БАКАЛАВРСЬКА РОБОТА

БР. ІП - 51.00.00.000 ПЗ

Група ІП-21-3

Равлюк Владислав

2025

Івано-Франківський національний технічний університет нафти і газу

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Равлюк Владислав Валентинович

(прізвище, ім'я, по батькові)

УДК 004

(індекс)

БАКАЛАВРСЬКА РОБОТА

Інформаційна система відслідковування інформації про транспортні засоби

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121 - Інженерія програмного забезпечення

(шифр і назва спеціальності)

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

Здобувач освітнього рівня Равлюк В.В.

(підпис, ініціали та прізвище здобувача)

Науковий керівник Крихівський Михайло Васильович, к.т.н., доцент

(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту
Завідувач кафедри

доц. Бандура В.В.

(посада) (підпис) (дата) (ініціали та прізвище)

Івано-Франківськ – 2025

Івано-Франківський національний технічний університет нафти і газу

Інститут, факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Ступінь вищої освіти бакалавр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою **ІПЗ**
доц. В.В. Бандура
“ ____ ” _____2025 р.

ЗАВДАННЯ

НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТОВІ

Равлюку Владиславу Валентиновичу

(прізвище, ім’я, по-батькові)

1. Тема проекту (роботи) “ Інформаційна система відслідковування інформації про транспортні засоби ”

керівник проекту (роботи) Крихівський М.В., к.т.н., доцент

затверджені наказом закладу вищої освіти від “ 28 ” квітня 2025 р. № 264/7

2. Строк подання студентом проекту (роботи) 10 червня 2025 р.

3. Вихідні дані до проекту (роботи) Результати і матеріали отримані під час проходження переддипломної практики

4. Зміст розрахунково - пояснювальної записки (перелік питань, які потрібно розробити)

1. Аналіз предметної області застосування технології блокчейну для розробки системи
2. Аналіз вимог та опис інструментів розробки системи відслідковування інформації про ТЗ
3. Моделювання процесів розробки інформаційної системи за допомогою UML діаграм
4. Програмна реалізація системи відслідковування інформації про транспортні засоби
5. Розробка інтерфейсу користувача інформаційної системи

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

1. Блок блокчейну (рис. 1.1)
2. Спрощений блокчейн (рис. 1.2)
3. Архітектура Ethereum (рис. 1.3)
4. Діаграма архітектури системи для смарт-контракту Ethereum (рис. 1.4)
5. Приклад опису ТЗ на ресурсі AUTO.RIA (рис. 1.5)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання **28 квітня 2025 р.**

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту	Примітка
1	Аналіз предметної області застосування технології блокчейну для розробки системи	04.05.2025	виконано
2	Аналіз вимог та опис інструментів розробки системи відслідковування інформації про ТЗ	15.05.2025	виконано
3	Моделювання процесів розробки інформаційної системи за допомогою UML діаграм	21.05.2025	виконано
4	Програмна реалізація системи відслідковування інформації про транспортні засоби	28.05.2025	виконано
5	Розробка інтерфейсу користувача інформаційної системи	03.06.2025	виконано
6	Оформлення пояснювальної записки дипломної роботи завідувачем кафедри	10.06.2025	виконано

Студент – дипломник _____
(підпис)

Керівник роботи _____
(підпис)

АНОТАЦІЯ

Бакалаврська робота містить 77 сторінок, 46 рисунків, список використаних джерел із 34 найменуваннями.

Мета роботи - розробити прототип децентралізованої інформаційної системи для відслідковування інформації про транспортні засоби на основі блокчейн-технологій, що забезпечує безпечне збереження, актуалізацію та перевірку даних.

Об'єкт дослідження - інформаційні системи для зберігання, обробки та обміну даними про транспортні засоби.

Предмет дослідження - методи, моделі та технології реалізації блокчейн-орієнтованої системи для фіксації і перевірки історії змін інформації про транспортні засоби.

В першому розділі визначено проблематику централізованих платформ обліку ТЗ та обґрунтовано доцільність використання блокчейн.

В другому розділі встановлено функціональні вимоги до системи та обрано відповідні інструменти розробки

В третьому розділі побудовано діаграми варіантів використання для різних ролей користувачів і сформовано логічну архітектуру системи

В четвертому розділі реалізовано розумний контракт для фіксації даних про транспортні засоби та модулі взаємодії з Firebase.

В п'ятому розділі розроблено інтерфейси для кінцевих користувачів, адміністратора та організацій, що забезпечують доступ до системних функцій

Висновок: в рамках розробленої інформаційної системи реалізовано зв'язок смарт-контрактів Ethereum з фронтендом через Web3.js із використанням Firebase для управління користувачами.

КЛЮЧОВІ СЛОВА: ІНФОРМАЦІЙНА СИСТЕМА, БЛОКЧЕЙН, ETHEREUM, РОЗУМНИЙ КОНТРАКТ, METAMASK, FIREBASE, ОБЛІК, ДЕЦЕНТРАЛІЗАЦІЯ, ДОСТОВІРНІСТЬ ДАНИХ

ANNOTATION

The bachelor's thesis contains 77 pages, 46 figures, a list of used sources with 34 names.

The purpose of the work is to develop a prototype of a decentralized information system for tracking information about vehicles based on blockchain technologies that ensure secure storage, updating and verification of data.

The object of the study is information systems for storing, processing and exchanging data about vehicles.

The subject of the study is methods, models and technologies for implementing a blockchain-oriented system for recording and verifying the history of changes in information about vehicles.

The first section identifies the issues of a centralized vehicle accounting platform and justifies the feasibility of using blockchain.

The second section establishes functional requirements for the system and selects appropriate development tools

The third section builds use case diagrams for various user roles and forms a logical architectural system

The fourth section implements a smart contract for recording data about vehicles and modules for interacting with Firebase.

The fifth section develops interfaces for end users, administrators and organizations that provide access to system functions.

Conclusion: within the framework of the developed information system, the connection of Ethereum smart contracts with the frontend via Web3.js using Firebase for user management is implemented.

KEYWORDS: INFORMATION SYSTEM, BLOCKCHAIN, ETHEREUM, SMART CONTRACT, METAMASK, FIREBASE, ACCOUNTING, DECENTRALIZATION, DATA RELIABILITY

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

VIN – Vehicle Identification Number – Ідентифікаційний номер транспортного засобу

DMV – Department of Motor Vehicles – Департамент реєстрації транспортних засобів

UID – Unique Identifier – Унікальний ідентифікатор

API – Application Programming Interface – Програмний інтерфейс застосунків

ABI – Application Binary Interface – Двійковий інтерфейс застосунків

DB – Database – База даних

Auth – Authentication – Аутентифікація

CRM – Customer Relationship Management – Управління взаємовідносинами з клієнтами

Frontend – Front-end – Зовнішній інтерфейс (клієнтська частина)

Backend – Back-end – Внутрішній інтерфейс (серверна частина)

ВСТУП

Інформаційні технології стрімко проникають у всі сфери суспільного життя, змінюючи способи взаємодії людей, бізнесу та держави. Одним із ключових завдань цифровізації є створення ефективних механізмів обміну, зберігання та перевірки даних, особливо в тих галузях, де точність, достовірність та доступність інформації мають критичне значення. Однією з таких галузей є облік та контроль інформації про транспортні засоби (ТЗ). Надійність даних про ТЗ — пробіг, власників, технічний стан, участь у ДТП — є надзвичайно важливою для потенційних покупців, страхових компаній, фінансових установ, державних реєстраторів та сервісних організацій.

В Україні основні сервіси, що надають інформацію про транспортні засоби, такі як Auto.ria та RST.ua, є централізованими комерційними платформами, які часто не гарантують повну достовірність даних. У багатьох випадках відсутні механізми перевірки автентичності інформації, що призводить до зростання шахрайства, приховування дефектів або спотворення реальної історії автомобіля. Така ситуація потребує впровадження нових підходів до зберігання та перевірки інформації.

У цьому контексті особливої уваги заслуговує використання технології блокчейн — інноваційного методу збереження даних у вигляді розподіленого реєстру, який гарантує незмінність, прозорість та доступність записів. Завдяки своїй децентралізованій природі, блокчейн дозволяє зберігати записи про події або дії (наприклад, зміну власника, техогляд, страхування, ремонт тощо) у незмінному вигляді з відкритим доступом для перевірки.

Розробка інформаційної системи на основі блокчейн-технології, яка дозволить фіксувати, перевіряти й відслідковувати інформацію про транспортні засоби, є актуальним і перспективним напрямом. Така система дозволить кожній уповноваженій організації або користувачу записувати верифіковану інформацію до блокчейн-реєстру, а іншим — перевіряти її

достовірність без потреби звертатися до посередників або довіряти сумнівним джерелам.

Актуальність роботи

Проблематика прозорості даних у сфері транспортних засобів особливо актуальна в умовах поширеного вторинного ринку, де значна частина інформації про попередні ремонти, ДТП, зміну власників або пробіг часто є недоступною, неповною або сфальсифікованою. Відсутність єдиного достовірного джерела ускладнює прийняття рішень для покупців, страхових компаній, банків та органів державної влади. Технологія блокчейну як інструмент захищеного розподіленого реєстру відкриває нові можливості для цифровізації обліку транспортних засобів, унеможливлюючи підміну даних, забезпечуючи відкритість і простежуваність усіх транзакцій. Це створює основу для розробки інформаційної системи нового покоління з високим рівнем довіри та захисту, що повністю відповідає тенденціям розвитку "цифрового суспільства".

Мета роботи - розробити прототип децентралізованої інформаційної системи для відслідковування інформації про транспортні засоби на основі блокчейн-технологій, що забезпечує безпечне збереження, актуалізацію та перевірку даних.

Завдання дослідження

1. Провести аналіз предметної області та чинних платформ для обміну інформацією про транспортні засоби.

2. Обґрунтувати доцільність використання блокчейн-технології у відповідному контексті.

3. Розробити архітектуру інформаційної системи із урахуванням технологій Ethereum, Solidity, Web3.js та Firebase.

4. Спроєктувати моделі взаємодії користувачів із системою на основі діаграм варіантів використання.

5. Імплементувати розумний контракт для зберігання даних про транспортні засоби.

6. Створити веб-інтерфейс для взаємодії з блокчейном і базою даних.

Об’єкт дослідження - інформаційні системи для зберігання, обробки та обміну даними про транспортні засоби.

