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)
18 грудня 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня магістр

зі спеціальності 122 - Комп'ютерних наук,
освітньо-професійної програми «Інформатика»
на тему: «Інформаційна технологія додатку для моніторингу цін на транзакції в блокчейні ethereum»
здобувача групи ІН.м - 22 Бондаря Арсенія Родіоновича

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

Арсеній БОНДАР
(підпис)

Керівник, Анна БАДАЛЯН
старший викладач
(підпис)

Суми – 2023

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

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

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня магістра

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»
здобувача групи ІН.м-22 Бондаря Арсенія Родіоновича

1. Тема роботи: «Інформаційна технологія додатку для моніторингу цін на транзакції в блокчейні ethereum»
затверджую наказом по СумДУ від *«06» грудня 2023 року № 1412-VI*

2. Термін здачі здобувачем кваліфікаційної роботи *до 18 друдня 2023 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, що використовуються для створення iOS додатків. 3) Розробка інтелектуальної системи для моніторингу цін на транзакції в блокчейні ethereum. 4) Аналіз результатів.

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 «__» _____ 20___ р.

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

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів кваліфікаційної роботи	Термін	Примітка

п/п		виконання	
1	*Аналіз проблеми предметної області, постановка й формування завдань дослідження*	08.12.2023	
2	*Огляд технологій для створення iOS додатків.*	09.12.2023	
3	*Розробка інтелектуальної системи для моніторингу цін на транзакції в блокчейні ethereum*	10.12.2023	
4	*Аналіз отриманих результатів*	15.12.2023	
5	*Оформлення пояснювальної записки до кваліфікаційної роботи*	16.12.2023	

Здобувач вищої освіти _____ Керівник _____

(підпис) (підпис)

АНОТАЦІЯ

Записка: 54 стр., 24 рис., 1 додаток, 31 використане джерело.

Обґрунтування актуальності теми роботи – тема кваліфікаційної роботи є актуальною, оскільки присвячена розв’язанню важливої практичної задачі моніторингу цін на транзакції в блокчейні ethtereum шляхом розробки інформаційної технології.

Об’єкт дослідження — процес моніторингу та аналізу цін на транзакції в мережі Ethereum, що включає збір, обробку та представлення даних про вартість газу в реальному часі.

Мета роботи — розробка інформаційної системи моніторингу цін на транзакції в блокчейні ethtereum.

Методи дослідження — технології створення iOS додатків.

Результати — розроблено додаток, що використовує сучасні технічні можливості та дизайнерські рішення, враховуючи специфіку цільової аудиторії. Програмний продукт забезпечує оперативний доступ до актуальної інформації про комісії в мережі ethereum, що важливо для користувачів при проведенні транзакцій.

ІНФОРМАЦІЙНА СИСТЕМА, iOS, БЛОКЧЕЙН, ETHTEREUM, XCODE, SWIFT, CORE DATA, USER NOTIFICATION.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми роботи

Актуальність. В епоху цифрових інновацій технологія блокчейну стрімко набуває популярності, переходячи від звичайного хобі до ключової складової різноманітних секторів економіки та інформаційних технологій. Блокчейн Ethereum є одним з найбільш перспективних, пропонуючи розширені можливості для розробки децентралізованих додатків та виконання транзакцій.

Особливістю Ethereum є використання для оплати транзакцій внутрішньої валюти - ефіру (ETH). Мінливість попиту та навантаження, складність виконання - тягне за собою необхідність постійного моніторингу цін на транзакції для оптимального планування витрат і ефективного управління фінансовими ресурсами.

У даній дипломній роботі акцентується увага на розробці спеціалізованого застосунку для операційної системи iOS, призначеного для моніторингу цін на транзакції в блокчейні Ethereum, який відповідає потребам широкого спектру користувачів, від розробників DApps до інвесторів та звичайних користувачів.

Об’єкт дослідження. Об'єктом дослідження є процес моніторингу та аналізу цін на транзакції в мережі Ethereum, що включає збір, обробку та представлення даних про вартість газу в реальному часі.

Предмет дослідження. Предметом дослідження є алгоритми та методи, які лежать в основі функціонування мобільного додатку для операційної системи iOS, призначені для моніторингу цін на транзакції Ethereum, зокрема механізми інтеграції з блокчейном, візуалізації даних та надання користувацького інтерфейсу.

Гіпотеза. Гіпотезою дослідження є припущення, що розробка мобільного додатку, оптимізованого для інтерактивного моніторингу цін на транзакції в блокчейні Ethereum, може значно поліпшити досвід користувачів, знизити їхні витрати на транзакції та підвищити загальну ефективність блокчейн-екосистеми.

Новизна. Новизна дослідження полягає в розробці інноваційного додатку для iOS, який використовує останні досягнення у галузі блокчейн-аналітики та мобільної розробки для створення оптимізованого, користувацьки орієнтованого інструменту. Додаток включає функціонал для оповіщення користувачів про оптимальні значення ціни на газ для проведення транзакцій, що є унікальним злиттям декількох технологій в один інструмент.

1 АНАЛІТИЧНИЙ ОГЛЯД

1.1 Визначення понять WEB3 та криптовалюта

Web3 і криптовалюти є взаємопов'язаними концепціями у світі цифрових технологій, особливо в контексті розвитку блокчейну та децентралізованих систем [1].

Web3 - це концепція нового покоління інтернету, яка прагне створити децентралізовану мережу. Основні характеристики web3:

- Децентралізація: замість централізованих серверів і платформ, web3 використовує децентралізовані технології, такі як блокчейн, для забезпечення розподіленого зберігання та обробки даних.
- Прозорість і безпека: web3 пропонує підвищену прозорість і безпеку даних за рахунок використання блокчейн-технологій.
- Користувацьке володіння та контроль: web3 акцентує увагу на володінні та контролі користувачем над його даними та цифровою ідентичністю.
- Інтероперабельність: web3 забезпечує сумісність і взаємодію між різними децентралізованими додатками (DApps).

Криптовалюта - це вид цифрової або віртуальної валюти, що використовує криптографію для забезпечення безпеки. Вона існує тільки в електронній формі та не має фізичних еквівалентів на кшталт паперових грошей або монет. Основна особливість криптовалюти - використання технології блокчейна, яка являє собою розподілений реєстр, що фіксує всі транзакції в мережі. Ці валюти зазвичай децентралізовані і не підкоряються контролю будь-якої центральної влади, що робить їх стійкими до втручання або маніпуляцій з боку урядів. Криптовалюти забезпечуються складними криптографічними алгоритмами, які захищають від шахрайства, такого як

подвійні витрати і підробка. Вони можуть пропонувати анонімність, хоча ступінь анонімності варіюється залежно від валюти. Більшість криптовалют мають обмежений запас, що означає, що існує межа кількості монет, які можуть бути створені. Створення нових одиниць криптовалюти часто відбувається через майнінг, де комп'ютери розв'язують складні математичні задачі для опрацювання транзакцій і додавання їх до блокчейну. Криптовалюти використовуються для різноманітних цілей, включно з інвестиціями, переказом коштів, покупками і як елемент нових фінансових і технологічних протоколів. Біткоін, створений 2009 року невідомою особою або групою людей під псевдонімом Сатоші Накамото [2], є першою і найвідомішою криптовалютою, за якою послідувало багато інших, включно з Ethereum, Solana і Litecoin.

Ключові особливості криптовалют:

- Децентралізація: більшість криптовалют працюють на децентралізованих мережах, заснованих на технології блокчейна.
- Безпека і прозорість: криптовалюти забезпечують високий рівень безпеки та прозорості транзакцій [3].
- Незалежність від центрального органу влади: криптовалюти не контролюються центральними банками або урядами.
- Цифровий характер: вони існують тільки в цифровій формі та використовуються для безготівкових транзакцій.
- Web3 і криптовалюти значною мірою спираються на блокчейн-технології. Блокчейн забезпечує інфраструктуру для децентралізованих додатків Веб3 і для створення та управління криптовалютами.

- Децентралізовані фінанси (DeFi): web3 включає в себе концепції DeFi, які безпосередньо пов'язані з використанням криптовалют для створення альтернативних фінансових систем і продуктів.
- Токенізація та економіка: web 3 використовує криптовалюти і токени для створення нових економічних моделей в інтернеті, включно із системами винагороди, управління і власності.
- Розумні Контракти: web3 і криптовалюти використовують смарт-контракти для автоматизації транзакцій і взаємодій у межах децентралізованих систем.

1.2 Чому саме Ethereum

Вибір блокчейну Ethereum для розробки та впровадження проектів має безліч причин, ґрунтуючись на його унікальних характеристиках і можливостях. Ось деякі з ключових аспектів, чому Ethereum є кращим вибором для багатьох розробників [4]:

- Він дозволяє розробникам створювати складні смарт-контракти, використовуючи мову програмування Solidity. Це забезпечує високий ступінь гнучкості та контролю над функціональністю. Смарт-контракти автоматично виконуються при виконанні певних умов, що зменшує необхідність у посередниках і знижує ризики людської помилки.
- Також є підтримка децентралізованих застосунків (DApps), включно з фінансовими сервісами, іграми, соціальними мережами та багато іншого.
- Блокчейн Ethereum має одну з найбільших і найактивніших спільнот розробників у світі web3, що забезпечує постійний розвиток і підтримку.

- Безпека та надійність – основні стовпи блокчейну Ethereum. Його було запущено у 2015 році, і відтоді він довів свою стабільність як платформа для розробки блокчейн-додатків.
- Перехід з PoW на PoS у 2022 році був запроваджений для поліпшення безпеки, масштабованості та швидкості обробки транзакцій. Також це відкриває двері для наступних оновлень.
- Ethereum популяризував стандарти токенів ERC-20, ERC-721 та ERC-1155. Вони є основою для більшості цифрових токенів.

Ethereum пропонує потужну комбінацію гнучкості, безпеки, масштабованості та підтримки спільноти, роблячи його ідеальною платформою для різноманітних блокчейн-додатків та інновацій. Він постійно розвивається, щоб відповідати мінливим вимогам індустрії та залишається одним з найвпливовіших і найбільш популярних блокчейнів на сьогоднішній день [5].

1.3 Що таке PoW та PoS, різниця між ними

У контексті блокчейна Ethereum, Proof of Work (PoW) і Proof of Stake (PoS) є двома різними алгоритмами консенсусу, які використовують для підтвердження транзакцій і створення нових блоків у мережі.

PoW є оригінальним алгоритмом консенсусу, що існував в Ethereum з моменту його запуску. Цей метод вимагає від майнерів розв'язання складних математичних задач для додавання нових блоків у блокчейн [6].

Розглянемо переваги:

- В першу чергу – безпека. Складність завдань PoW забезпечує низький рівень колапсу мережі, оскільки для атаки на мережу потрібна величезна кількість обчислювальної потужності.
- В ідеальному світі сприяє децентралізації, оскільки будь-хто з відповідним обладнанням може брати участь у майнінгу.

Недоліки:

- PoW вимагає значних обчислювальних ресурсів та енергії, що викликає стурбованість з точки зору екології.
- Насправді майнінг стає дедалі централізованішим через високі витрати на обладнання та електроенергію.
- PoW обмежує швидкість транзакцій і масштабованість мережі.

PoS - це альтернативний алгоритм консенсусу, який Ethereum планує повністю впровадити в межах оновлення Ethereum 2.0 [7]. У PoS валідатори блоків обираються на основі кількості монет, які вони тримають у смарт-контракті, а не на основі їхньої обчислювальної потужності.

Переваги:

- PoS значно енергоефективніший порівняно з PoW, оскільки не потребує інтенсивних обчислень.
- Дає змогу обробляти транзакції швидше, що покращує масштабованість мережі.
- PoS забезпечує безпеку мережі через економічні стимули, оскільки пропонує від 2 до 10% на рік, що нараховуються на заблоковані монети Eth у смарт-контракті.

Недоліки:

- Учасники, які володіють великою кількістю монет, можуть мати більший вплив на процес валідації.
- Оскільки PoS є відносно новим, його довгострокова безпека та стійкість до різних атак все ще оцінюються.
- Перехід з PoW на PoS є складним процесом, що вимагає значних змін у коді блокчейна.

Кожен із цих алгоритмів має свої унікальні переваги та недоліки. PoW довів свою надійність і безпеку протягом багатьох років, але його недоліки в плані енергоспоживання і масштабованості стали критичними. PoS пропонує рішення цих проблем, збільшуючи ефективність і масштабованість мережі, але його довгострокова надійність і безпека ще будуть перевірятися з часом.

1.4 Основні можливості Ethereum

Блокчейн Ethereum відкрив нові горизонти для розробки та використання децентралізованих технологій. Ось основні можливості, які пропонує блокчейн Ethereum:

- Розробка і розгортання автономних смарт-контрактів, які виконуються при настанні заданих умов без втручання третіх сторін. Їх використовують для автоматизації юридичних і комерційних угод, спрощуючи процеси і знижуючи витрати.
- Різноманітність застосунків у найрізноманітніших галузях, від фінансів і страхування до соціальних медіа та ігор, що працюють без центрального контролю. Це підвищує їхню надійність і безпеку.

