}&ip=${ip}`).then(
      response => response.json(),
    );
  }
}
```

server\src\ipInfo\types.ts:

```
export type GetIpInfoResponse = {
  as: string;
  asn: string;
  city_name: string;
  country_code: string;
```

```
  country_name: string;
  ip: string;
  is_proxy: boolean;
  latitude: number;
  longitude: number;
  region_name: string;
  time_zone: string;
  zip_code: string;
};
```

server\src\traceroute\index.ts:

```
export * from './traceroute.service';
```

server\src\traceroute\traceroute.service.ts:

```
import {exec} from 'child_process';
import {IpInfo} from '../ipInfo';

type Hop = {
  ipAddress?: string;
  full: string;
  latitude?: number;
  longitude?: number;
};

export class Traceroute {
  static trace(ip: string): Promise<{hops: Hop[]; full: string}> {
    return new Promise((resolve, reject) => {
```

```
    exec(`traceroute ${ip}`, (error, stdout, stderr) => {
      if (error) {
        reject(stderr);
      }
      Traceroute.parseTraceroute(stdout)
        .then(hops => {
          resolve({hops, full: stdout});
        })
        .catch(reject);
    });
  });
}

private static async parseTraceroute(
  tracerouteResult: string,
): Promise<Hop[]> {
  const lines = tracerouteResult.trim().split('\n');
  const hops: Hop[] = [];

  for (const line of lines) {
    const ipAddress = Traceroute.extractIPFromTraceroute(line);

    try {
      const ipInfo = await IpInfo.getInfo(ipAddress);

      if (!ipInfo || !ipInfo.latitude || !ipInfo.longitude) {
        throw Error('Not coorect hop');
      }
```

```
        hops.push({
          ipAddress,
          full: line,
          latitude: ipInfo.latitude,
          longitude: ipInfo.longitude,
        });
      } catch (e) {
        console.log(e);
      }
    }

    return hops;
  }

  private static extractIPFromTraceroute(tracerouteResult: string) {
    const ipRegex = /\((\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})\)/g;

    return ipRegex.exec(tracerouteResult)?.[0].slice(1, -1);
  }
}
```

server\src\index.ts:

```
import express, {Express, Request, Response} from 'express';
import dotenv from 'dotenv';
import {Traceroute} from './traceroute';
import {IpInfo} from './ipInfo';

import cors from 'cors';
```

```
import bodyParser from 'body-parser';

dotenv.config();

const app: Express = express();
const port = process.env.PORT || 3000;

app.use(cors());
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({extended: true}));

app.set('trust proxy', true);

app.get('/', (req: Request, res: Response) => {
  res.send('Express + TypeScript Server');
});

app.get('/ping', (req, res) => {
  res.send('pong');
});

app.get('/traceroute', async (req: Request, res: Response) => {
  const ip = req.query.ip as string;

  try {
    const result = await Traceroute.trace(ip);
    res.status(200).send(result);
  } catch (error) {
```

```
    res.status(500).send(error);
  }
});

app.get('/ipInfo', async (req: Request, res: Response) => {
  const ip = req.query.ip as string;

  try {
    const response = await IpInfo.getInfo(ip);
    res.send(response);
  } catch (error) {
    res.status(500).send(error);
  }
});

app.get('/myIp', async (req: Request, res: Response) => {
  res.send(req.ip);
});

app.listen(port, () => {
  console.log(`[server]: Server is running at http://localhost:${port}`);
});
```