Предмет дослідження - методи, моделі та технології реалізації блокчейн-орієнтованої системи для фіксації і перевірки історії змін інформації про транспортні засоби.

Методи дослідження:

- Метод системного аналізу для вивчення предметної області.

- Моделювання варіантів використання (UML) для проектування взаємодії компонентів.

- Метод об'єктно-орієнтованого програмування.

- Технології блокчейн-розробки (Solidity, Ethereum smart contracts).

- Методи веб-розробки (JavaScript, Web3.js, Firebase, Node.js).

- Експериментальний метод для перевірки функціональності системи.

Наукова новизна

У дипломній роботі запропоновано концепцію поєднання блокчейн-реєстру з централізованою базою даних для досягнення балансу між незмінністю критичних даних та гнучкістю веб-сервісу. В рамках вузькоспеціалізованої транспортної інформаційної системи реалізовано зв’язок смарт-контрактів Ethereum з фронтендом через Web3.js із використанням Firebase для управління користувачами.

Практичне застосування

Розроблений прототип може бути використаний як основа для створення національної системи цифрового обліку транспортних засобів, інтеграції з державними реєстрами, комерційними автоплатформами або страховими сервісами. Запропоновані рішення також можуть бути адаптовані

для подібних об'єктів обліку — нерухомості, обладнання, логістичних одиниць тощо.

Бакалаврська робота містить 77 сторінок, 46 рисунків, 5 розділів список використаних джерел із 34 найменуваннями.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ЗАСТОСУВАННЯ ТЕХНОЛОГІЇ БЛОКЧЕЙНУ ДЛЯ РОЗРОБКИ ІНФОРМАЦІЙНОЇ СИСТЕМИ ВІДСЛІДКОВУВАННЯ ТРАНСПОРТНИХ ЗАСОБІВ

1.1. Функціональність інформаційної системи відслідковування транспортних засобів на основі блокчейну

Основною метою розробки інформаційної системи відслідковування транспортних засобів на основі блокчейну є створення прозорої та надійної платформи, що забезпечуватиме споживачів повною та достовірною інформацією щодо історії експлуатації автомобіля. Ця система призначена для підвищення довіри під час купівлі-продажу транспортних засіобів.

Система функціонуватиме за принципом створення незмінного, хронологічно впорядкованого ланцюга подій для кожного транспортного засобу, починаючи з моменту його первинного продажу. Цей блокчейн міститиме записи про ключові етапи життєвого циклу автомобіля, включаючи, але не обмежуючись, дані про страхування, виконані ремонтні роботи та операції з перепродажу. Таким чином, забезпечується цілісність та достовірність інформації, що зберігається.

Реалізація цього проєкту передбачає поділ на три ключові компоненти, кожен з яких виконує специфічні функції:

1. Адміністративний модуль.

Цей компонент відповідає за ініціалізацію блокчейн-мережі та управління доступом. Адміністрація системи здійснює генерацію блокчейну та надає необхідні автентифікаційні дані різним організаціям-учасникам, що дозволяє їм долучатися до мережі та створювати власні блоки.

2. Модуль взаємодії з організаціями.

Цей компонент призначений для організацій (наприклад, страхових компаній, сервісних центрів, дилерів), які є постачальниками даних. Вони

використовують цей модуль для формування та додавання нових блоків до блокчейну, що містять актуальну інформацію про транспортні засоби.

2. Клієнтський модуль.

Цей компонент розроблений для кінцевих користувачів — потенційних покупців транспортних засобів. За допомогою цього модуля клієнти отримують доступ до детальної історії експлуатації автомобіля, що зберігається в блокчейні, забезпечуючи інформоване прийняття рішення щодо придбання.

1.2. Актуальність розробки системи відслідковування інформації про транспортні засоби

На сучасному етапі розвитку автомобільного ринку України спостерігається значна частка угод з транспортними засобами з пробігом. За даними статистики, значна частина українських домогосподарств володіє автомобілем, причому значний відсоток цих транспортних засобів є вживаними.

Придбання вживаного автомобіля в Україні часто супроводжується використанням онлайн-платформ, таких як Auto.ria, RST.ua та OLX Авто, а також веб-сайтів офіційних дилерів. Однак, доступ до повної та достовірної інформації про історію транспортного засобу залишається обмеженим. Існуючі сервіси для перевірки історії автомобілів, як-от ті, що надають витяги з державних реєстрів, можуть не містити всієї необхідної інформації, що створює ризики для покупців.

1.2.1. Проблематика та цілі проєкту

При купівлі вживаного автомобіля споживачі в Україні прагнуть знайти оптимальне співвідношення ціни та якості. Ключовими критеріями вибору є технічний стан транспортного засобу та його ринкова вартість.

Покупці зацікавлені в отриманні вичерпної інформації про історію експлуатації автомобіля, включаючи:

- Умови використання: тип експлуатації (наприклад, таксі, приватний транспорт), регіони використання.

- Регулярність обслуговування: частота та обсяг технічного обслуговування, записи про ремонтні роботи.

Ця інформація є критично важливою для прогнозування потенційних витрат на утримання транспортного засобу в майбутньому.

Метою даного проєкту є створення прозорої та надійної інформаційної системи з використанням технології блокчейн. Ця система дозволить споживачам впевнено купувати транспортні засоби з пробігом, мінімізуючи ризики шахрайства та недобросовісних операцій. Впровадження такої системи сприятиме зниженню рівня корупції та нелегальної діяльності на автомобільному ринку України, підвищуючи його прозорість та довіру між учасниками.

1.3. Архітектура та технологічна основа інформаційної системи відслідковування транспортних засобів

Представлений веб-додаток призначений для забезпечення доступу до вичерпної інформації про транспортні засоби. Його функціональність базується на технології блокчейну, зокрема, на використанні розумних контрактів.

1.3.1. Структура та принцип функціонування блокчейну

Блокчейн за своєю суттю є розподіленою базою даних, що використовує криптографічні методи для забезпечення цілісності та незмінності даних. Інформація в блокчейні організована у вигляді блоків. Кожен блок містить:

- Дані: безпосередня інформація, що записується в блок.

- Хеш даних: криптографічний відбиток (дайджест) даних, що гарантує їхню цілісність.

- Хеш попереднього блоку: посилання на хеш попереднього блоку в ланцюгу, що забезпечує хронологічну послідовність та незмінність даних.

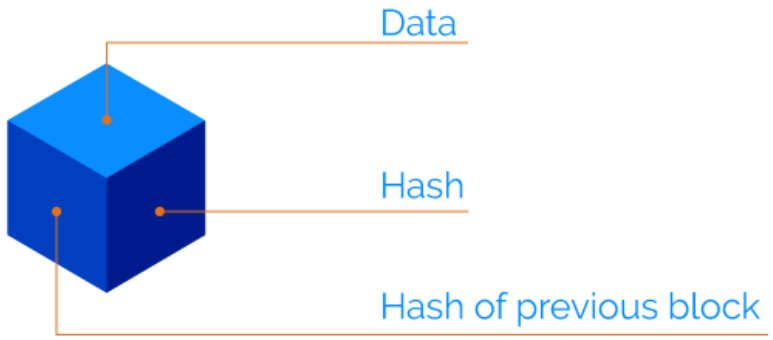

Рисунок 1.1 - Блок блокчейну

Таким чином, блоки утворюють криптографічно зв'язаний список, де кожен наступний блок містить хеш попереднього. Це створює механізм, за якого будь-яка спроба змінити дані в одному блоці призводить до порушення цілісності всього ланцюга, оскільки хеші наступних блоків стануть недійсними. Дані в блокчейні є глобально доступними, при цьому кожному учаснику мережі надається копія повного ланцюга.

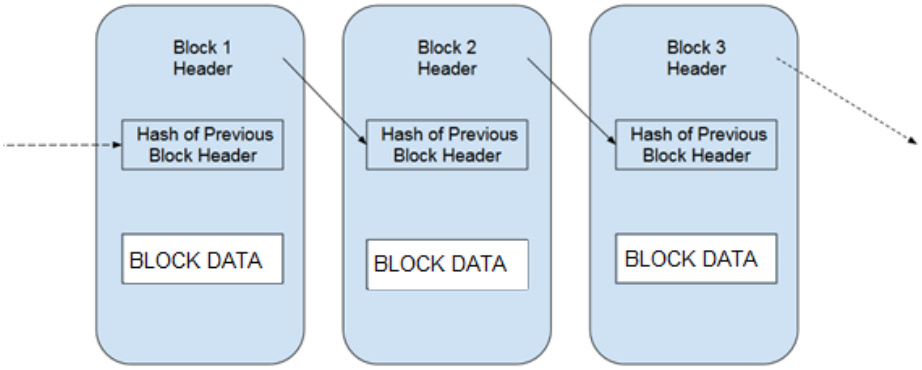

Рисунок 1.2 - Спрощений блокчейн

Взаємозв'язок між блоками забезпечується шляхом включення хешу попереднього блоку як вхідного параметра для наступного блоку, що формує нерозривний ланцюг подій та забезпечує взаємопов'язаність усієї інформації.

1.3.2. Використання платформи Ethereum

Для реалізації даного проєкту обрано платформу Ethereum, яка є провідним блокчейн-сервісом, що підтримує функціональність розумних контрактів. Взаємодія з блокчейном Ethereum здійснюється за допомогою розроблених розумних контрактів.

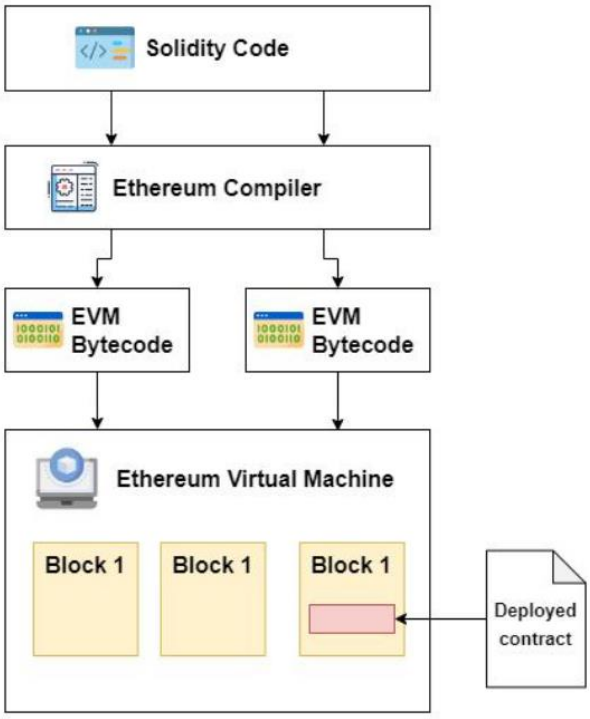

Рисунок 1.3 – Архітектура Ethereum

Блокчейн Ethereum є публічною розподіленою мережею, що надає можливість будь-якому учаснику долучатися до неї та виходити з неї за власним бажанням. Однією з ключових переваг публічного блокчейну є повна прозорість: кожна транзакція може бути відстежена будь-ким шляхом використання адреси. Ця характеристика є фундаментальною для розробки запропонованої архітектури системи.