- Створення децентралізованих платформ для кредитування, торгівлі та управління активами, стейкінгу та надання ліквідності для отримання доходу.
- Ethereum дозволяє випускати та торгувати унікальними цифровими активами (NFT). Це може бути мистецтво, музика, відео та інші колекційні предмети. Технологію NFT використовують для підтвердження авторства і прав на цифрові активи.
- Створення децентралізованих автономних організацій (DAO) з демократичним управлінням, де всі рішення ухвалюються спільнотою. Це дозволяє керувати корпоративними ресурсами та фондами з повною прозорістю та звітністю.
- Розробка ігор з використанням криптовалют, NFT і смарт-контрактів, де гравці володіють внутрішньоігровими предметами.

1.5 Що таке газ у блокчейні Ethereum

Газ в Ethereum - це міра, що використовується для кількісної оцінки обсягу обчислювальних зусиль, необхідних для виконання операцій або смарт-контрактів у мережі Ethereum [8]. Газ вимірює, скільки "палива" буде потрібно для виконання тієї чи іншої операції або смарт-контракту. Кожна дія в Ethereum вимагає певної кількості газу: прості транзакції споживають менше газу, тоді як складні операції зі смарт-контрактами вимагають більшого обсягу. Газ забезпечує обмеження ресурсів, запобігаючи зловживанням, як-от нескінченні цикли в смарт-контрактах або перевантаження мережі безглуздими обчисленнями. Вартість газу допомагає захистити мережу від спам-атак, оскільки зловмисникам доводиться платити за кожну операцію.

Користувачі встановлюють ціну газу (зазвичай вимірювану в Gwei [9], де 1 Gwei дорівнює 10^-9 ефірів) для своїх транзакцій. Висока ціна газу може прискорити виконання транзакції, оскільки майнери, які валідують транзакції, схильні вибирати ті, що приносять більше доходу. Також є можливість встановлювати ліміт газу для кожної транзакції, який користувач готовий витратити. Якщо транзакція потребує більше газу, ніж встановлено в ліміті, вона не буде виконана. Підсумкова вартість транзакції в Ethereum розраховується шляхом множення використаної кількості газу на ціну газу.

Розуміння механізму газу важливе для ефективної взаємодії з блокчейном Ethereum, оскільки це дає змогу користувачам оптимально налаштувати свої транзакції з погляду вартості та швидкості виконання.

1.6 Etherscan

Etherscan є одним із найвідоміших і найбільш широко використовуваних оглядачів блокчейна Ethereum. Цей веб-додаток дає змогу користувачам відстежувати транзакції, їх статус, відправлені та отримані кошти, комісії. І все це доступно в реальному часі. Etherscan надає можливість перегляду коду смарт-контрактів, їхньої активності та взаємодій з іншими контрактами й адресами, аналітичні інструменти і статистику по мережі Ethereum, такі як інформація про газ, токени ERC-20, DeFi і багато іншого [10].

Etherscan автоматично збирає дані з блокчейна Ethereum, використовуючи вузли мережі для отримання інформації про блоки, транзакції та смарт-контракти. Додаток надає зручний і зрозумілий інтерфейс для перегляду та аналізу даних блокчейна. Користувачі можуть шукати інформацію за адресами гаманців, хешами транзакцій, блоками і адресами

смарт-контрактів. Також Etherscan надає API, який дає змогу розробникам інтегрувати дані блокчейна Ethereum у свої додатки та сервіси.

1.7 Аналіз відомих рішень

Перед розробкою додатку для моніторингу цін на транзакції в блокчейні Ethereum важливо ознайомитися з існуючими аналогами. Розуміння того, що вже пропонують конкуренти, допомагає визначити їхні сильні та слабкі сторони. Це може вказати на незадоволені потреби користувачів або можливості для покращення нового додатку, зокрема інтерфейсу та функціональності. Знання про проблеми та обмеження існуючих рішень може допомогти уникнути подібних недоліків у власному проекті.

Почнемо огляд з додатку GAS [11]. Його фунціонал дійсно вражає:

- Універсальні комісії за мережеві дії.
- Зручний погодинний графік цін на газ.
- Можливість відстеження цін на криптовалюту.
- Функція пуш-сповіщень.

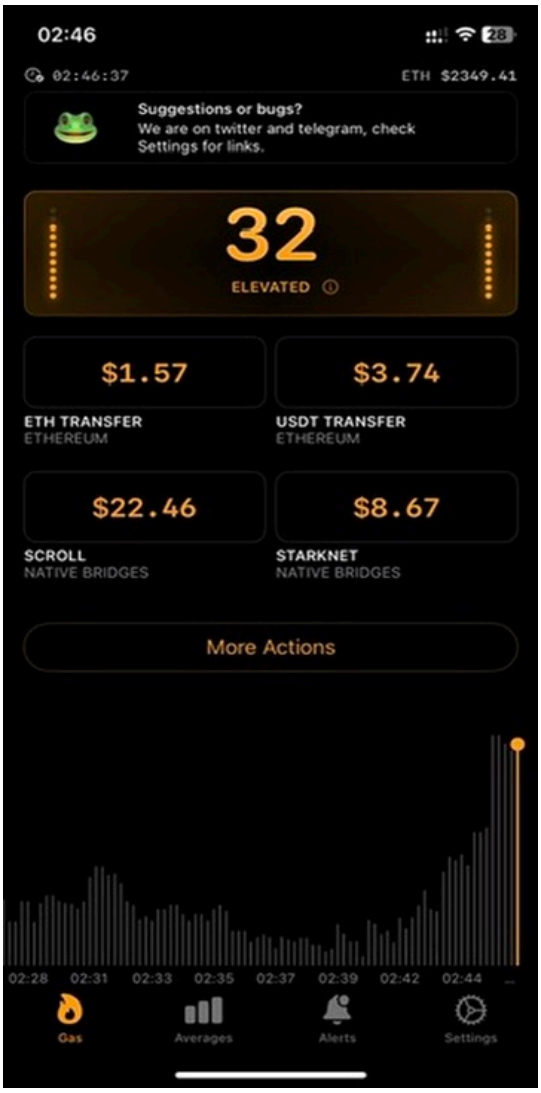

Рисунок 1.1 – Головна сторінка Gas

Рисунок 1.2 – Погодинний графік цін

Попри всі переваги додатку, наявні суттєві мінуси. Є реклама, доволі дорога підписка, яка відкриває усі можливості додатку. Наприклад – доступ до пуш-сповіщень про зміну ціни на газ в блокчейні Ethereum.

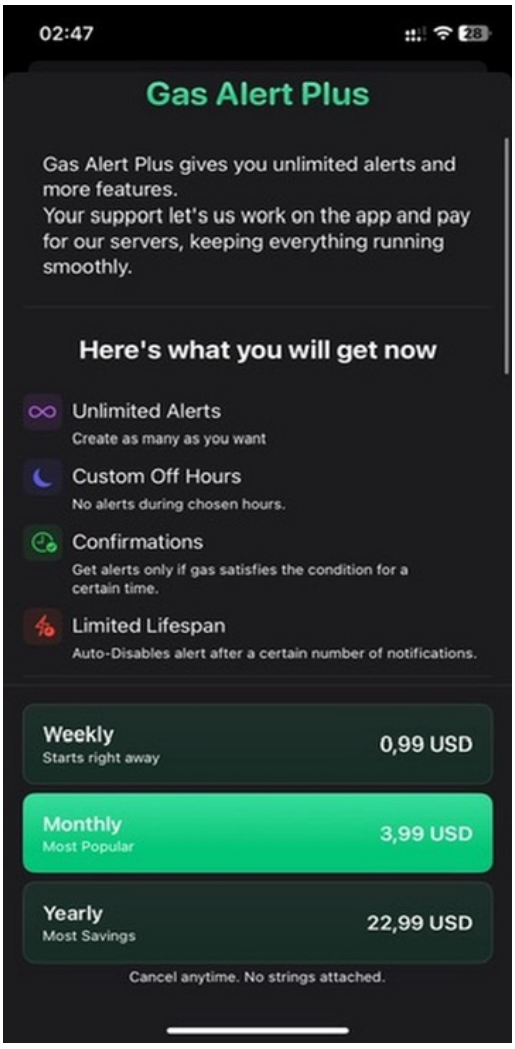

Рисунок 1.3 – Сторінка для оформлення підписки

Наступним додатком в огляді буде GasPump [12]. Як для повністю безкшотовного – пропонує не погані можливості для мониторінгу. Нажаль, провести тестування не вдалося. Для демонстрації роботи додатку я зробив 3 перезавантяження з видаленням із оперативної пам’яті, але результату отримано не було. Пристрій для запуску та тестування – iPhone 12, версія iOS – 17.2 (останнє оновлення системи). Скріншот додатку в момент зависання далі на рисунку.

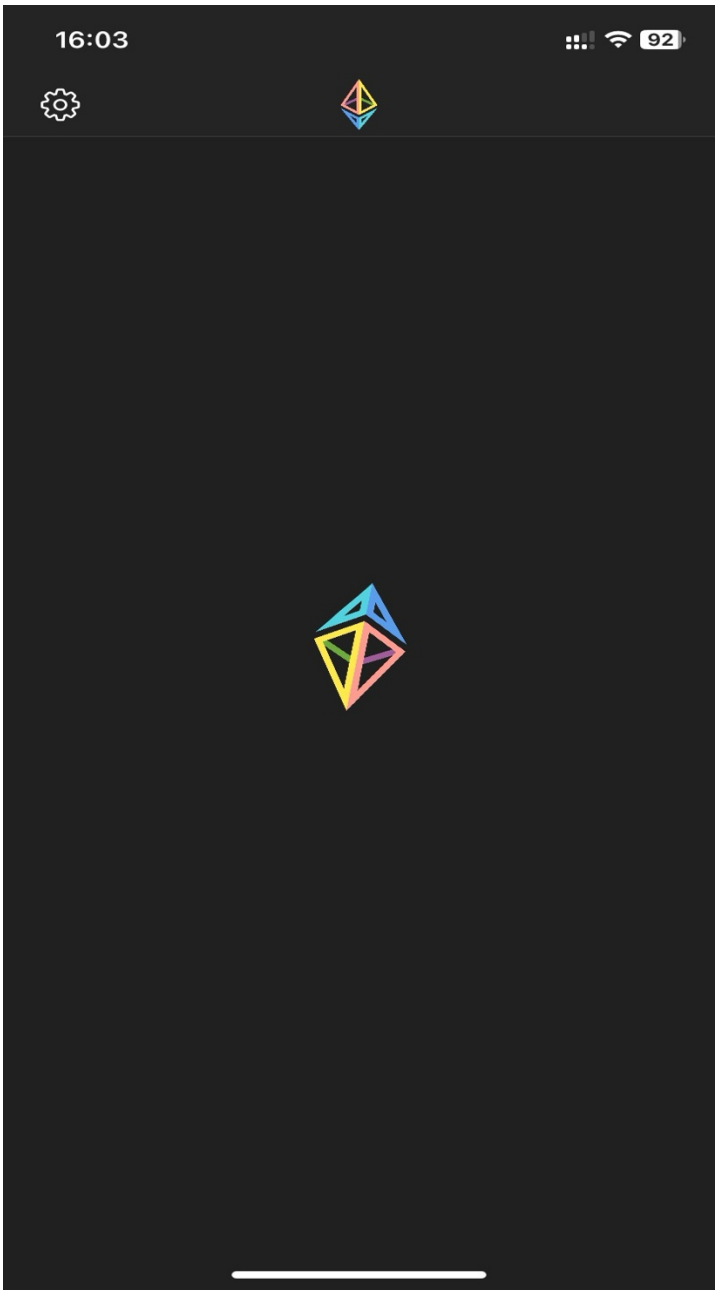

Рисунок 1.4 – Процес запуску GasPump

ETH Gas Fees – наступний претендент [13]. Має лаконічний інтерфейс, позбавлений зайвих деталей. Хочу відмітити швидку роботу та мале споживання ресурсів пристрою та батареї. Але додаток не обділений недоліками. До них я би відніс нав'язливу рекламу та відсутність ключових

функцій. Наприклад, пуш-сповіщення про різку зміну цін. Демонстрація роботи на наступному рисунку.