Бізнес-логіка в Ethereum реалізується за допомогою мови програмування Solidity, яка має синтаксичну схожість з такими мовами, як Java, Python та JavaScript. Після написання вихідного коду він компілюється за допомогою компілятора Ethereum, який перевіряє наявність синтаксичних та логічних помилок. Компілятор генерує байт-код, що є машинно-зрозумілим для Віртуальної машини Ethereum (EVM). Після успішної компіляції та генерації байт-коду, контракт розгортається в мережі Ethereum. Валідація транзакцій здійснюється майнерами — вузлами мережі, що відповідають за їхню перевірку. Ethereum використовує алгоритм Proof of Work (Доказ роботи) для вибору майнера, який підтверджує блок транзакцій.

Віртуальна машина Ethereum (EVM) є середовищем виконання, в якому здійснюється обробка розумних контрактів. EVM функціонує в ізольованому "пісочному ящику", що означає повну відсутність доступу до зовнішніх ресурсів, таких як Інтернет, файлові системи або інші процеси, з боку контрактів, що виконуються всередині EVM.

Кожна транзакція, що створюється, асоціюється з певною кількістю, відомою як "газ" (Gas). Основними функціями газу є:

- Зниження обчислювальних зусиль, необхідних для виконання транзакції.

- Оплата за виконання операцій в EVM.

Вартість газу поступово зменшується в процесі обробки транзакції EVM відповідно до правил, визначених у розумному контракті. Ціна газу визначається ініціатором транзакції, який повинен сплатити її з рахунку відправника.

1.3.3.Розумні контракти та мова програмування Solidity

Розумний контракт є самовиконуваним протоколом, який містить програмний код, що визначає логіку взаємодії та умови виконання певних операцій. Він складається з:

- Контрактів: основних програмних одиниць.

- Функцій: блоків коду, що виконують певні дії.

- Методів: функцій, що можуть бути викликані ззовні.

Розумні контракти є фундаментальним елементом блокчейн-мережі, що регулює всі транзакції, які в ній відбуваються. Їх можна розглядати як програмний код, що визначає правила обробки будь-яких транзакцій, що виконуються в блокчейн-мережі. Розумний контракт — це набір програмних інструкцій, які розгортаються на блокчейні та визначають узгоджений набір правил, що повинні бути дотримані декількома сторонами для їхньої взаємодії. Виконання розумного контракту відбувається автоматично за умови дотримання певних, заздалегідь визначених умов.

Ethereum є провідною платформою для розробки блокчейн-мереж у сфері управління ланцюгами поставок. Для реалізації розумних контрактів, що виступають як складові угод, буде залучена організація Ethereum та фреймворк Truffle. Архітектура мережі Ganache (локальний блокчейн) демонструє, що представлена модель здатна ефективно виконувати транзакції управління ланцюгами поставок без використання централізованої бази даних.

Детальна архітектура програми для управління ланцюгом поставок з використанням розумних контрактів представлена на рисунку 1.4.

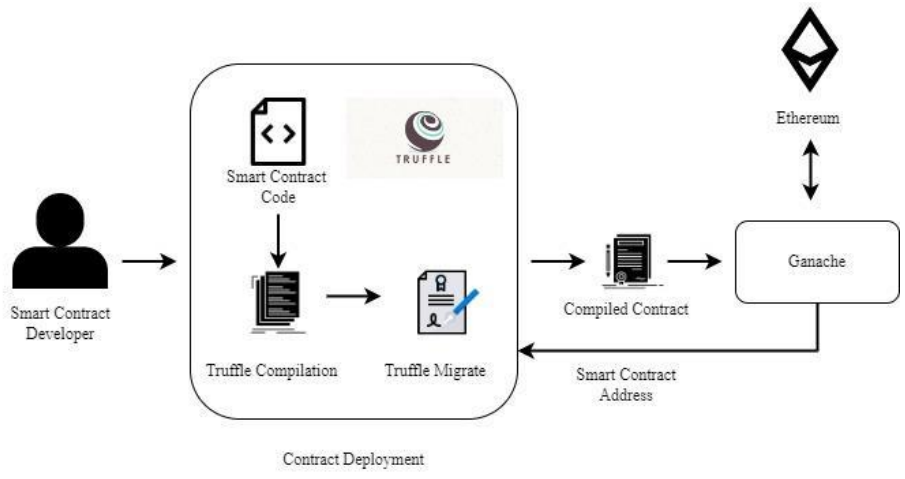

Рисунок 1.4 - Діаграма архітектури системи для смарт-контракту Ethereum

На рисунку 1.4 зображено архітектуру системи для розумних контрактів Ethereum в контексті управління ланцюгом поставок. Розробник смарт-контрактів створює розумний контракт для управління ланцюгом поставок і здійснює його розгортання за допомогою фреймворку Truffle. Агрегований байт-код розробленого розумного контракту відправляється в мережу Ethereum за допомогою Ganache

Розумні контракти можуть встановлювати зв'язки між окремими особами, організаціями та їхніми активами. Використання розумних контрактів дозволяє значно знизити транзакційні витрати шляхом автоматизації та стандартизації правил взаємодії. Це включає скорочення витрат на досягнення домовленостей, їх формалізацію та забезпечення виконання.

На основі таких розумних контрактів створюються децентралізовані додатки (DApps).

Розумні контракти відіграють ключову роль у процесі створення нових блоків у блокчейні. Розробка розумних контрактів для даного проєкту здійснюється за допомогою мови програмування Solidity.

У межах цього веб-додатка розроблено розумний контракт, що включає три основні функції:

1. Функція адміністрування.

Призначена для адміністраторів системи, що дозволяє ініціалізувати блокчейн та надавати авторизацію різним організаціям для додавання інформації в блокчейн.

2. Функція для організацій.

Дозволяє акредитованим організаціям (наприклад, сервісним центрам, страховим компаніям) створювати та додавати нові блоки даних до блокчейну.

3. Функція для клієнтів.

Забезпечує користувачів (потенційних покупців транспортних засобів) можливістю отримувати доступ до історичних даних про транспортний засіб, що зберігаються в блокчейні.

1.3.4. Взаємодія з Web3.js

Для забезпечення взаємодії між клієнтською частиною веб-додатка (фронтендом) та розумними контрактами, розгорнутими на блокчейні Ethereum, використовується бібліотека Web3.js. Ця JavaScript-бібліотека виступає як "міст", дозволяючи веб-додатку викликати функції розумних контрактів та отримувати дані з блокчейну.

Web3.js абстрагує складність взаємодії з Ethereum RPC (Remote Procedure Call) API. Замість того, щоб вручну формувати JSON-RPC запити, розробники можуть використовувати інтуїтивно зрозумілі JavaScript-методи. Наприклад, щоб отримати баланс рахунку, замість формування RPC-запиту, достатньо викликати web3.eth.getBalance('адреса_рахунку').

Web3.js є ключовим інструментом для розробки децентралізованих додатків (DApps), оскільки вона забезпечує основний інтерфейс для фронтенду DApp для взаємодії з базовим блокчейном Ethereum.

1.4. Аналіз існуючих платформ надання інформації про транспортні засоби

1.4.1. Веб-ресурс auto.ria

Авторіа (AUTO.RIA) – це одна з найбільших українських онлайн-платформ для купівлі, продажу та обміну транспортних засобів. Вона функціонує як агрегатор оголошень, дозволяючи приватним особам та дилерам розміщувати пропозиції щодо продажу автомобілів, мотоциклів, вантажівок, спецтехніки та інших видів транспорту.

Розглянемо ссновні функції та можливості Авторіа.

1. Розміщення та пошук оголошень

Для продавців: Можливість створювати детальні оголошення з фотографіями, описом технічних характеристик (марка, модель, рік випуску, пробіг, тип палива, об'єм двигуна, тип КПП, привід, кузов, колір, стан), комплектації та ціни. Платформа пропонує інструменти для просування оголошень (підняття в топ, виділення, тощо).

Для покупців: Розширені фільтри пошуку за різними параметрами, що дозволяють максимально точно підібрати потрібний автомобіль. Можливість зберігати цікаві оголошення, підписуватися на оновлення.