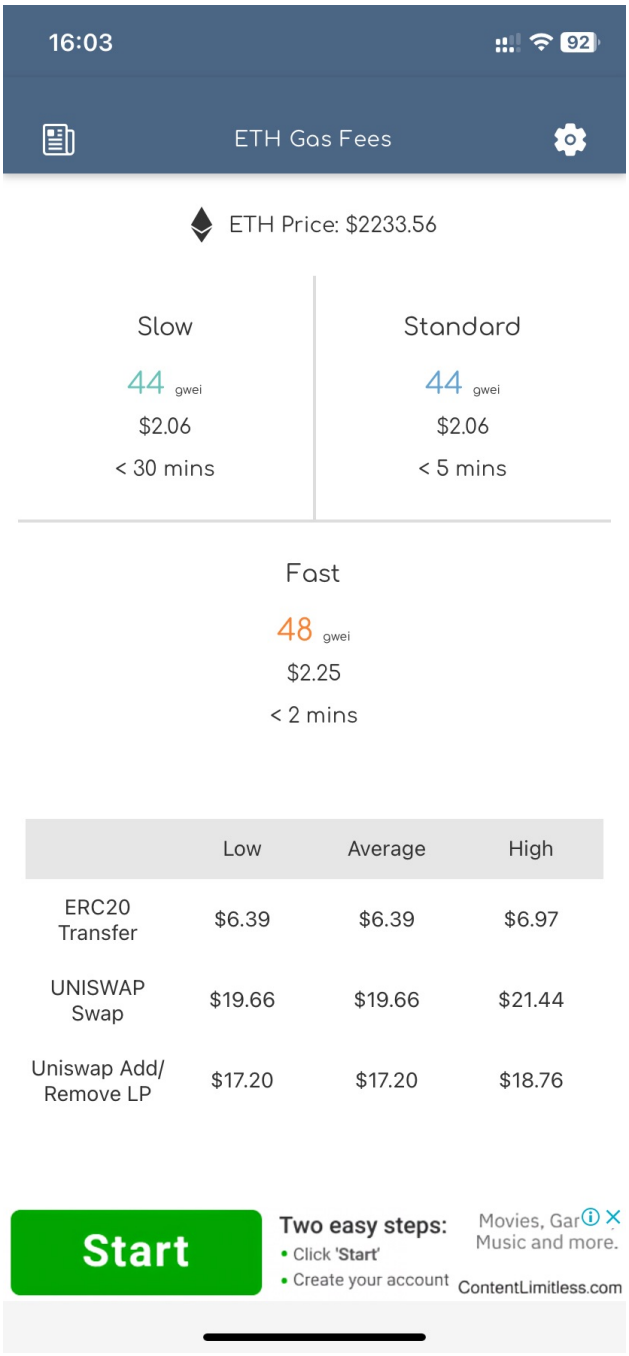

Рисунок 1.5 – Головна сторінка ETH Gas Fees

Додаток widgETH [14] також має зручний інтерфейс, відстуність щомісячних платежів, темну тему та попри все не позбавлений критичних недоліків. До них я би відніс надто повільне оновлення даних, відсутність пуш-сповіщень та останнє оновлення у 2021 році. Тобто розробник не надає своєчасну підтримку.

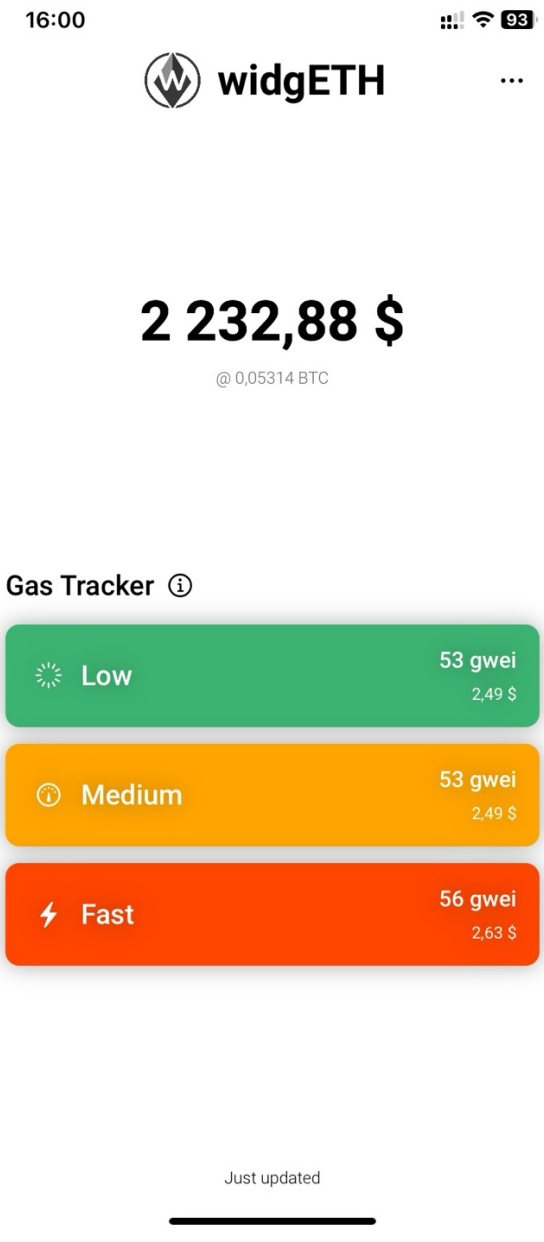

Рисунок 1.6 – Додаток widgETH

Наступним в огляді додаток Ethereum Gas Price [15]. Має непоганий функціонал, як для безкоштовної моделі розповсюдження. Але одразу впадає в очі перегружений інтерфейс, який у 2023 виглядає застарілим. Також до недоліків я би відніс пуш-сповіщення, які приходять із затримкою 5-10 хвилин, що свідчить про погану оптимізацію фонових задач.

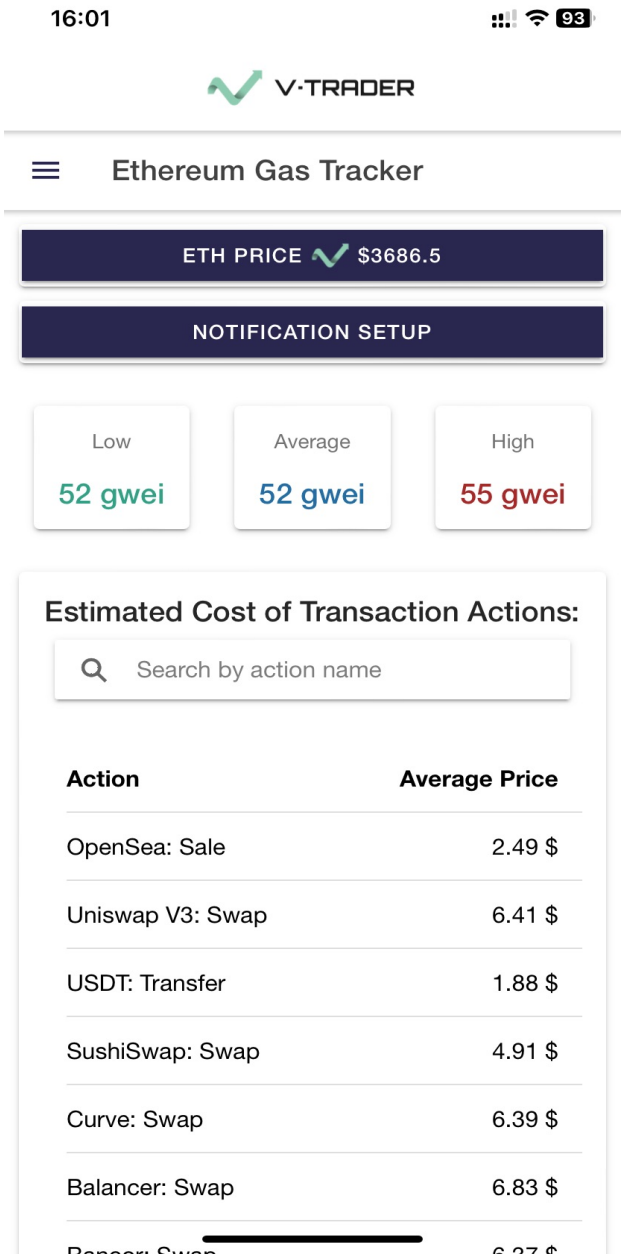

Рисунок 1.7 – Додаток Ethereum Gas Price

1.7 Постановка задачі

Для досягнення поставленої мети необхідно вирішити наступні завдання:

- Вивчення особливостей блокчейну Ethereum і механізмів формування ціни на транзакції.
- Аналіз наявних рішень для моніторингу цін на транзакції в блокчейні та виявлення їхніх недоліків.
- Розробка концепції та архітектури програми для iOS.
- Програмна реалізація додатка з використанням сучасних технологій розробки мобільних додатків.
- Тестування та оптимізація додатка для забезпечення його стабільної роботи та високої продуктивності.
- Реалізація цієї дипломної роботи має важливе практичне значення, оскільки сприяє поліпшенню взаємодії користувачів із блокчейном Ethereum, надаючи актуальні дані для ухвалення зважених рішень у сфері цифрових фінансів і криптовалют.

Результатом роботи має стати додаток для моніторингу цін на транзакції в блокчейні Ethereum.

2 ВИБІР МЕТОДУ РІШЕННЯ

2.1 Інструменти та теоретична частина розробки

Apple залишається однією з провідних компаній у галузі виробництва смартфонів, ноутбуків та іншої електроніки [16], і особлива тим, що розробляє власне програмне забезпечення для всіх своїх пристроїв. Компанія надає розробникам все необхідне для створення додатків для iOS та macOS. Сюди входить інтегроване середовище розробки Xcode, мови програмування Swift та Objective-C, а також широкий спектр бібліотек та фреймворків. Кожного року в Каліфорнії проводиться Worldwide Developers Conference (WWDC), де Apple представляє новинки, розробки та оновлення своїх платформ [17].

Xcode, ключовий інструмент розробки для tvOS, watchOS, macOS та iOS, забезпечує інструментарій для створення додатків для всіх актуальних платформ Apple, включаючи Mac, iPad, iPhone, Apple Watch та Apple TV [18]. Середовище Xcode 15 [19], яке тепер займає на 45% менше місця на комп'ютері Mac, включає у себе покращене автодоповнення коду, що дозволяє швидше писати більш безпечний код, інтерактивні попередні перегляди та живі анімації, а також новітній компілятор коду. Все це оптимізовано під архітектуру Apple silicon. За допомогою Xcode 15 та Xcode Cloud, які тісно інтегровані між собою, розробники можуть швидко розпочати процес створення додатків, отримувати оновлення статусу збірки в реальному часі, генерувати звіти та автоматично розподіляти додатки серед тестерів та користувачів. Також наявний інструмент для навчання початківців – Swift Playground.

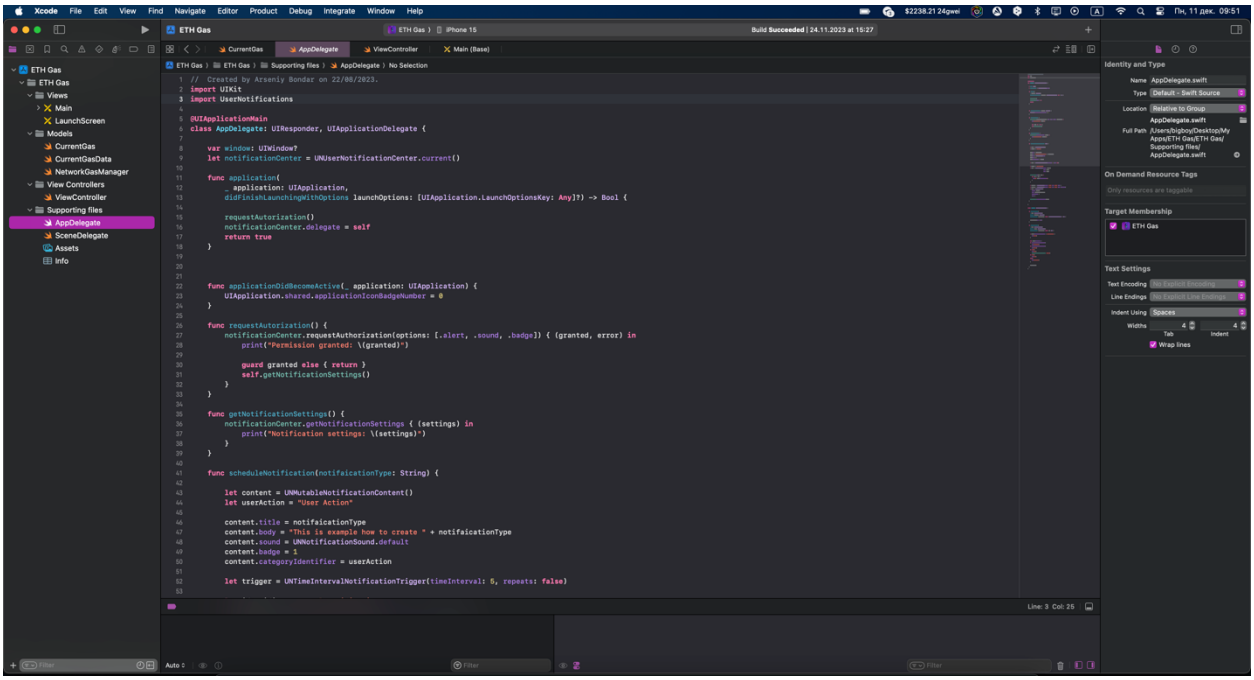

Рисунок 2.1 – Робоча область Xcode

2.2 Вибір мови реалізації

Традиційно розробка програм для iOS та MacOS опиралася на використання мови Objective-C. Однак з появою Swift з'явилася альтернатива з великим потенціалом. Незважаючи на те, що Objective-C досі застосовується, вона вважається застарілою. Swift відрізняється легкістю у навчанні та сприяє прискоренню розробки. Apple мала мету створити мову програмування, що була б інтуїтивно зрозумілою для початківців та ефективною у процесі створення програмного забезпечення.

Swift має характеристики мови програмування нової генерації і відчутно перевершує Objective-C. Ця мова включає ряд функцій безпеки: суворе відношення до null-значень, захист від помилок у розмірах масивів,

автоматичне управління пам'яттю, запобігання переповненню, а також усуває неініціалізовані змінні. Ці особливості допомагають розробникам сконцентруватися на реалізації ідей, а не на виправленні помилок у коді. Swift також спрощує синтаксис, роблячи код легшим для написання та розуміння. За інформацією від Apple, Swift у декілька разів швидший, ніж Python, та в 2,2 рази швидший, ніж Objective-C [20].

Swift вирізняється не тільки своєю швидкістю, але й багатим набором сучасних функцій, які підтримують функціональне програмування. Це включає ітератори, замикання, можливість множинного повернення значень, дженерики, функціональні шаблони та кортежі. Такі інновації, поряд з удосконаленням синтаксису, роблять Swift більш безпечним, знижуючи ризики, пов'язані з управлінням пам'яттю та помилками в даних, а також спрощують обробку помилок [21].

Apple представила Swift як відкриту мову програмування, що стало несподіваним ходом для компанії такого масштабу. Завдяки статусу відкритого коду мова постійно вдосконалюється спільнотою розробників. Згідно зі звітом GitHub Octoverse 2023 [22] та опитуванням StackOverflow 2023 [23], Swift займає високі позиції серед розробників за популярністю і перевагами, а його значення та популярність з часом будуть тільки зростати.

2.3 Архітектура iOS

iOS - це операційна система для мобільних пристроїв, створена компанією Apple, яка використовується в iPhone, iPad та інших мобільних пристроях від Apple. За популярністю та використанням, iOS посідає друге місце після Android [24].

Структура iOS організована на декілька рівнів, які не взаємодіють безпосередньо між собою. На нижньому рівні знаходяться базові сервіси, які є фундаментом для всіх програм. Вищі рівні забезпечують графічні та інтерфейсні послуги. Більшість системних інтерфейсів в iOS доступні через спеціалізовані пакети, відомі як фреймворки [25].

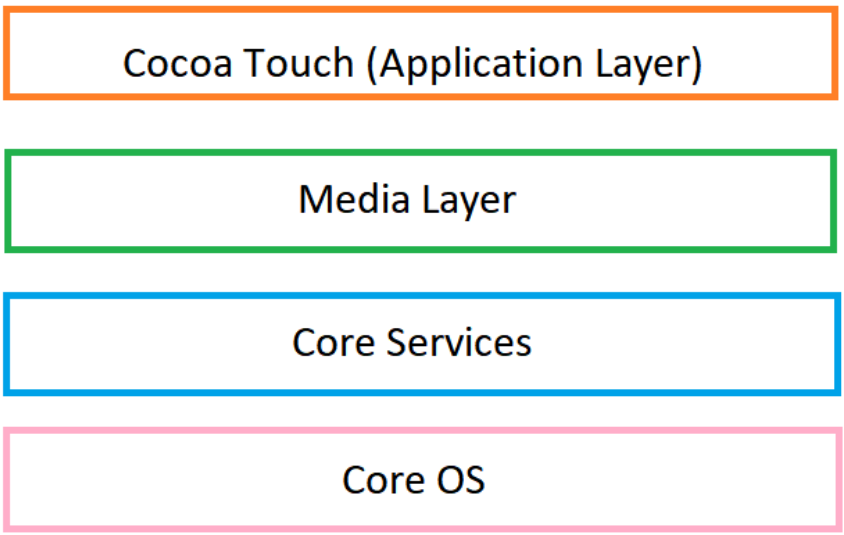

Рисунок 2.1 – Архітектура iOS

Фреймворк - це набір рекомендацій, інструментів та бібліотек, який спрощує процес розробки програмного забезпечення. Він надає вже готову структуру та шаблони для пришвидшення створення програм, дозволяючи розробникам зосередитися на унікальних аспектах своїх проектів, замість того, щоб витрачати час на вирішення звичайних технічних завдань. Фреймворки часто містять бібліотеки для виконання розширених завдань. Це може бути управління даними, взаємодія з базами даних, графічний інтерфейс користувача та інші елементи, які потрібні для розробки сучасних програм.

Використання фреймворку може значно підвищити ефективність і якість кінцевого продукту [26].

Розглянемо більш детально кожен рівень операційної системи:

COCOA TOUCH:

Цей шар функціонує як інтерфейс для взаємодії користувача з операційною системою iOS. Він обробляє дотики до екрану, анімації та багато інших можливостей. Cocoa Touch включає такі фреймворки [27]:

- PushKit Framework
- MapKit Framework
- GameKit Framework
- EvenKit Framework

MEDIA Layer:

Він є одним із основних компонентів архітектури операційної системи, відповідає за обробку, відображення та управління різними типами медіа-контенту на пристроях Apple. Ось основні аспекти та можливості Media Layer в iOS:

- GL Kit
- Core Images
- AV Kit
- Media Player
- Framework
- Open AL
- Core Animation
- Core Graphics Framework
- ULKit Graphics

CORE SERVICES Layer:

Ряд ключових фреймворків розташовані на цьому рівні, що сприяє поліпшенню функціоналу операційної системи iOS. Вони формують другий рівень системної архітектури. Ось деякі з основних фреймворків, які вбудовані у цей шар:

- StoreKit Framework
- Social Framework
- HomeKit Framework
- HealthKit Framework
- Core Location Framework
- Core Foundation Framework
- Address Book Framework
- Foundation Framework
- Cloud Kit Framework
- Core Data Framework
- Core Motion Framework

CORE OS Layer:

В архітектурі операційної системи - Core OS є нижнім рівнем системи, який забезпечує основні сервіси та функції операційної системи. Цей шар знаходиться безпосередньо над апаратним забезпеченням і забезпечує взаємодію між високорівневим програмним забезпеченням і фізичними компонентами пристрою.

- Local Authorization Framework
- Security Services Framework
- Accelerate Framework
- External Accessories Framework

- Core Bluetooth Framework

До переваг системи iOS відносять:

- Високий рівень безпеки в iOS, який досягається завдяки регулярним оновленням захисних систем і суворій політиці щодо додатків в App Store.
- Тісна інтеграція між програмним забезпеченням та апаратною частиною виробів Apple забезпечує високий рівень оптимізації і стабільності.
- Системи та пристрої Apple створені для гармонійної взаємодії, що забезпечує відмінну узгодженість праці між різними пристроями.
- iOS відрізняється зрозумілим і привабливим дизайном інтерфейсу, що забезпечує зручність використання.
- Apple завжди забезпечує надійну підтримку та своєчасні оновлення для своїх пристроїв, вводячи нові можливості та захисні функції.

Тим не менше, є деякі обмеження:

- Усі пристрої Apple мать вищу вартість порівняно з продукцією конкурентів.
- iOS пропонує значно менше варіантів для персоналізації в порівнянні з Android.
- Пристрої на базі iOS можуть мати проблеми зі сумісністю з продукцією інших виробників.
- Вартість ремонту та обслуговування продукції Apple може бути високою і обмеженою авторизованими сервісними центрами.
- iOS не надає такого ж рівня доступу до файлової системи, як це робить Android, що може ускладнити управління файлами.

2.4 Фреймворк Core Data

Core Data - це фреймворк для управління моделлю даних у додатках iOS і macOS, розроблений Apple. Він надає об'єктно-орієнтоване API для роботи з даними, що дає змогу розробникам зручно й ефективно керувати складними даними та відносинами між ними. Також включає в себе створення, читання, оновлення та видалення даних (CRUD) [28].

Ось деякі переваги фреймворку:

- Core Data дає змогу працювати з даними як з об'єктами, що спрощує програмування і підвищує читабельність коду.
- Фреймворк автоматизує багато аспектів управління даними, як-от відстеження змін, кешування, управління зв'язками, відкладене завантаження і збереження даних.
- Core Data оптимізовано для роботи з великими обсягами даних і комплексними моделями даних, забезпечуючи високу продуктивність додатків.
- Core Data тісно інтегрований з іншими частинами екосистеми iOS і macOS, включно з UIKit і SwiftUI, що полегшує створення інтерфейсів, які автоматично оновлюються під час зміни даних.
- Фреймворк підтримує різні механізми зберігання даних, включно з SQLite, XML, бінарними файлами і навіть пам'яттю пристрою. Оптимізує запити до бази даних і ефективно використовує кешування для поліпшення продуктивності.

2.5 Фреймворк UIKit

UIKit - це основний фреймворк, який використовується для розробки користувацького інтерфейсу на платформах iOS, tvOS і watchOS, розроблений компанією Apple. Він забезпечує інфраструктуру для побудови графічних інтерфейсів у застосунках і відіграє центральну роль у розробці застосунків для пристроїв Apple [29].

UIKit надає розробникам стандартний набір елементів керування, уявлень та інших інтерфейсних об'єктів, що забезпечує однаковість і впізнаваність інтерфейсів додатків на платформі Apple. Він діє як проміжний шар між апаратним забезпеченням пристрою і додатками, спрощуючи обробку сенсорних вхідних даних, жестів, рухів та інших користувацьких дій. Фреймворк підтримує адаптивний дизайн, що дає змогу застосункам ефективно працювати на пристроях із різними розмірами та роздільною здатністю екрана, зокрема iPhone та iPad. UIKit забезпечує багаті можливості для створення анімацій і графіки, що підвищує привабливість і динамічність користувацького інтерфейсу. Фреймворк допомагає керувати багатозадачністю в додатках, забезпечуючи плавне перемикання між завданнями та оптимальне використання ресурсів пристрою.

Ключові компоненти та функціональність UIKit:

- Кнопки, перемикачі, слайдери та інші елементи керування, які дають змогу користувачам взаємодіяти з додатком.
- Базові компоненти для побудови користувацького інтерфейсу, включно з UIView і його підкласами, які визначають вміст екрана і його візуальну презентацію.
- View Controllers керують відображенням уявлень на екрані, організацією потоку даних і обробкою взаємодій користувача.
- UIKit взаємодіє з Core Animation для створення плавних і складних анімацій інтерфейсу.

- Обробка дотиків, жестів, прискорень та інших форм введення, що робить взаємодію з застосунком інтуїтивно зрозумілою і зручною.
- Наявні інструменти для створення гнучких інтерфейсів, що адаптуються до різних розмірів та орієнтацій екрана.
- UIKit тісно інтегрований з іншими фреймворками Apple, як-от Core Data для управління даними, Core Graphics для малювання графіки та багатьма іншими.

2.6 Фреймворк Foundation

Фреймворк Foundation є одним з основних компонентів програмного середовища Apple, призначеним для розробки як на macOS, так і на iOS, tvOS і watchOS. Він надає базовий рівень функціональності для всіх додатків і фреймворків на платформах Apple [30].

Foundation визначає базові типи даних і структури, як-от рядки, дати та числа, що є фундаментом для складніших операцій і структур даних у додатках. Він пропонує зручні засоби для роботи з колекціями даних (масивами, словниками, множинами) і забезпечує їхнє ефективне управління. Фреймворк містить інструменти для читання і запису файлів, а також для здійснення мережевих операцій, підтримує локалізацію та інтернаціоналізацію, що дає змогу додаткам працювати з різними мовами та культурними налаштуваннями. Foundation працює з датами і часом, включно з часовими зонами і календарними обчисленнями, містить механізми для ефективного оброблення помилок і логування, що критично важливо для розроблення надійних застосунків.

Основний функціонал фреймворку:

- NSString, NSNumber, NSDate, NSData та інші типи даних, що забезпечують стандартизовані способи представлення та обробки.
- Колекції NSArray, NSDictionary, NSSet та їхні варіанти, що дають змогу ефективно керувати групами даних.
- Класи для роботи з файлами і директоріями, такі як NSFileManager і NSURL, надають засоби для читання, запису та управління файловими системами.
- Клас NSURLSession надає можливості для виконання HTTP-запитів, завантаження даних і взаємодії з мережею.
- Foundation надає інструменти для багатопотоковості та асинхронного виконання завдань, такі як NSThread, NSOperation і NSOperationQueue.
- Підтримка для кодування об'єктів і їхньої серіалізації у формати, як-от JSON або XML, за допомогою NSCoder і пов'язаних класів.
- NSTimer та інші утиліти для роботи з часом і планування завдань.
- Різні утиліти для маніпуляцій із рядками, включно з регулярними виразами.

2.7 Фреймворк UserNotification

Фреймворк UserNotifications в екосистемі Apple призначений для управління доставкою та відображенням локальних і віддалених повідомлень в iOS, macOS, watchOS і tvOS. Цей фреймворк відіграє ключову роль у залученні користувачів, інформуванні їх про важливі події та дані, навіть коли застосунок не активний [31].