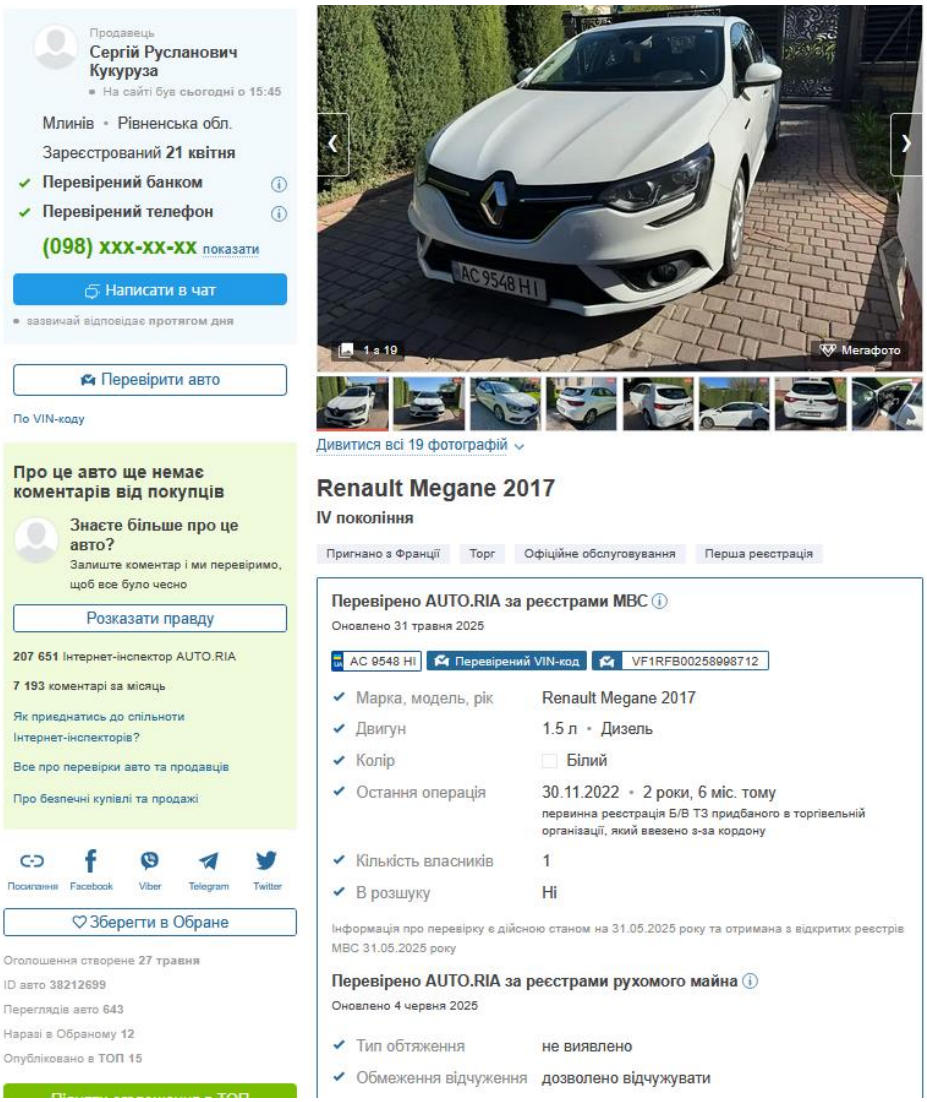

Рисунок 1.5 – Приклад опису ТЗ на ресурсі AUTO.RIA

2. Перевірка транспортних засобів:

Авторіа надає послуги з перевірки VIN-коду автомобіля за різними базами даних (державні реєстри, історія ДТП, дані про пробіг, сервісне обслуговування, участь у судових справах, наявність обтяжень, страхових випадків). Це дозволяє покупцям отримати додаткову інформацію про історію автомобіля та уникнути проблем.

Можливість замовлення діагностики автомобіля на партнерських СТО.

Додаткові сервіси:

- Кредитування та страхування: Співпраця з банками та страховими компаніями для надання послуг з оформлення кредитів та страхових полісів безпосередньо через платформу.

- Дошка оголошень: Розміщення оголошень про продаж запчастин, аксесуарів, послуг СТО.

- Новини та огляди: Інформаційний розділ з новинами автомобільного ринку, оглядами нових моделей, статтями про вибір та експлуатацію автомобілів.

- Оцінка вартості: Сервіси для онлайн-оцінки орієнтовної ринкової вартості автомобіля.

- Підтримка та консультації: Служба підтримки для користувачів.

Платформа побудована на сучасних веб-технологіях (Backend: Python/PHP/Node.js, Frontend: React/Angular/Vue.js). Використовує реляційні бази даних для зберігання великого обсягу структурованої інформації про оголошення, користувачів, перевірки.

Інтегрується із зовнішніми API державних реєстрів (наприклад, МВС України для даних про реєстрацію, VIN-коди) та інших постачальників даних (страхові компанії, СТО). Має мобільні додатки для iOS та Android.

Переваги Авторіа:

- Велика база оголошень, широкий вибір транспортних засобів.

- Інструменти перевірки дозволяють знизити ризики для покупців.

- Екосистема сервісів, додаткові послуги, що покривають значну частину потреб покупців і продавців.

Недоліки:

- Можливість шахрайства з боку недобросовісних продавців, незважаючи на інструменти перевірки.

- Платні послуги для просування оголошень.

- Залежність від актуальності та повноти даних, що надаються зовнішніми реєстрами.

Авторіа є комплексною онлайн-платформою, що охоплює значну частину вторинного ринку автомобілів в Україні та надає низку сервісів для забезпечення прозорості та зручності угод.

1.4.2. Веб-ресурс RST.ua

RST.ua – онлайн автомобільний майданчик в Україні, з понад 500 000 оголошень (і понад 400 000 вживаних авто). Платформа дозволяє купувати, продавати, обмінювати авто — від легкових до вантажівок, спецтехніки, мото та автобусів. Окрім продажу авто, RST.ua надає автоновини і огляди, має каталог нових автомобілів, поради з експлуатації та ремонту.

Функціонал та особливості:

- Розширений фільтр пошуку: за регіоном, брендом, роком, типом кузова тощо .

- Безкоштовне розміщення оголошень, можливість підняти їх — за кошт користувачів або дилерів

- Кожне авто має детальний опис, фотографії високої якості та технічні параметри

- Індикація цін у доларах та гривнях — комфорт для українського ринку

Переваги:

- Найбільший вибір: сотні тисяч авто.

- Простий інтерфейс, зручний на ПК та мобільному.

- Можливість торгуватися без посередників, працювати з дилерами.

- Детальний функціонал: пошук, обмін, новини.

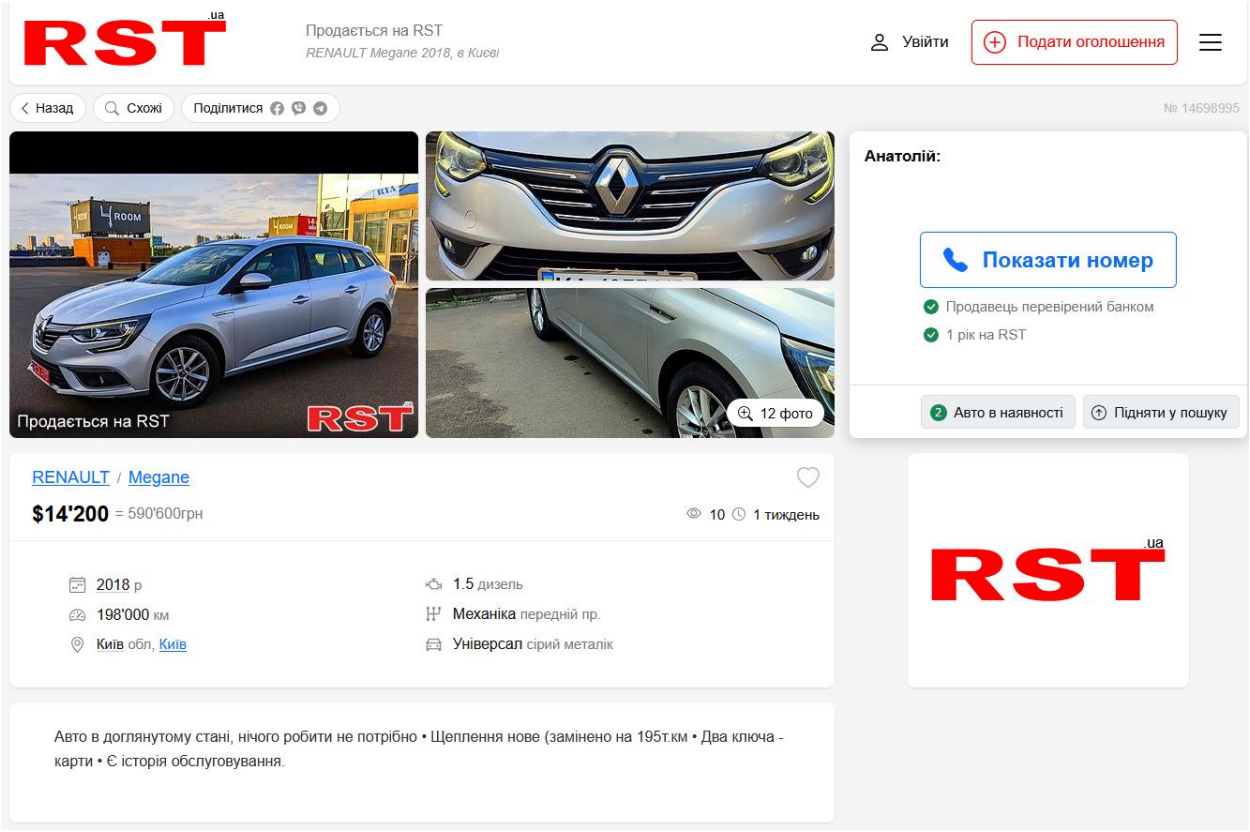

Рисунок 1.6 – Приклад опису ТЗ на ресурсі RST.UA

Нелодіки:

- Існують шахрайські оголошення — користувачі зазначають випадки фейкових продавців, шахрайства з передоплатами чи дрібними запчастинами

- Надійність завантажених фото/інформації залежить від продавця — необхідна обережність і перевірка.

RST.ua — потужний майданчик, який поєднує магазин авто + інформаційний ресурс. Це платформа для того, щоб знайти авто своєї мрії в Україні: зручний пошук, багатий вибір, мобільні додатки плюс корисні експертні матеріали. Проте, як і з будь-яким великим маркетплейсом, важливо перевіряти продавців і оголошення, щоб уникнути ризиків.

РОЗДІЛ 2. АНАЛІЗ ВИМОГ ТА ОПИС ІНСТРУМЕНТІВ РОЗРОБКИ СИСТЕМИ ВІДСЛІДКОВУВАННЯ ІНФОРМАЦІЇ ПРО ТРАНСПОРТНІ ЗАСОБИ

2.1. Специфікація системних вимог інформаційної системи відслідковування транспортних засобів

Цей підрозділ деталізує необхідні системні вимоги для розробки, тестування та експлуатації інформаційної системи. Вимоги розділені на дві основні категорії: для розробників та для кінцевих користувачів.

2.1.1. Системні вимоги для розробки та тестування

Апаратні вимоги:

- Пристрій: Ноутбук або персональний комп'ютер (ПК) з достатньою обчислювальною потужністю для розгортання локального блокчейн-оточення та компіляції смарт-контрактів.

Програмні вимоги:

- Операційна система: Microsoft Windows 10 або macOS (будь-яка актуальна версія).

- Інтегровані середовища розробки (IDE) / Редактори коду:

  - Visual Studio Code: для розробки фронтенду та бекенду веб-додатку.

  - Remix IDE: для розробки, компіляції та тестування розумних контрактів Solidity.