Ось деякі переваги фреймворку:

- Дає змогу застосункам планувати локальні сповіщення та обробляти віддалені (push-сповіщення), які надсилають із сервера. Розробники налаштовують час і умови для доставлення сповіщень, включно з повторюваними сповіщеннями.
- Є можливість додавання дій, як-от кнопки для виконання певних оперцацій.
- Користувацькі дані та медіа-контент можуть бути інтегровані в повідомлення для створення більш персоналізованої взаємодії з користувачем.
- Загальний підхід до сповіщень на всіх платформах Apple, що спрощує розробку міжплатформних додатків.

Ключові компоненти та функціональність UserNotifications:

- UNUserNotificationCenter: центральний об'єкт, що керує доставкою повідомлень. Він використовується для налаштування і планування локальних повідомлень, а також для обробки інформації про віддалені повідомлення, що надходять.
- UNNotificationRequest: об'єкт, що представляє запит на показ повідомлення. Це включає в себе тригер повідомлення (який визначає, коли повідомлення має бути доставлено) і вміст повідомлення.
- Різні типи тригерів (UNCalendarNotificationTrigger, UNTimeIntervalNotificationTrigger, UNLocationNotificationTrigger), які визначають умови для доставки сповіщень.
- UNMutableNotificationContent використовується для визначення вмісту сповіщення, включно із заголовком, текстом повідомлення, звуком і прикріпленими медіа-файлами.

- Фреймворк надає механізми для запиту дозволу користувача на показ сповіщень і налаштування налаштувань сповіщень.

2.8 Grand Central Dispatch

Grand Central Dispatch (GCD) - це технологія багатозадачності на рівні системи, розроблена Apple для платформ iOS і macOS. GCD надає ефективний і легкий спосіб керування асинхронними операціями та багатопотоковістю, покращуючи продуктивність додатків та швидкість відгуку.

Робота з потоками може бути складною, особливо під час керування великою кількістю асинхронних операцій. GCD спрощує цей процес, автоматично керуючи пулом потоків і виконуючи завдання на оптимальних потоках. Технологія дає змогу застосункам виконувати завдання асинхронно (наприклад, завантаження даних або складні обчислення) без блокування основного потоку, що забезпечує плавність інтерфейсу та швидкість відгуку застосунків, максимізує продуктивність і знижує ймовірність уповільнення або зависання застосунку.

GCD працює на рівні системи, що означає, що він краще, ніж розробники, знає, як ефективно розподілити завдання на доступні ядра процесора і керувати пулами потоків. Технологія використовує концепцію черг для управління виконанням завдань. Черги можуть бути послідовними (serial) або паралельними (concurrent), що надає гнучкість в організації та синхронізації завдань. Також GCD пропонує простий і зрозумілий API, який спрощує роботу з асинхронними операціями і багатопотоковістю, знижуючи ймовірність помилок, пов'язаних з паралельним виконанням коду.

Загалом, Grand Central Dispatch значно спрощує багатопотокове програмування в iOS і macOS, даючи змогу розробникам зосередитися на логіці застосунку, а не на складнощах керування потоками та асинхронними операціями.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ДОДАТКУ

Перед початком розробки потрібно встановити IDE Xcode 15 з магазину App Store. Створюємо проект та обираємо платформу iOS та запропонований шаблон App.

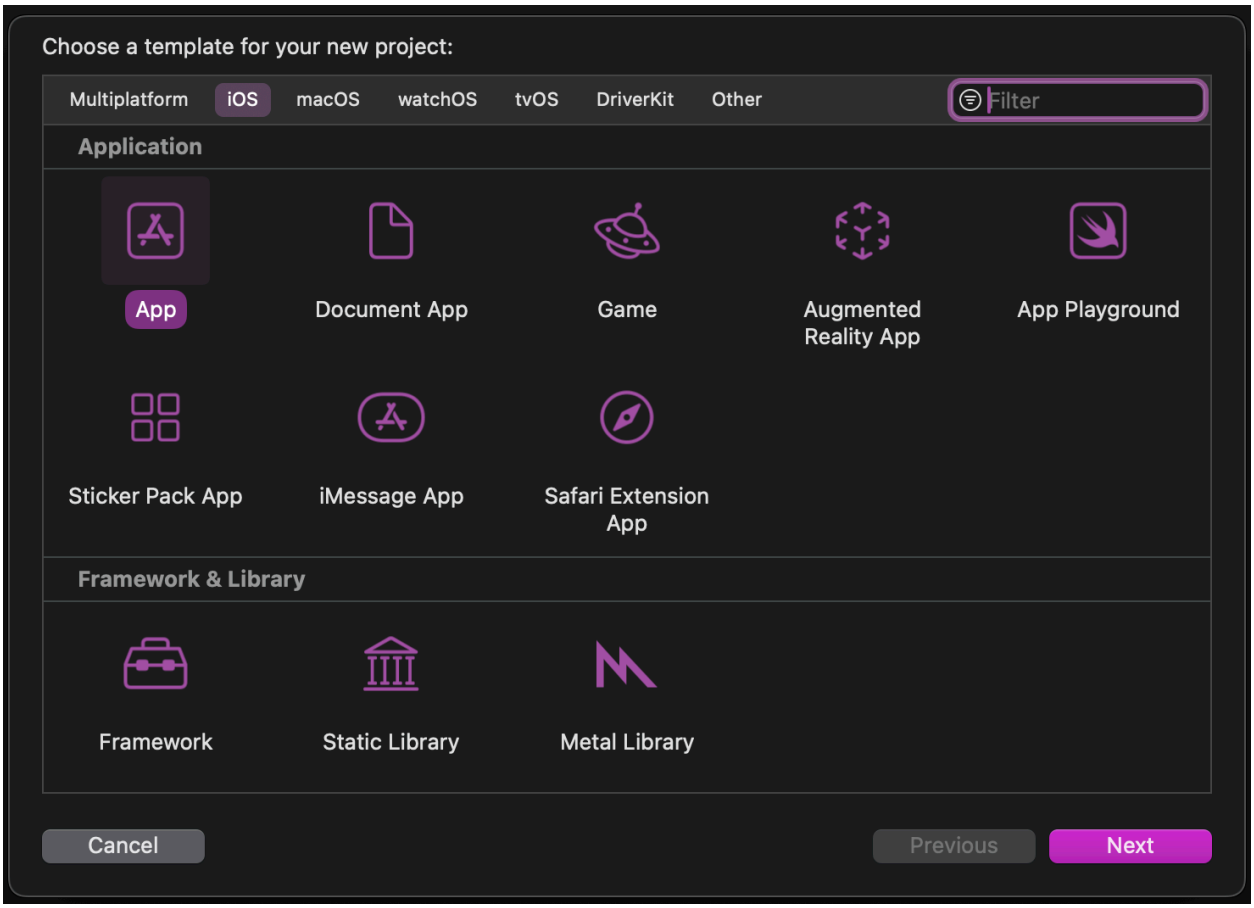

Рисунок 3.1 – Вибір шаблону додатку

Далі потрібно обрати команду розробки та назву проєкту. Цей крок дозволяє встановлювати додаток на iPhone для тестування на реальному пристрої. Симулятор не дає повної картини про швидкодію, оптимізацію та зручність використання.

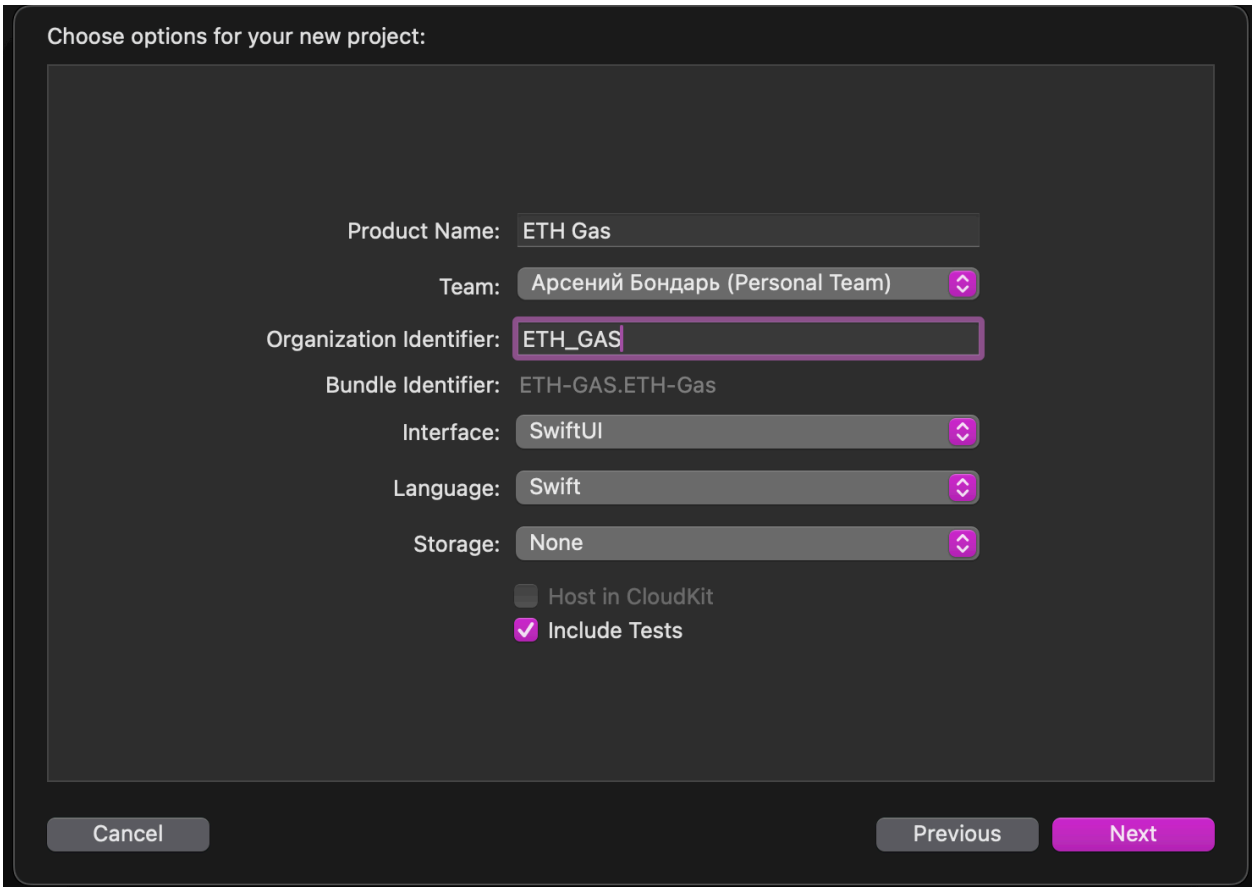

Рисунок 3.2 – Назва додатку та вибір команди

Наступним кроком є імпорт ключових бібліотек, зокрема: Core Data, Foundation, UIKit, User Notification. Для зручності збору діагностичної інформації під час тестування, я додаю до класу ViewController властивість sceneView.showsStatistics, встановлюючи її значення на true.

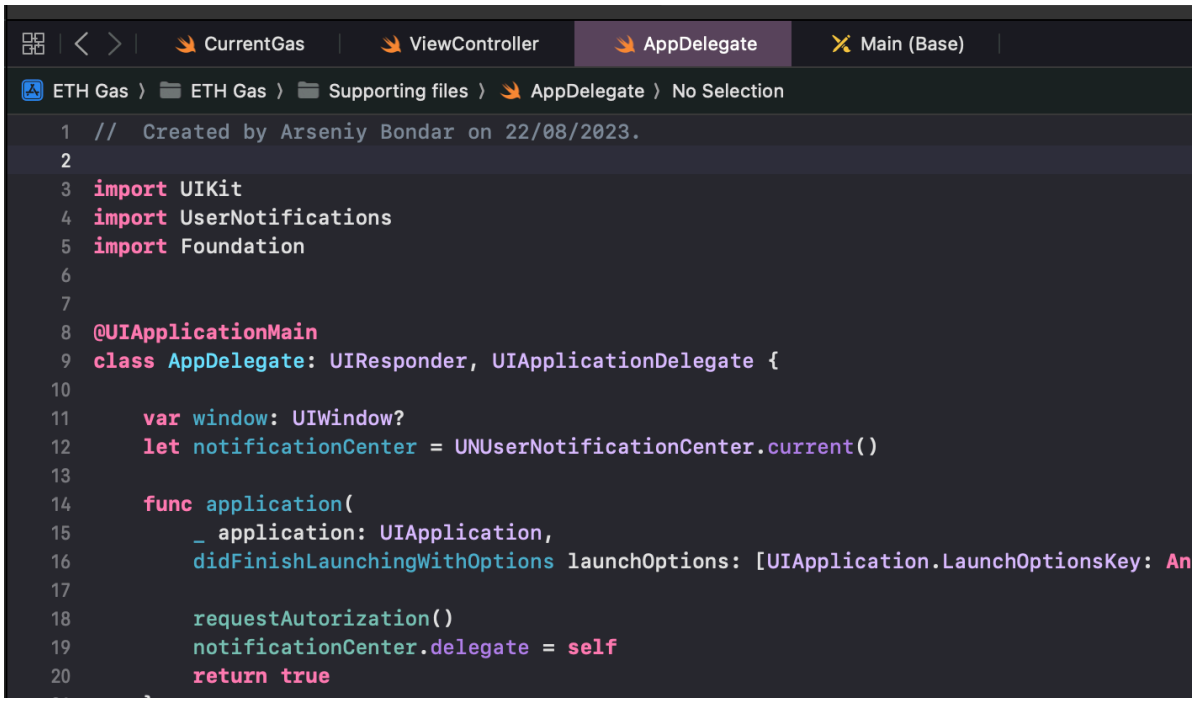

Рисунок 3.3 – Імпорт необхідних фреймворків

За попередньо створеним макетом інтерфейсу у додатку Figma, потрібно написати інтерфейс за допомогою фреймворку SwiftUI.

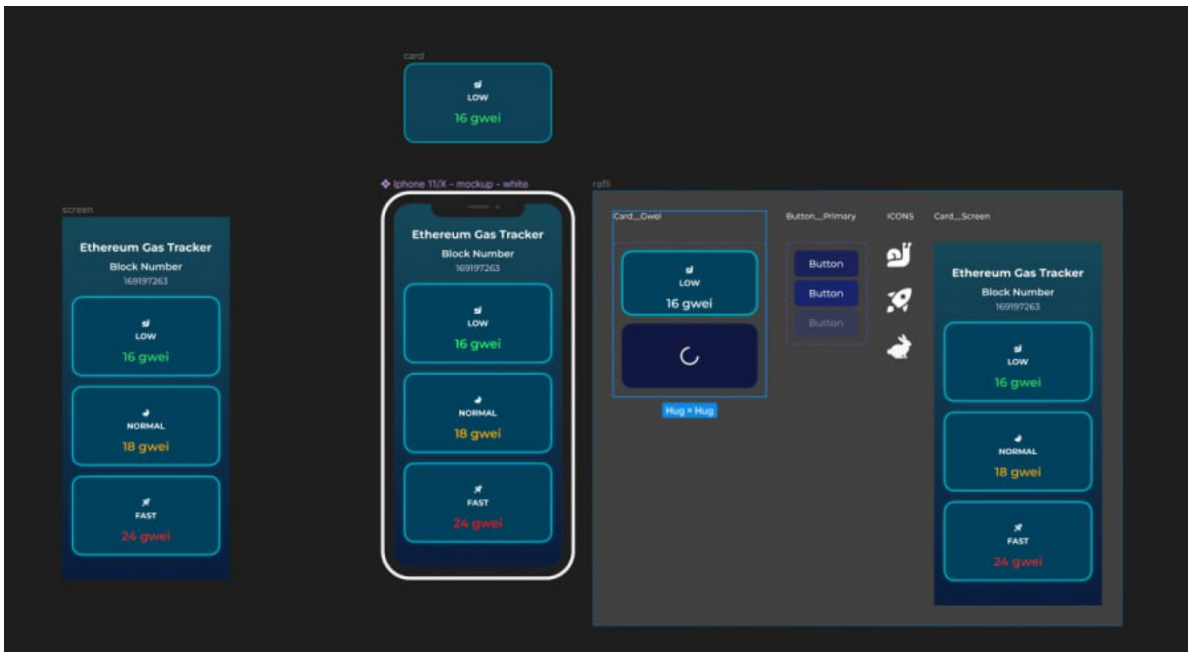

Рисунок 3.4 – Макет інтерфейсу

На наступному рисунку можна побачити код інтерфейсу та його попередній перегляд. При роботі додатку placeholder (три крапки) замінюються на значення змінних safeGas, normalGas та fastGas відповідно.

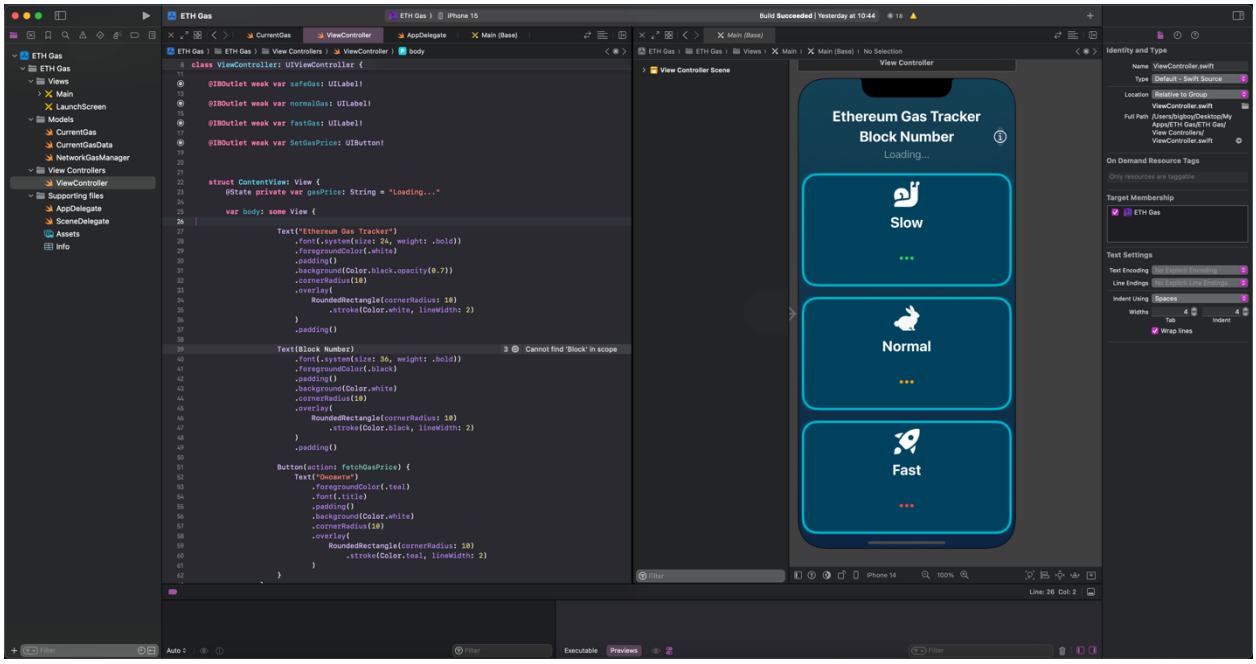

Рисунок 3.5 – Код та попередній перегляд інтерфейсу

Для відстеження цін на транзакції в блокчейні Ethereum, я скористався API Etherscan, який дає можливість отримати різноманітну інформацію про блокчейн Ethereum, включаючи дані про транзакції та ціни на газ. Вони надають детальну документацію та простий у використанні інтерфейс. Але як і більшість публічних API, має певні обмеження та вимоги, які важливо враховувати при його використанні. Ось декілька ключових обмежень:

- Etherscan зазвичай обмежує кількість запитів, які можна виконати за певний проміжок часу. Це робиться для запобігання перевантаження їх серверів та забезпечення стабільної роботи сервісу для всіх

користувачів. Ліміти можуть варіюватися в залежності від того, чи використовуєте ви безкоштовний або платний план.

- Для доступу до Etherscan API зазвичай потрібно зареєструватися та отримати особистий API ключ. Це дозволяє Etherscan ідентифікувати та контролювати використання своїх ресурсів.
- Оскільки Etherscan залежить від стану блокчейну Ethereum, можуть бути затримки у наданні актуальної інформації, особливо під час періодів високого завантаження мережі.
- Etherscan може встановлювати обмеження щодо комерційного використання даних, тому важливо уважно вивчити їхні умови використання перед інтеграцією API у проєкт.

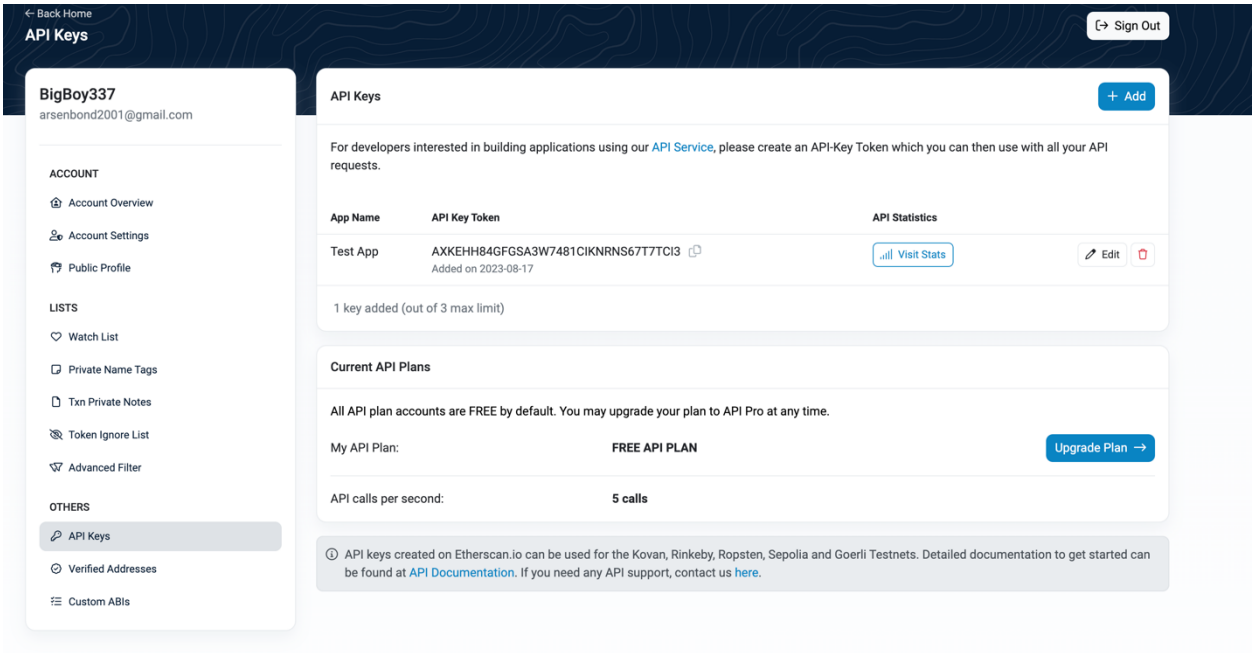

Рисунок 3.6 – Отримання API-key

Наступним кроком потрібно розробити логіку отримання даних з API та їх подальше декодування. Для цього створюю структуру NetworkGasManager.

Функція fetchCurrentGas ініціює мережевий запит до API Etherscan для отримання даних про ціни на газ. completionnHandler - це замикання, яке викликається з результатом запиту (CurrentGas), якщо запит успішний. В середині функції створюється URL запиту, ініціюється сеанс URLSession і виконується запит (dataTask). Після отримання відповіді, дані передаються в parseJSON (функцію для аналізу).

Функція parseJSON приймає дані в необробленому виді від мережевого запиту та спробує декодувати їх у тип CurrentGasData за допомогою JSONDecoder. Якщо декодування успішне, вона ініціалізує та повертає екземпляр CurrentGas, використовуючи отримані дані. У функції parseJSON є блок do-catch, який обробляє помилки, що можуть виникнути під час декодування JSON. Також я додав декілька викликів print(), які виводять у консоль значення fastGasPrice та проблеми з локалізацією. Це все допомагає при тестуванні додатку без взаємодії з користувацьким інтерфейсом.

```swift
struct NetworkGasManager {

    func fetchCurrentGas(forCity status: String, completionnHandler: @escaping (CurrentGas) -> Void) {
        let urlString =
            "https://api.etherscan
            .io/api?module=gastracker&action=gasoracle&apikey=AXKEHH84GFGSA3W7481CIKNRNS67T7TCI3"
        guard let url = URL(string: urlString) else { return }
        let session = URLSession(configuration: .default)
        let task = session.dataTask(with: url) { data, response, error in
            if let data = data {
                let dataString = String(data: data, encoding: .utf8)
                print(dataString!)
                if let currentGas = self.parseJSON(withData: data) {
                    completionnHandler(currentGas)
                }
            }
        }
            task.resume()
        }

        func parseJSON(withData data: Data) -> CurrentGas? {
            let decoder = JSONDecoder()
            do {
                let currentGasData = try decoder.decode(CurrentGasData.self, from: data)

                print(currentGasData.result.fastGasPrice)

                guard let currentGas = CurrentGas(currentGasData: currentGasData) else {
                    return nil
                }
                return currentGas
            } catch let error as NSError{
                print(error.localizedDescription)
            }
            return nil
        }
    }
```

Рисунок 3.7 – Отримання та декодування отриманих даних

Далі створюю дві структури CurrentGasData та Result, які використовуються для декодування даних з JSON. Це здійснюється за допомогою протоколу Codable, який полегшує серіалізацію та десеріалізацію даних, особливо коли мова йде про роботу з мережевими запитами та JSON відповідями.

Структура CurrentGasData має три властивості: status, message та result. status та message є рядками (String), які містять інформацію про статус відповіді API (наприклад, чи був запит успішним) та будь-яке повідомлення, пов'язане з відповіддю. result є типом Result, що містить деталізованішу інформацію.

Структура Result містить інформацію, таку як lastBlock, safeGasPrice, proposeGasPrice, fastGasPrice, suggestBaseFee та gasUsedRatio, всі вони є типом String. Ці поля представляють останній блок в блокчейні, різні рівні цін на газ (безпечний, пропонований, швидкий), пропоновану базову вартість транзакції та співвідношення використаного газу.