- Веб-браузер: Google Chrome (актуальна версія) для тестування веб-додатку.

- Розширення браузера: MetaMask (для взаємодії з блокчейн-мережею та управління цифровими активами).

- База даних / Зберігання даних:

- Розумний контракт (на основі блокчейну Ethereum): Основне сховище для незмінних даних про транспортні засоби.

- Firebase: Допоміжна база даних (наприклад, для зберігання метаданих або користувацьких налаштувань, які не потребують децентралізованого зберігання).

2.1.2. Системні вимоги для користувачів

Апаратні вимоги:

- Пристрій: Будь-який ноутбук або персональний комп'ютер.
- Оперативна пам'ять (RAM): Мінімум 2 ГБ.

Програмні вимоги:

- Операційна система: Microsoft Windows 10 або macOS (будь-яка актуальна версія).
- Веб-браузер: Google Chrome (актуальна версія)

2.2. Деталізація інструментів та технологій, використаних у розробці інформаційної системи

Цей підрозділ присвячений аналізу ключових інструментів та технологій, що були інтегровані в процес розробки інформаційної системи відстеження транспортних засобів.

2.2.1. MetaMask

MetaMask є програмним криптогаманцем, призначеним для зберігання Ether (ETH) – нативної криптовалюти блокчейну Ethereum. Він функціонує як розширення для звичайних веб-браузерів (наприклад, Chrome, Firefox), забезпечуючи взаємодію з блокчейном Ethereum. На відміну від спеціалізованих браузерів, таких як Mist або Dapps, які мають вбудовану сумісність з Ethereum, стандартні веб-браузери потребують додаткових

механізмів для прямої взаємодії з блокчейном. MetaMask вирішує цю проблему, ін'єктуючи бібліотеку Web3.js у середовище браузера, що дозволяє веб-додаткам взаємодіяти з блокчейном Ethereum через MetaMask.

Коли користувач бажає взаємодіяти з блокчейном Ethereum через звичайний браузер, MetaMask виступає посередником, зберігаючи Ether та керуючи транзакціями, тоді як браузер відповідає за відображення інтерфейсу та обробку даних. MetaMask підтримує як основну мережу Ethereum (Mainnet), так і три тестові мережі: Ropsten, Kovan та Rinkeby. Для тестування функціональності розробленого веб-додатку використовується тестова мережа Rinkeby.

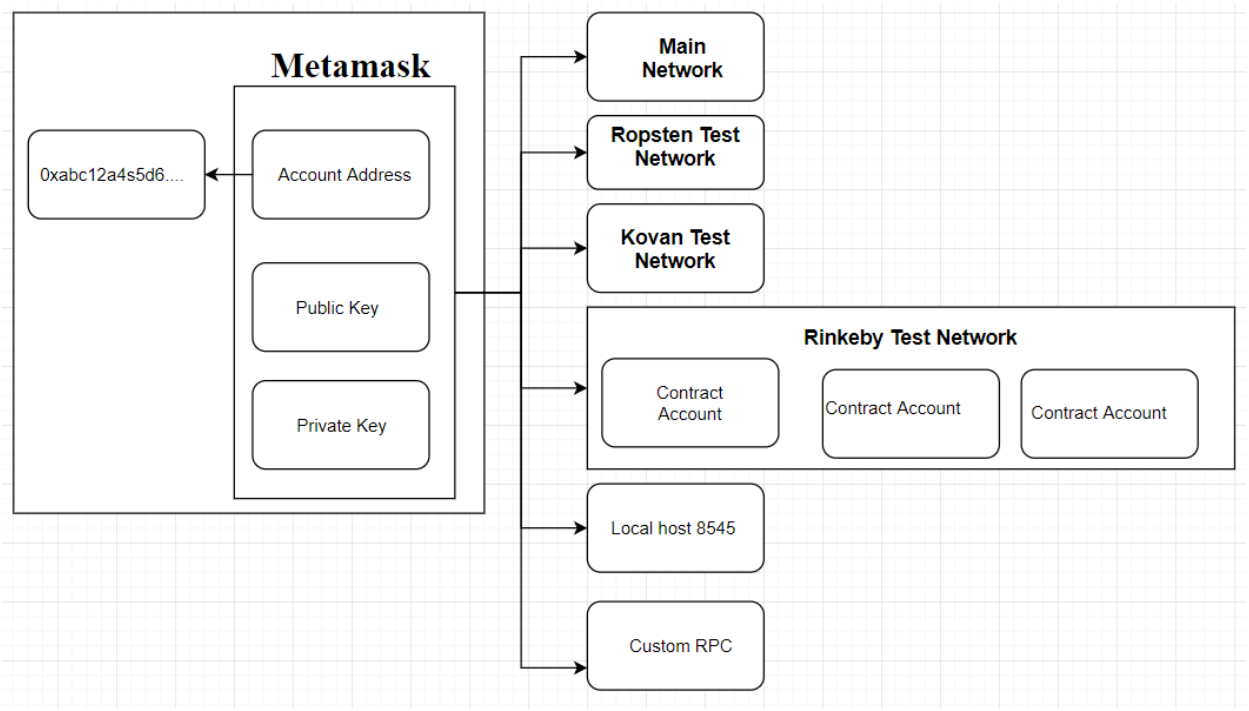

Рисунок 2.1 - Структурна діаграма Metamask

2.2.2 База даних Firebase

Firebase Realtime Database є хмарною базою даних, що надає API для синхронізації даних додатка в реальному часі між пристроями (iOS, Android, Web). Firebase є платформою, що дозволяє розробникам створювати спільні веб-додатки з функціональністю реального часу.

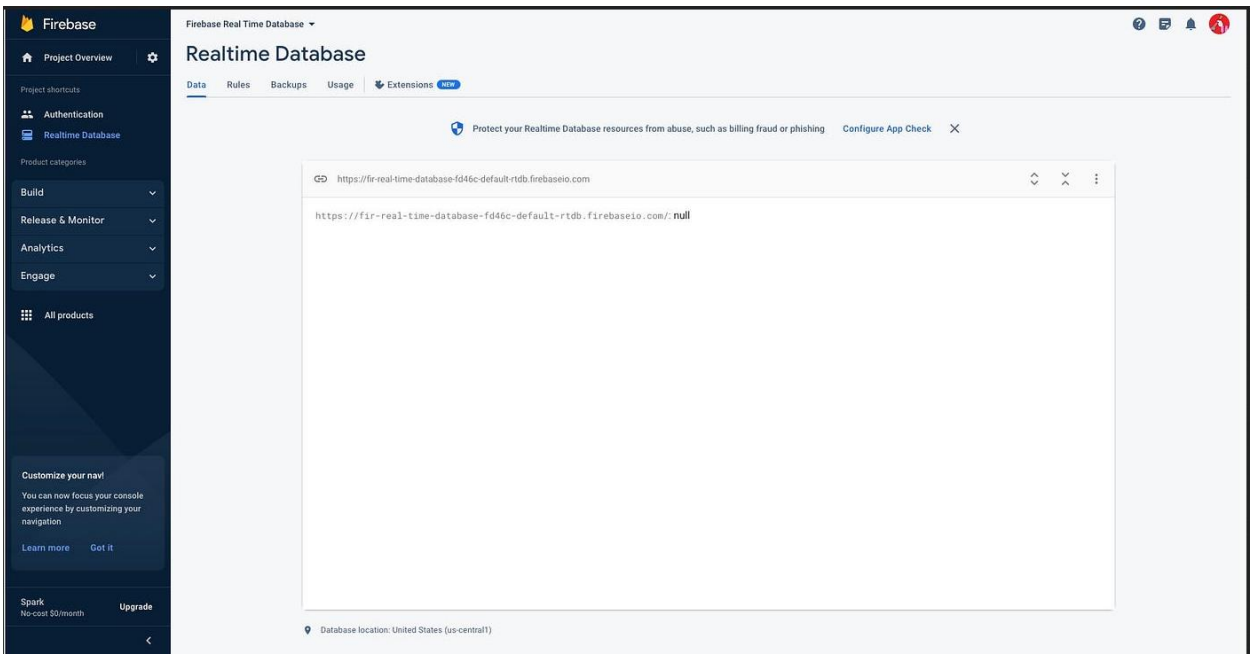

Рисунок 2.2 - Firebase Realtime Database

Firebase надає розширений набір сервісів, які дозволяють розробляти повноцінні веб-додатки, що включають:

- Аутентифікація Firebase: Сервіс для управління користувацькою аутентифікацією, підтримуючи інтеграцію з різними постачальниками соціального входу (наприклад, Gmail, Facebook, Twitter).

- База даних реального часу (Realtime Database): Хмарна база даних, що зберігає дані та надає асинхронний API для взаємодії.

- Сховище (Storage): Хмарне сховище для зберігання файлів.

- Хостинг (Hosting): Для розгортання веб-додатків.

- ML Kit: Мобільна платформа для машинного навчання.

- Хмарні повідомлення (Cloud Messaging): Для надсилання сповіщень.

Firebase підтримує взаємодію з клієнтськими бібліотеками для Node.js, JavaScript, Java, iOS та Android, забезпечуючи кросплатформну розробку.

2.2.3. Сервіс Infura

Для здійснення транзакцій в мережі Ethereum необхідно бути підключеним до її вузла. Infura є сервісом, що надає доступ до вузлів мережі

Ethereum без необхідності самостійного розгортання та підтримки повного вузла. Це дозволяє веб-додаткам взаємодіяти з мережею Ethereum.

Infura надає доступ до вузлів для чотирьох мереж: однієї основної мережі (Mainnet) та трьох тестових мереж (Rinkeby, Kovan та Ropsten). Infura забезпечує безпечний та спрощений доступ до мережі Ethereum. Після реєстрації на веб-сайті Infura користувачу надається унікальний токен, який використовується для встановлення з'єднання з відповідною мережею Ethereum.