CodingKeys – це внутрішнє перерахування, яке використовується для визначення відповідності між ключами JSON та властивостями структури. Це дозволяє правильно декодувати JSON, якщо ключі JSON не відповідають назвам властивостей у Swift коді.

```swift

import Foundation


struct CurrentGasData: Codable {
    let status, message: String
    let result: Result
}

struct Result: Codable {
    let lastBlock, safeGasPrice, proposeGasPrice, fastGasPrice: String
    let suggestBaseFee, gasUsedRatio: String

    enum CodingKeys: String, CodingKey {
        case lastBlock = "LastBlock"
        case safeGasPrice = "SafeGasPrice"
        case proposeGasPrice = "ProposeGasPrice"
        case fastGasPrice = "FastGasPrice"
        case suggestBaseFee, gasUsedRatio
    }
}
```

Рисунок 3.8 – Структура CurrentGasData

Обов’язково потрібно створити структуру CurrentGas, яка використовується для зберігання інформації про поточну ціну газу в блокчейні Ethereum.

Структура CurrentGas оголошена для представлення даних про ціну газу. Вона містить наступні властивості:

- lastBlock: Рядок, що зберігає інформацію про останній блок.

- safeGasPrice: Рядок, що відображає безпечну ціну газу.
- proposeGasPrice: Рядок, що відображає пропоновану ціну газу.
- fastGasPrice: Рядок, що відображає швидку ціну газу.

Ініціалізатор init? - це опціональний ініціалізатор, який приймає параметр currentGasData типу CurrentGasData. Він використовує дані з currentGasData для ініціалізації властивостей CurrentGas та присвоює значення властивостям lastBlock, safeGasPrice, proposeGasPrice, і fastGasPrice на основі відповідних полів у currentGasData.result. Опціональний ініціалізатор (init?) означає, що ініціалізація може не відбутися (наприклад, якщо дані в currentGasData не можна адекватно обробити), і в цьому випадку результатом буде nil.

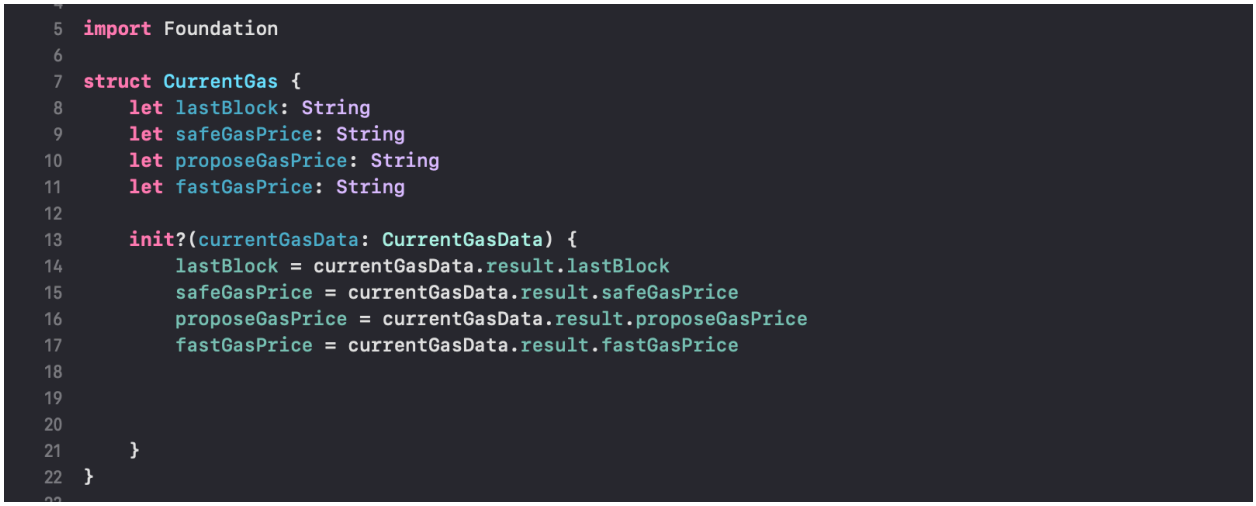

Рисунок 3.9 – Структура CurrentGas

Для реалізації функції пуш-сповіщень про ціну газу застосуємо 2 фреймворки - Core Data та User Notification. На першому етапі потрібно створити новий файл GasPriceUser.xcdatamodeld та за допомогою вбудованого редактору моделей даних редагую сутності, атрибути та відношення.

Функція saveGas(Gas: String) використовується для зберігання значення газу в базі даних Core Data. Спочатку вона отримує AppDelegate та з нього — контекст управління об'єктами (managed object context). Далі створюється новий об'єкт NSManagedObject за допомогою опису сутності (entity) з іменем Gas. Значення газу зберігається в цьому об'єкті за ключем Gas. Якщо виникає помилка під час збереження, вона виводиться в консоль.

Функція fetchGas() -> String? використовується для отримання збереженого значення газу з бази даних Core Data. Аналогічно, вона отримує AppDelegate та managed object context, створює та виконує запит на вибірку (NSFetchRequest) для сутності Gas. Потім намагається отримати масив збережених об'єктів і повертає значення властивості Gas першого об'єкта у масиві. Якщо виникає помилка під час вибірки, вона виводиться в консоль, і функція повертає nil.

```
import CoreData

func saveGas(Gas: String) {
    guard let appDelegate = UIApplication.shared.delegate as? AppDelegate else {
        return
    }
    let managedContext = appDelegate.persistentContainer.viewContext

    let entity = NSEntityDescription.entity(forEntityName: «Gas», in: managedContext)!
    let user = NSManagedObject(entity: entity, insertInto: managedContext)
    user.setValue(name, forKeyPath: Gas)
    do {
        try managedContext.save()
    } catch let error as NSError {
        print(Cant Save Data. \(error), \(error.userInfo)")
    }
}

func fetchGas() -> String? {
    guard let appDelegate = UIApplication.shared.delegate as? AppDelegate else {
        return nil
    }
    let managedContext = appDelegate.persistentContainer.viewContext

    let fetchRequest = NSFetchRequest<NSManagedObject>(entityName: «Gas»)

    do {
        let users = try managedContext.fetch(fetchRequest)
        return users.first?.value(forKey: «Gas») as? String
    } catch let error as NSError {
        print(«Can't extract data. \(error), \(error.userInfo)")
        return nil
```

Рисунок 3.10 – Функції saveGas та fetchGas

Наступним кроком ініціалізуємо networkGasManager для здійснення мережевих запитів. Всередині функції viewDidLoad ініціалізується таймер, який кожні 7 секунд викликає функцію updateCounting, що повторно запитує дані про газ.

Функція getCurrentGasData використовує networkGasManager для отримання актуальних даних про газ оновлює текстові поля у користувацькому інтерфейсі за допомогою updateUIWith.

При натисканні на кнопку відображається вспливаюче вікно (UIAlertController) для введення користувачем ціни газу. Далі, якщо поточна ціна збігається з тою, що ввів користувач - йому надсилається локальне пуш-сповіщення.

```
    func updateCounting(){
        //   toggleActivityIndicator(on: true)
        getCurrentGasData()
    }

    func getCurrentGasData() {
        networkGasManager.fetchCurrentGas(forCity: "1") { currentGas in
            //  self.toggleActivityIndicator(on: false)
            let lastBlockTemp = currentGas.lastBlock
            let safeGasTemp = currentGas.safeGasPrice
            let normalGasTemp = currentGas.proposeGasPrice
            let fastGasTemp = currentGas.fastGasPrice
            self.updateUIWith(lastBlockTemp: lastBlockTemp, safeGasTemp: safeGasTemp, normalGasTemp: normalGasTemp, fastGasTemp: fastGasTemp)
            print(lastBlockTemp)
        }
    }

    func updateUIWith(lastBlockTemp: String, safeGasTemp: String, normalGasTemp: String, fastGasTemp: String) {
        DispatchQueue.main.async {
            self.lastBlock.text = lastBlockTemp
            self.safeGas.text = safeGasTemp + " gwei"
            self.normalGas.text = normalGasTemp + " gwei"
            self.fastGas.text = fastGasTemp + " gwei"
        }
    }

    @IBAction func buttonTapped(_ sender: UIButton) {
        let alertController = UIAlertController(title: "Enter Gas Price", message: nil, preferredStyle: .alert)

        // Добавляем текстовое поле
        alertController.addTextField { textField in
            textField.placeholder = "GWEI"
        }
        // Добавляем кнопку "Ok"
        let okAction = UIAlertAction(title: "Ok", style: .default) { [weak self] action in
                    // Обработка нажатия на кнопку "Ok"
                    if let textField = alertController.textFields?.first, let enteredText = textField.text {
                        self?.test2 = enteredText
                        print("Введенный текст: \(enteredText)")
                    }
                }
```

Рисунок 3.11 – Взаємодія з інтерфейсом додатку

В ході тестування додатку я знайшов критичний баг. А саме: отримані дані з API коректно декодувалися та відображалися у консолі розробника. Але інтерфейс програми мав час оновлення близько 15 секунд, що не є нормальною роботою. Тому довелось застосувати метод DispatchQueue.main.async. У мові Swift він використовується для виконання коду асинхронно на головному потоці (main thread). Цей метод є важливим інструментом для управління взаємодією з користувацьким інтерфейсом (UI) та виконанням коду, що не блокує головний потік.

```
func updateUIWith(lastBlockTemp: String, safeGasTemp: String, normalGasTemp: String, fastGasTemp: String) {
    DispatchQueue.main.async {
        self.lastBlock.text = lastBlockTemp
        self.safeGas.text = safeGasTemp + " gwei"
        self.normalGas.text = normalGasTemp + " gwei"
        self.fastGas.text = fastGasTemp + " gwei"
    }
}
```

Рисунок 3.12 – Застосування методу DispatchQueue.main.async

На наступних рисунках продемонстрована робота додатку після усунення всіх відомих багів.

Рисунок 3.13 – Головний екран додатку

Рисунок 3.14 – Поле для вводу бажаної ціни газу

Рисунок 3.13 – Демонстрація пуш-сповіщення

ВИСНОВКИ

В ході виконання дипломного проєкту було проведено глибоке дослідження ринку мобільних додатків для моніторингу блокчейн-транзакцій. Виявлено, що наявні рішення часто не задовольняють потреби користувачів щодо зручності, швидкодії та інформативності. Таким чином, актуальність розробки спеціалізованого додатку для iOS, який би оптимізував процес моніторингу цін на транзакції в Ethereum, була повністю підтверджена.

Розроблений додаток використовує сучасні технічні можливості та дизайнерські рішення, враховуючи специфіку цільової аудиторії. Програмний продукт забезпечує оперативний доступ до актуальної інформації про комісії в мережі Ethereum, що важливо для користувачів при проведенні транзакцій.

З огляду на стрімкий розвиток криптовалютного сектору та збільшення кількості транзакцій у мережі Ethereum, проект має значний потенціал для масштабування та подальшого розвитку. В майбутньому планується розширення функціональності додатку, введення додаткових інструментів для аналізу ринку, а також інтеграція з іншими блокчейн-платформами.

СПИСОК ЛІТЕРАТУРИ

1. What is web3 [Електронний ресурс] – Режим доступу: https://hbr.org/2022/05/what-is-web3
2. Satoshi Nakamoto [Електронний ресурс] – Режим доступу: https://www.investopedia.com/terms/s/satoshi-nakamoto.asp
3. Cryptocurrency features [Електронний ресурс] – Режим доступу: https://conotoxia.com/cryptocurrencies/what-are-cryptocurrencies/cryptocurrency-features
4. Why crypto dev are so valuable [Електронний ресурс] – Режим доступу: https://mobilunity.com/blog/why-cryptocurrency-developers-are-so-valuable-in-2023/
5. The future of Ethereum [Електронний ресурс] – Режим доступу: https://www.forbes.com/sites/digital-assets/article/the-future-of-ethereum/?sh=7419e89d35a7
6. Proof of work vs proof of stake [Електронний ресурс] – Режим доступу: https://academy.binance.com/en/articles/proof-of-work-vs-proof-of-stake
7. Ethereum 2.0 [Електронний ресурс] – Режим доступу: https://www.gate.io/ru/learn/articles/what-is-ethereum-2-0/102
8. What is ETH gas [Електронний ресурс] – Режим доступу: https://www.bitcoin.com/ru/get-started/what-is-ETH-gas-and-how-do-fees-work-in-ethereum/
9. Gwei [Електронний ресурс] – Режим доступу: https://www.investopedia.com/terms/g/gwei-ethereum/
10. Etherscan Docs [Електронний ресурс] – Режим доступу: https://docs.etherscan.io/

11. GAS app [Електронний ресурс] – Режим доступу: https://apps.apple.com/ua/app/gas-alert-ethereum-gas-tracker/id6446234870?l=ru

12. Ethereum gas pump app [Електронний ресурс] – Режим доступу: https://apps.apple.com/ua/app/ethereum-gas-pump/id1559392846?l=ru