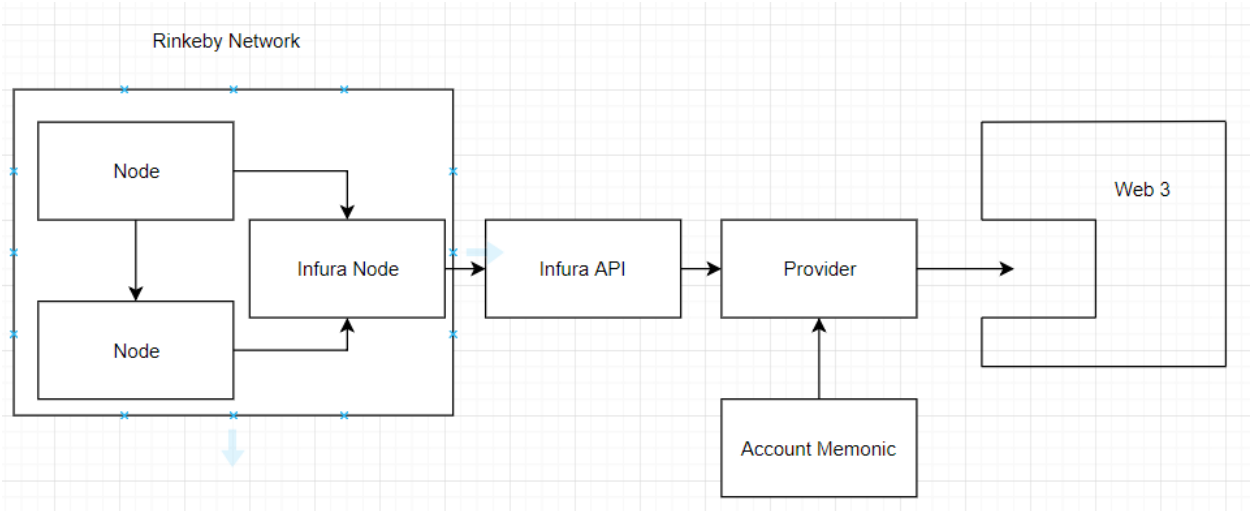

Рисунок 2.3 - Інтеграція Infura

У даному проєкті використовуються послуги Infura для підключення до тестової мережі Rinkeby та основної мережі Ethereum.

2.2.4. Node.js

Node.js є відкритим вихідним кодом, кросплатформним середовищем виконання JavaScript, що дозволяє виконувати код JavaScript поза браузером, зокрема на стороні сервера. Це розширює функціональність JavaScript для розробки серверних додатків.

У контексті розробки інформаційної системи відслідковування транспортних засобів, Node.js відіграє ключову роль як середовище

виконання для JavaScript на серверній стороні. Його використання дозволяє розширити можливості JavaScript за межі традиційного клієнтського виконання у веб-браузері.

Основні аспекти використання Node.js у даному проєкті:

1. Серверне середовище виконання JavaScript:

- Node.js надає можливість виконувати JavaScript-код безпосередньо на сервері. Це дозволяє використовувати одну мову програмування (JavaScript) для розробки як фронтенду (завдяки Web3.js для взаємодії з блокчейном та DOM-маніпуляціям), так і бекенду додатка. Така уніфікація спрощує процес розробки та підтримки кодової бази.

- Він слугує платформою для побудови API, що може обробляти запити від клієнтської частини веб-додатка та взаємодіяти з іншими компонентами системи, такими як Firebase або зовнішні сервіси.

2. Інтеграція з децентралізованими технологіями:

- Хоча основна логіка взаємодії з блокчейном Ethereum через розумні контракти реалізується на клієнтській стороні за допомогою Web3.js, Node.js може бути використаний для серверних компонентів, які потребують взаємодії з блокчейном. Наприклад, для запуску фонових процесів, які моніторять події в блокчейні, або для виконання більш складних обчислень, що не підходять для клієнтської сторони.

- Він може слугувати як проксі-сервер для перенаправлення запитів до таких сервісів, як Infura, або для управління ключами та підписання транзакцій на стороні сервера в контрольованому середовищі, якщо це передбачено архітектурою безпеки.

3. Підтримка асинхронних операцій:

- Node.js заснований на асинхронній, подієво-орієнтованій архітектурі, що є ідеальною для роботи з I/O-інтенсивними операціями, такими як мережеві запити. Це дозволяє ефективно обробляти велику кількість одночасних з'єднань, що є важливим для веб-додатків.

- У контексті взаємодії з блокчейном, де транзакції можуть займати певний час для підтвердження, асинхронність Node.js дозволяє додатку залишатися чутливим до користувача, не блокуючи виконання інших операцій під час очікування відповіді від блокчейн-мережі.

4. Екосистема NPM:

Node.js має найбільшу у світі екосистему пакетів (NPM - Node Package Manager), що надає доступ до тисяч готових бібліотек. Це прискорює розробку, дозволяючи інтегрувати вже існуючі рішення для різних завдань, включаючи роботу з базами даних, валідацію даних, безпеку та багато іншого.

Таким чином, Node.js в даному проєкті виступає як потужне і гнучке серверне середовище, що доповнює клієнтську частину на JavaScript, забезпечуючи надійну взаємодію з блокчейном Ethereum та іншими допоміжними сервісами, такими як Firebase, для реалізації повнофункціональної інформаційної системи.

2.3. Використані технології та мови програмування

2.3.1. Ethereum

Ethereum використовує децентралізовану та розподілену мережу, де кожна система/вузол мережі зберігає копію блокчейну — послідовної бази даних транзакцій. Для запису даних у блокчейн Ethereum застосовуються розумні контракти, написані мовою програмування Solidity. Виконання розумного контракту призводить до збереження даних у блокчейні, забезпечуючи їхню незмінність та прозорість.

2.3.1. JavaScript

JavaScript – це клієнтська скриптова мова програмування, яка була використана у цьому проєкті для веб-розробки. Вона відповідає за обробку

клієнтської валідації даних та інтерактивної взаємодії з користувачем. JavaScript активно використовує об'єктну модель документа (DOM), що дозволяє маніпулювати елементами веб-сторінки, такими як зображення, посилання, форми тощо, надаючи властивості та методи для цієї мети.

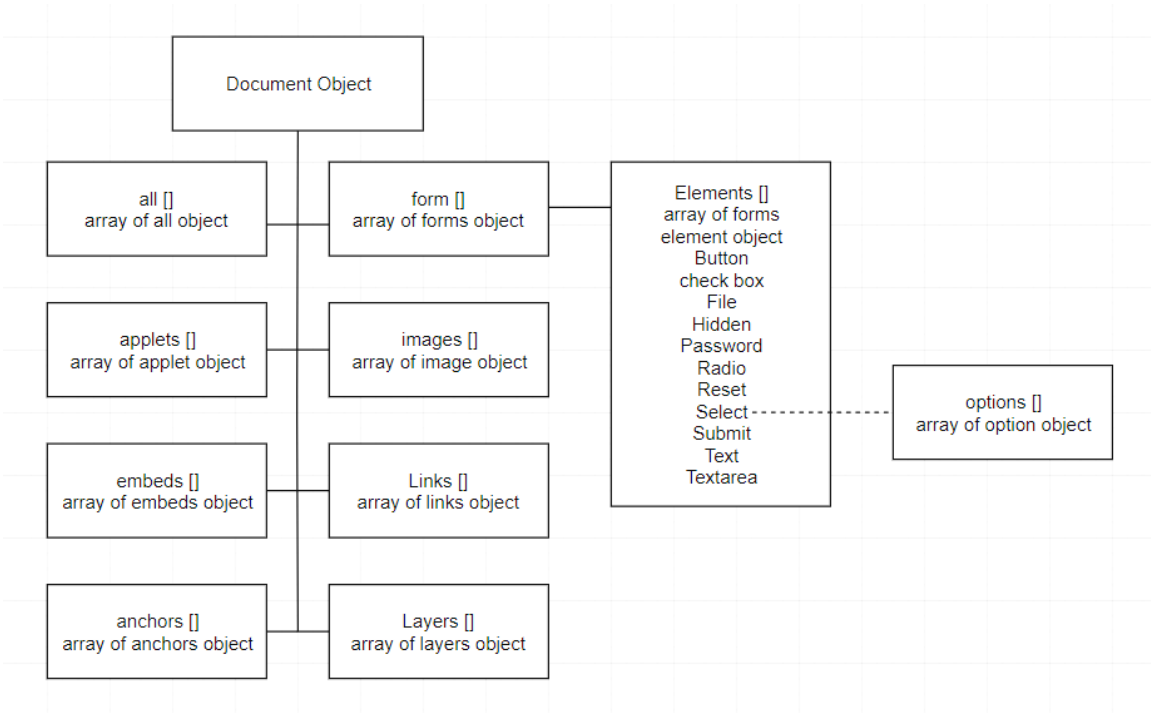

Рисунок 2.4 - Об'єктна модель документа JavaScript (DOM)

2.3.2. Solidity

Solidity – це об'єктно-орієнтована мова програмування, спеціально розроблена для написання розумних контрактів, що виконуються на Віртуальній машині Ethereum (EVM). Розроблена ключовими членами команди Ethereum, Solidity зазнала впливу таких мов, як JavaScript, C++ та Python.

Для розробки розумних контрактів на Solidity використовується Remix IDE – веб-редактор, який надає всі необхідні версії компілятора. Альтернативно, для компіляції коду Solidity без використання Remix IDE, застосовується компілятор Solc.

Remix IDE доступний безпосередньо через веб-браузер за адресою remix.ethereum.org. Це усуває потребу в локальній інсталяції, що робить його дуже зручним для швидкого початку роботи та доступу з будь-якого пристрою.

Інтерфейс є інтуїтивно зрозумілим і поділений на кілька панелей: файловий менеджер, редактор коду, компілятор, розгортання та панель налагодження.

Вбудований компілятор Solidity дозволяє компілювати написаний код безпосередньо в IDE. Після компіляції Remix надає згенерований байт-код та ABI (Application Binary Interface) контракту, які необхідні для розгортання та взаємодії з контрактом. Виявляє синтаксичні помилки та попередження, допомагаючи розробникам виправляти недоліки в коді.