13. ETH GAS FEES app[Електронний ресурс] – Режим доступу: https://apps.apple.com/ua/app/crypto-utility-eth-gas-fees/id1592336055?l=ru

14. widgETH app [Електронний ресурс] – Режим доступу: https://apps.apple.com/ua/app/widgeth/id6449916833?l=ru

15. Ethereum Gas app [Електронний ресурс] – Режим доступу: https://apps.apple.com/ua/app/%D0%B3%D0%B0%D0%B7%D0%BE%D0%B2%D1%8B%D0%B9-%D1%82%D1%80%D0%B5%D0%BA%D0%B5%D1%80-ethereum/id6449078810?l=ru

16. Top companies in the world [Електронний ресурс] – Режим доступу: https://www.statista.com/statistics/263264/top-companies-in-the-world-by-market-capitalization/

17. WWDC 2023 [Електронний ресурс] – Режим доступу: https://thepage.ua/ua/news/prezentaciya-apple-wwdc-2023-sho-predstavleno

18. SwiftBook [Електронний ресурс] – Режим доступу: https://docs.swift.org/swift-book/#

19. Xcode 15 Apple[Електронний ресурс] – Режим доступу: https://developer.apple.com/xcode/#:~:text=Xcode%2015%20and%20Xcode%20Cloud,to%20your%20testers%20and%20users

20. Xcode 15 vs other IDE [Електронний ресурс] – Режим доступу: https://developer.apple.com/xcode/

21. Swift 5 [Електронний ресурс] – Режим доступу: https://www.swift.org/

22. GitHub Octoverse 2023 [Електронний ресурс] – Режим доступу: https://octoverse.github.com/static/octoverse-report-2023.pdf

23. StackOverflow Survey 2021 [Електронний ресурс] – Режим доступу: https://insights.stackoverflow.com/survey/2021#technology

24. Most popular mobile OS [Електронний ресурс] – Режим доступу: https://gs.statcounter.com/os-market-share/mobile/worldwide

25. iOS architecture [Електронний ресурс] – Режим доступу: https://docs.elementscompiler.com/Platforms/Cocoa/Frameworks/iOSSDKFrameworks/

26. iOS architecture [Електронний ресурс] – Режим доступу: https://www.foxrichcode.com/page/52/ios-operating-system-architecture

27. Cocoa Touch [Електронний ресурс] – Режим доступу: https://www.wikidata.uk-ua.nina.az/Cocoa_Touch.html

28. Core Data [Електронний ресурс] – Режим доступу: https://habr.com/ru/articles/493262/

29. UIKit [Електронний ресурс] – Режим доступу: https://developer.apple.com/documentation/uikit

30. Foundation [Електронний ресурс] – Режим доступу: https://code.tutsplus.com/exploring-the-foundation-framework--mobile-13976t

31. UserNotification [Електронний ресурс] – Режим доступу: https://developer.apple.com/documentation/usernotifications

ДОДАТОК

Файл CurrentGas.swift

```
import Foundation

struct CurrentGas {
    let lastBlock: String
    let safeGasPrice: String
    let proposeGasPrice: String
    let fastGasPrice: String

    init?(currentGasData: CurrentGasData) {
        lastBlock = currentGasData.result.lastBlock
        safeGasPrice = currentGasData.result.safeGasPrice
        proposeGasPrice = currentGasData.result.proposeGasPrice
        fastGasPrice = currentGasData.result.fastGasPrice
    }
}
```

Файл CurrentGasData.swift

```
import Foundation

struct CurrentGasData: Codable {
    let status, message: String
    let result: Result
}

struct Result: Codable {
    let lastBlock, safeGasPrice, proposeGasPrice, fastGasPrice: String
    let suggestBaseFee, gasUsedRatio: String

    enum CodingKeys: String, CodingKey {
        case lastBlock = "LastBlock"
        case safeGasPrice = "SafeGasPrice"
        case proposeGasPrice = "ProposeGasPrice"
        case fastGasPrice = "FastGasPrice"
```

```
        case suggestBaseFee, gasUsedRatio
    }
}
```

Файл NetworkGasManager.swift

```
import Foundation

struct NetworkGasManager {

    func fetchCurrentGas(forCity status: String, completionnHandler: @escaping
(CurrentGas) -> Void) {
        let urlString =
"https://api.etherscan.io/api?module=gastracker&action=gasoracle&apikey=AXKE
HH84GFGSA3W7481CIKNRNS67T7TCI3"
        guard let url = URL(string: urlString) else { return }
        let session = URLSession(configuration: .default)
        let task = session.dataTask(with: url) { data, response, error in
            if let data = data {
                let dataString = String(data: data, encoding: .utf8)
                print(dataString!)
                if let currentGas = self.parseJSON(withData: data) {
                    completionnHandler(currentGas)
                }
            }
        }
            task.resume()
        }

        func parseJSON(withData data: Data) -> CurrentGas? {
            let decoder = JSONDecoder()
            do {
                let currentGasData = try decoder.decode(CurrentGasData.self, from:
data)

                print(currentGasData.result.fastGasPrice)

                guard let currentGas = CurrentGas(currentGasData: currentGasData) else
{
```

```
                return nil
            }
            return currentGas
        } catch let error as NSError{
            print(error.localizedDescription)
        }
        return nil
    }
}
```

Файл AppDelegate.swift

```
import UIKit
import UserNotifications
import Foundation


@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

    var window: UIWindow?
    let notificationCenter = UNUserNotificationCenter.current()

    func application(
        _ application: UIApplication,
        didFinishLaunchingWithOptions launchOptions:
[UIApplication.LaunchOptionsKey: Any]?) -> Bool {

        requestAutorization()
        notificationCenter.delegate = self
        return true
    }



    func applicationDidBecomeActive(_ application: UIApplication) {
        UIApplication.shared.applicationIconBadgeNumber = 0
    }
```

```
    func requestAutorization() {
        notificationCenter.requestAuthorization(options: [.alert, .sound, .badge]) {
(granted, error) in
            print("Permission granted: \(granted)")

            guard granted else { return }
            self.getNotificationSettings()
        }
    }

    func getNotificationSettings() {
        notificationCenter.getNotificationSettings { (settings) in
            print("Notification settings: \(settings)")
        }
    }

    func scheduleNotification(notifaicationType: String) {

        let content = UNMutableNotificationContent()
        let userAction = "User Action"

        content.title = notifaicationType
        content.body = "This is example how to create " + notifaicationType
        content.sound = UNNotificationSound.default
        content.badge = 1
        content.categoryIdentifier = userAction

        let trigger = UNTimeIntervalNotificationTrigger(timeInterval: 5, repeats:
false)

        let identifire = "Local Notification"
        let request = UNNotificationRequest(identifier: identifire,
                                            content: content,
                                            trigger: trigger)

        notificationCenter.add(request) { (error) in
            if let error = error {
                print("Error \(error.localizedDescription)")
            }
        }
```

```
        let snoozeAction = UNNotificationAction(identifier: "Snooze", title:
"Snooze", options: [])
        let deleteAction = UNNotificationAction(identifier: "Delete", title: "Delete",
options: [.destructive])
        let category = UNNotificationCategory(
            identifier: userAction,
            actions: [snoozeAction, deleteAction],
            intentIdentifiers: [],
            options: [])

        notificationCenter.setNotificationCategories([category])
    }

}

extension AppDelegate: UNUserNotificationCenterDelegate {

    func userNotificationCenter(
        _ center: UNUserNotificationCenter,
        willPresent notification: UNNotification,
        withCompletionHandler completionHandler: @escaping
(UNNotificationPresentationOptions) -> Void) {

        completionHandler([.alert, .sound])
    }

    func userNotificationCenter(
        _ center: UNUserNotificationCenter,
        didReceive response: UNNotificationResponse,
        withCompletionHandler completionHandler: @escaping () -> Void) {

        if response.notification.request.identifier == "Ethereum Gas Price" {
            print("... GWEI")
        }

        switch response.actionIdentifier {
        case UNNotificationDismissActionIdentifier:
            print("Dismiss Action")
        case UNNotificationDefaultActionIdentifier:
```

```
            print("Default")
        case "Snooze":
            print("Snooze")
            scheduleNotification(notifaicationType: "Reminder")
        case "Delete":
            print("Delete")
        default:
            print("Unknown action")
        }

        completionHandler()
    }
}
```

Файл ViewController.swift

```
import UIKit
import SwiftUI

class ViewController: UIViewController {

    @IBOutlet weak var lastBlock: UILabel!

    @IBOutlet weak var safeGas: UILabel!

    @IBOutlet weak var normalGas: UILabel!

    @IBOutlet weak var fastGas: UILabel!

    @IBOutlet weak var SetGasPrice: UIButton!

    import SwiftUI

    struct ContentView: View {
        @State private var gasPrice: String = "Loading..."

        var body: some View {
            ZStack {
```

```
Color.teal
   .ignoresSafeArea()

VStack {
   Text("ETH GAS PRICE")
      .font(.system(size: 24, weight: .bold))
      .foregroundColor(.white)
      .padding()
      .background(Color.black.opacity(0.7))
      .cornerRadius(10)
      .overlay(
         RoundedRectangle(cornerRadius: 10)
            .stroke(Color.white, lineWidth: 2)
      )
      .padding()

   Text(gasPrice)
      .font(.system(size: 36, weight: .bold))
      .foregroundColor(.black)
      .padding()
      .background(Color.white)
      .cornerRadius(10)
      .overlay(
         RoundedRectangle(cornerRadius: 10)
            .stroke(Color.black, lineWidth: 2)
      )
      .padding()

   Button(action: fetchGasPrice) {
      Text("GWEI")
         .foregroundColor(.teal)
         .font(.title)
         .padding()
         .background(Color.white)
         .cornerRadius(10)
         .overlay(
            RoundedRectangle(cornerRadius: 10)
               .stroke(Color.teal, lineWidth: 2)
         )
   }
```

```
            }
        }
    }

    func fetchGasPrice() {

    }
}

struct ContentView_Previews: PreviewProvider {
    static var previews: some View {
        ContentView()
    }
}


var test2 : String?
//  @IBOutlet weak var activityIndicator: UIActivityIndicatorView!

// @IBOutlet weak var refreshGasData: UIButton!

//   func toggleActivityIndicator(on: Bool)
//   {
//       DispatchQueue.main.async {
// self.refreshGasData.isHidden = on
//           if on {
//               self.activityIndicator.startAnimating()
//           } else {
//               self.activityIndicator.stopAnimating()
//           }
//       }
//   }


let networkGasManager = NetworkGasManager()


var timer: Timer?

override func viewDidLoad() {
```

```
        super.viewDidLoad()
        getCurrentGasData()
        timer = Timer.scheduledTimer(withTimeInterval: 7, repeats: true, block: { _
in
            self.updateCounting()
        } )
    }

    func updateCounting(){
        //   toggleActivityIndicator(on: true)
        getCurrentGasData()
    }

    func getCurrentGasData() {
        networkGasManager.fetchCurrentGas(forCity: "1") { currentGas in
            //  self.toggleActivityIndicator(on: false)
            let lastBlockTemp = currentGas.lastBlock
            let safeGasTemp = currentGas.safeGasPrice
            let normalGasTemp = currentGas.proposeGasPrice
            let fastGasTemp = currentGas.fastGasPrice
            self.updateUIWith(lastBlockTemp: lastBlockTemp, safeGasTemp:
safeGasTemp, normalGasTemp: normalGasTemp, fastGasTemp: fastGasTemp)
            print(lastBlockTemp)
        }
    }

    func updateUIWith(lastBlockTemp: String, safeGasTemp: String,
normalGasTemp: String, fastGasTemp: String) {
        DispatchQueue.main.async {
            self.lastBlock.text = lastBlockTemp
            self.safeGas.text = safeGasTemp + " gwei"
            self.normalGas.text = normalGasTemp + " gwei"
            self.fastGas.text = fastGasTemp + " gwei"
        }
    }

    @IBAction func buttonTapped(_ sender: UIButton) {
        let alertController = UIAlertController(title: "Enter Gas Price", message: nil,
preferredStyle: .alert)
```

```
        alertController.addTextField { textField in
            textField.placeholder = "GWEI"
        }
        let okAction = UIAlertAction(title: "Ок", style: .default) { [weak self] action
in
                if let textField = alertController.textFields?.first, let enteredText =
textField.text {
                    self?.test2 = enteredText
                    print("Введенный текст: \(enteredText)")
                }
            }

        alertController.addAction(okAction)

        let cancelAction = UIAlertAction(title: "Cancel", style: .cancel, handler: nil)
        alertController.addAction(cancelAction)

        present(alertController, animated: true, completion: nil)
        let content = UNMutableNotificationContent()
        content.title = "Ethereum Gas Price"
        content.body = "27 GWEI"

        content.sound = UNNotificationSound.default

        let trigger = UNTimeIntervalNotificationTrigger(timeInterval: 10, repeats:
false)

        let request = UNNotificationRequest(identifier: "myNotification", content:
content, trigger: trigger)

        UNUserNotificationCenter.current().add(request, withCompletionHandler: {
error in
            if let error = error {
                print("Error: \(error.localizedDescription)")
            } else {
                print("Push notification complete.")
            }
        })
       }
}
```