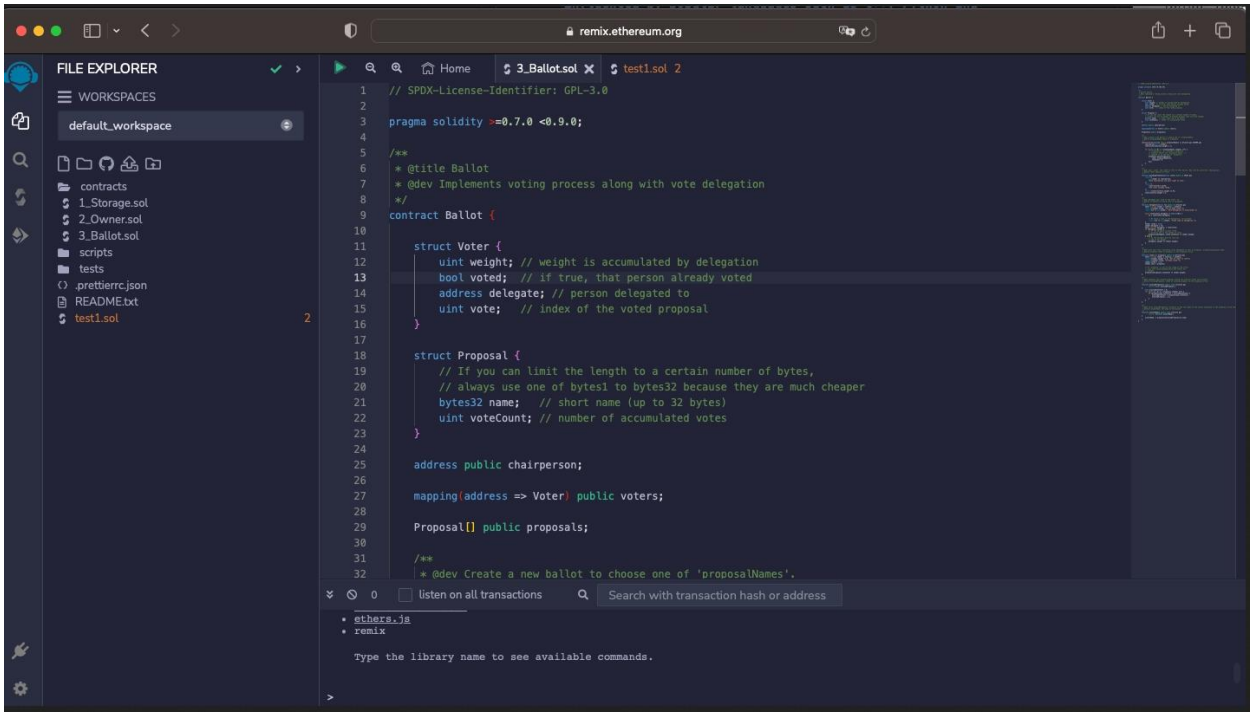

Рисунок 2.5 – Редактор Remix IDE

Remix IDE дозволяє розгортати контракти в декількох середовищах:

- JavaScript VM (віртуальна машина JavaScript) - локальне середовище, яке імітує блокчейн, ідеальне для швидкого тестування без витрат газу.

- Injected Provider (наприклад, MetaMask) - дозволяє підключатися до реальних мереж Ethereum (Mainnet або тестових мереж, таких як Rinkeby, Ropsten тощо) через розширення браузера, таке як MetaMask, для розгортання та взаємодії з реальними транзакціями.

- Web3 Provider - дозволяє підключатися до локальної ноди Ethereum (наприклад, Ganache або Geth) або віддаленої ноди (наприклад, через Infura).

Після розгортання контракту Remix надає зручний інтерфейс для виклику функцій контракту, передачі параметрів та перегляду результатів транзакцій.

Remix має модульну архітектуру з підтримкою плагінів, що дозволяє розширювати його функціональність. Наприклад, існують плагіни для аналізу безпеки контрактів, інтеграції з IPFS, візуалізації потоку виконання та інші.

РОЗДІЛ 3. МОДЕЛЮВАННЯ ПРОЦЕСІВ РОЗРОБКИ ІНФОРМАЦІЙНОЇ СИСТЕМИ ЗА ДОПОМОГОЮ ДІАГРАМ ВАРІАНТІВ ВИКОРИСТАННЯ

Цей розділ представляє аналіз взаємодії різних категорій користувачів із розробленою інформаційною системою відстеження транспортних засобів та її ключовими компонентами. Для візуалізації цих взаємозв'язків використано діаграми варіантів використання, що відображають функціональні вимоги до системи з точки зору акторів.

3.1. Розробка діаграм варіантів використання взаємодії адміністратора та автентифікованих користувачів

3.1.1. Діаграма варіантів використання взаємодії адміністратора

На рисунку 3.1 зображено взаємодію адміністратора з розумним контрактом та веб-додатком. Адміністратор виконує початкове розгортання розумного контракту у блокчейн-мережі, після чого взаємодіє з ним через інтерфейс веб-додатку. Цей процес є критично важливим для ініціалізації системи.

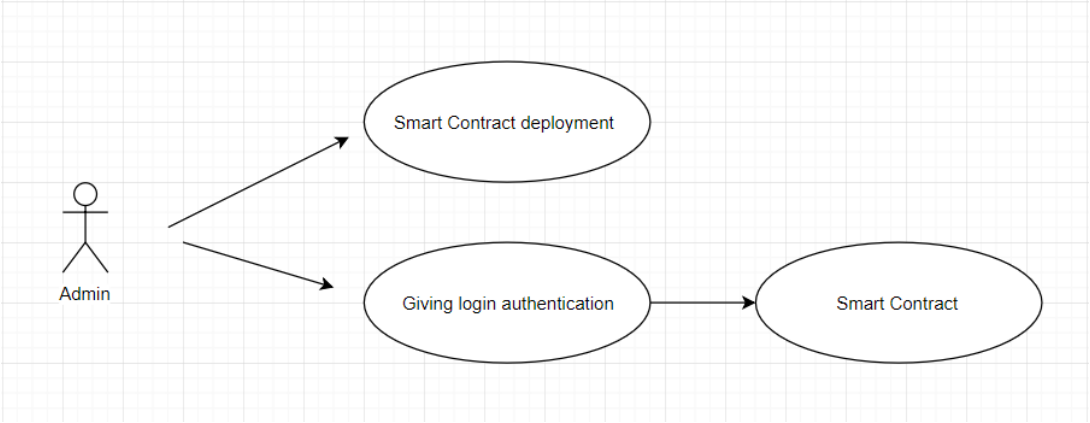

Рисунок 3.1 - Діаграма варіантів використання взаємодії адміністратора

На цій діаграмі представлений один актор Admin (Адміністратор), що представляє користувача або роль, яка виконує адміністративні функції в системі.

Від актора "Admin" виходять два основні варіанти використання:

1. Smart Contract deployment (Розгортання смарт-контракту).

Цей варіант використання відображає пряму дію, яку виконує Адміністратор. Він означає, що Адміністратор відповідає за початкове розгортання або оновлення смарт-контракту в блокчейн-мережі.

2. Giving login authentication (Надання логін-аутентифікації).

Цей варіант використання показує іншу ключову функцію Адміністратора – управління доступом користувачів до системи. Адміністратор надає дозволи на вхід (аутентифікацію) для інших користувачів або організацій.

Від цього варіанта використання "Giving login authentication" веде стрілка до іншого варіанта використання "Smart Contract". Це інтерпретується як відношення «включення» (include) або «узалежнення». У даному контексті, це означає, що процес надання логін-аутентифікації включає (або залежить від) взаємодію зі Смарт-контрактом. Тобто, логіка управління аутентифікацією реалізована або зберігається в смарт-контракті, і адміністратор використовує функціональність смарт-контракту для виконання цієї операції. Це підкреслює, що смарт-контракт не тільки розгортається, але й активно використовується для управління доступом.

Управління системою аутентифікації користувачів, яка, у свою чергу, тісно інтегрована зі смарт-контрактом, вказуючи на те, що смарт-контракт використовується для зберігання або обробки інформації про доступ.

На діаграмі показана типова роль адміністратора в децентралізованих системах, де початкове налаштування та управління правами доступу часто є централізованою функцією, реалізованою через адміністративні функції смарт-контракту.

3.1.2. Взаємодія автентифікованих користувачів з веб-додатком

Рисунок 3.3 ілюструє взаємодію автентифікованих користувачів з розумним контрактом через веб-додаток. Адміністратор несе відповідальність за надання прав доступу (аутентифікації) певним організаціям, що дозволяє цим автентифікованим користувачам додавати інформацію до розумного контракту.

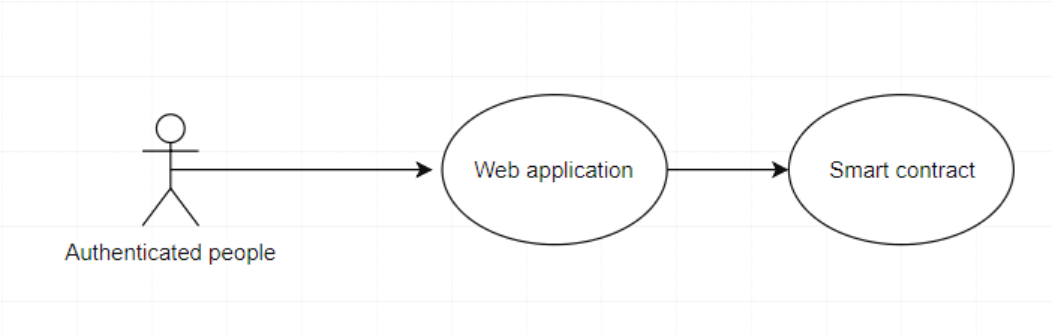

Рисунок 3.2 - Взаємодія автентифікованих користувачів з веб-додатком

Ця діаграма включає одного актора та два варіанти використання:

1. Authenticated people (Автентифіковані особи).

Цей актор представляє користувачів або групи користувачів, які вже пройшли процес аутентифікації в системі. Це можуть бути, наприклад, представники організацій (страхові компанії, СТО, дилери), які отримали дозвіл від адміністратора.

2. Web application (Веб-додаток).

Це перший варіант використання, з яким взаємодіє актор "Authenticated people". Стрілка від актора до "Web application" вказує на те, що автентифіковані особи використовують веб-додаток для виконання своїх функцій. Це означає, що веб-додаток є основним інтерфейсом для взаємодії користувачів із системою.

3. Smart contract (Смарт-контракт).

Це другий компонент, який взаємодіє з "Web application". Стрілка від "Web application" до "Smart contract" вказує на те, що веб-додаток, у свою

чергу, взаємодіє зі смарт-контрактом. Це означає, що функціональність, доступна через веб-додаток для автентифікованих користувачів, реалізується або зберігається за допомогою смарт-контракту на блокчейні. Наприклад, додавання, зміна або отримання даних, що стосуються транспортних засобів, відбувається через виклики функцій смарт-контракту.

Автентифіковані особи взаємодіють із системою через веб-додаток. Веб-додаток слугує проміжним шаром, який перетворює дії користувача на виклики до смарт-контракту, розташованого в блокчейн-мережі.

Таким чином, смарт-контракт є кінцевим сховищем даних та логіки, тоді як веб-додаток надає зручний інтерфейс для доступу до цієї функціональності для автентифікованих користувачів.

3.2. Проектування діаграм варіантів використання аутентифікація організацій та взаємодія користувача з веб-додатком

3.2.1. Діаграма варіантів використання для організації

На рисунку 3.3 показано процес, за якого різні організації отримують аутентифікацію від адміністратора для взаємодії з системою. Ця аутентифікація є передумовою для додавання даних до блокчейну.

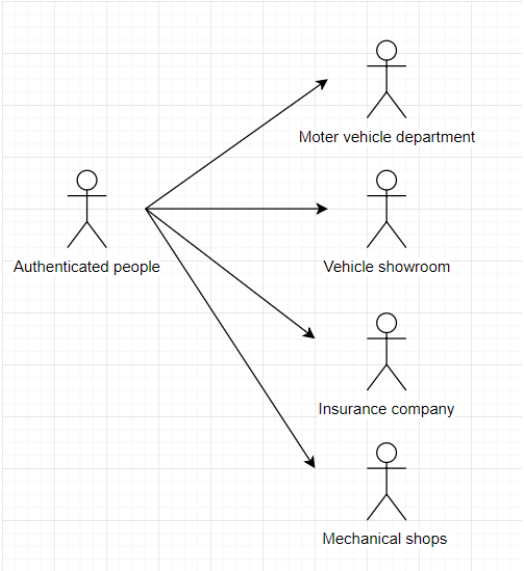

Рисунок 3.3 – Діаграма варіантів для аутентифікованих організацій

На цій діаграмі представлено п'ять акторів:

1. Authenticated people (Автентифіковані особи) - це головний актор. Він представляє загальну категорію користувачів, які пройшли аутентифікацію в системі. Розширення ролі: "Authenticated people" є узагальненою роллю, яка потім спеціалізується на конкретних організаціях. Це означає, що "Моторний відділ транспортних засобів", "Автосалон", "Страхова компанія" та "Механічні майстерні" є специфічними типами "Автентифікованих осіб", які мають дозволи на взаємодію з системою.

Інші чотири актори є конкретними типами організацій, які будуть автентифікованими учасниками системи:

2. Motor vehicle department (Моторний відділ транспортних засобів / Відділ реєстрації транспортних засобів) - це державний орган або департамент, відповідальний за реєстрацію, ліцензування або інші адміністративні функції, пов'язані з транспортними засобами. Вони додаватимуть офіційні дані до системи.

3. Vehicle showroom (Автосалон) - представляє дилерські центри або продавців автомобілів. Вони можуть вносити дані про первинний продаж транспортних засобів, інформацію про гарантії або акції.

4. Insurance company (Страхова компанія) - цей актор відповідає за внесення даних, пов'язаних зі страхуванням транспортних засобів, включаючи поліси, страхові випадки та виплати.

5. Mechanical shops (Механічні майстерні / СТО) - представляють станції технічного обслуговування, які вноситимуть дані про виконані ремонтні роботи, планове технічне обслуговування, заміни деталей тощо.

3.2.2. Діаграма варіантів використання взаємодії кінцевого користувача

Після того, як дані додані до блокчейну авторизованими організаціями, кінцеві користувачі отримують доступ до цієї інформації через веб-додаток.

Рисунок 3.4 деталізує цю взаємодію, де користувач запитує та отримує дані про транспортний засіб.

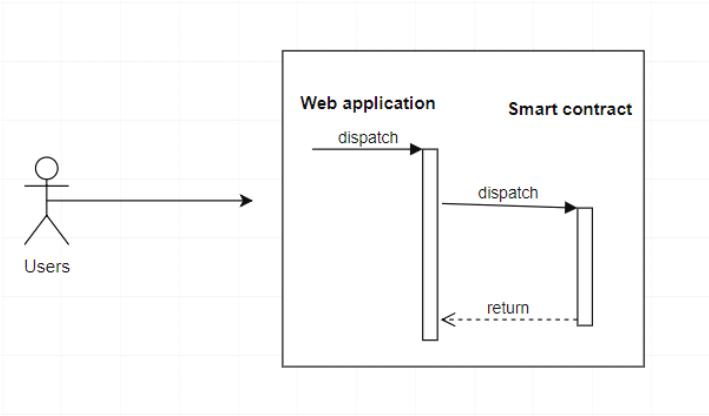

Рисунок 3.4 - Взаємодія користувача з веб-додатком

Початковим кроком у функціонуванні системи є розгортання розумного контракту адміністратором у мережі Ethereum. Після цього організації та кінцеві користувачі взаємодіють з ним через веб-додаток. Важливо зазначити, що розгортається лише один розумний контракт, який містить різні функції, призначені для виконання специфічних завдань для кожної категорії акторів.

3.3. Діаграма варіантів використання взаємодії з розумним контрактом

На рисунку 3.5 представлена узагальнена діаграма, що ілюструє всі типи взаємодії з розумним контрактом, демонструючи, як адміністратор, автентифіковані організації та кінцеві користувачі спілкуються з центральним блокчейн-компонентом.

Діаграма представляє кілька акторів та компоненти взаємодії.

Admin - цей актор виконує початкову дію — взаємодіє зі Смарт-контрактом. Стрілка від "Admin" до "Smart contract" вказує на те, що

Адміністратор ініціює дії, пов'язані зі смарт-контрактом (наприклад, його розгортання, ініціалізацію, або управління основними параметрами).

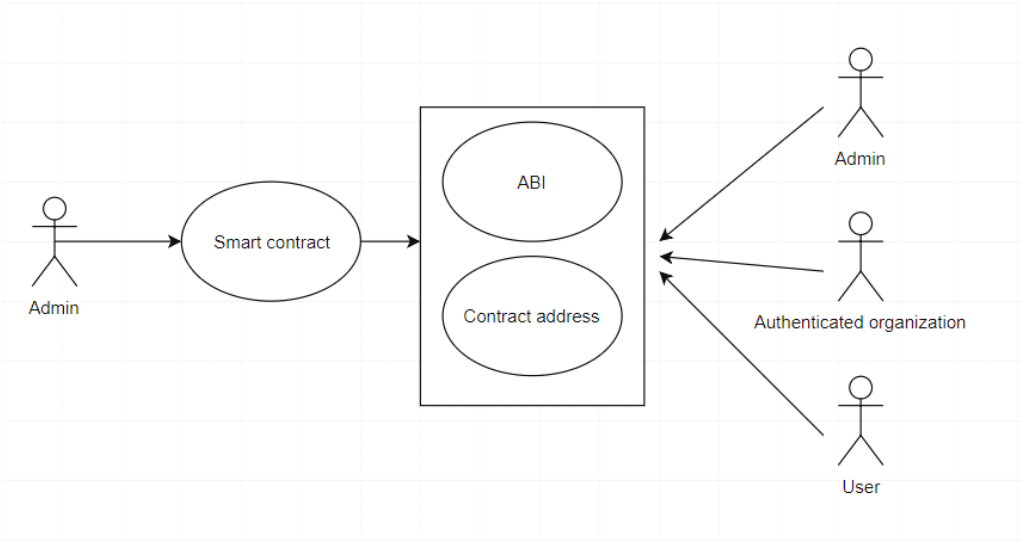

Рисунок 3.5 – Діаграма варіантів використання взаємодії користувачів з розумним контрактом

Admin [другий екземпляр на правій стороні] - це той самий актор, але його повторне зображення (або використання) з правої сторони діаграми вказує на його подальшу взаємодію з системою через інтерфейс контракту.

Authenticated organization - цей актор представляє організації, які пройшли аутентифікацію і мають дозвіл на взаємодію зі смарт-контрактом (наприклад, для додавання або оновлення даних).

User - цей актор представляє кінцевих користувачів системи, які, ймовірно, мають доступ до даних, що зберігаються в смарт-контракті.

Розглянемо компоненти взаємодії.

Smart contract (Смарт-контракт) - центральний елемент системи. Стрілка від першого "Admin" до "Smart contract" показує його початкову роль.

Прямокутник, що містить "ABI" та "Contract address" візуально об'єднує два ключові елементи, необхідні для взаємодії зі смарт-контрактом після його розгортання:

- ABI (Application Binary Interface) - опис інтерфейсу смарт-контракту. Він визначає, які функції доступні в контракті, які аргументи вони приймають, і які типи даних повертають. Без ABI неможливо програмно викликати функції смарт-контракту.

- Contract address (Адреса контракту) - унікальна адреса в блокчейн-мережі, за якою розгорнуто смарт-контракт. Це аналог IP-адреси або URL для веб-сервісу, що дозволяє знайти та взаємодіяти з конкретним екземпляром контракту.

Стрілка від "Smart contract" до цього прямокутника показує, що після розгортання смарт-контракту генеруються або стають доступними його ABI та адреса.

Стрілки від "Admin" (правого), "Authenticated organization" та "User" спрямовані до прямокутника з "ABI" та "Contract address" означає, що всі ці актори взаємодіють зі смарт-контрактом, використовуючи його ABI та адресу.

Admin використовує ABI та адресу для виконання адміністративних функцій (наприклад, управління правами, оновлення контракту, якщо це передбачено). Authenticated organization використовує ABI та адресу для додавання або зміни інформації, пов'язаної з транспортними засобами (наприклад, дані про ремонт, страховку). User використовує ABI та адресу для отримання інформації про транспортні засоби.

Ця діаграма показує архітектуру взаємодії з блокчейн-системою:

1. Адміністратор ініціює розгортання смарт-контракту.

2. Після розгортання, ABI та адреса контракту стають доступними. Ці два елементи є "інтерфейсом" для взаємодії зі смарт-контрактом.

3. Всі типи користувачів (адміністратор, автентифіковані організації та користувачі) взаємодіють зі смарт-контрактом виключно через його ABI та адресу, що є стандартним механізмом для взаємодії з контрактами в блокчейні Ethereum.

3.4. Представлення архітектурної діаграми системи

Рисунок 3.6 відображає комплексну діаграму взаємодії між усіма технологіями та компонентами, використаними у веб-додатку, забезпечуючи цілісне уявлення про архітектуру системи та її функціональність.

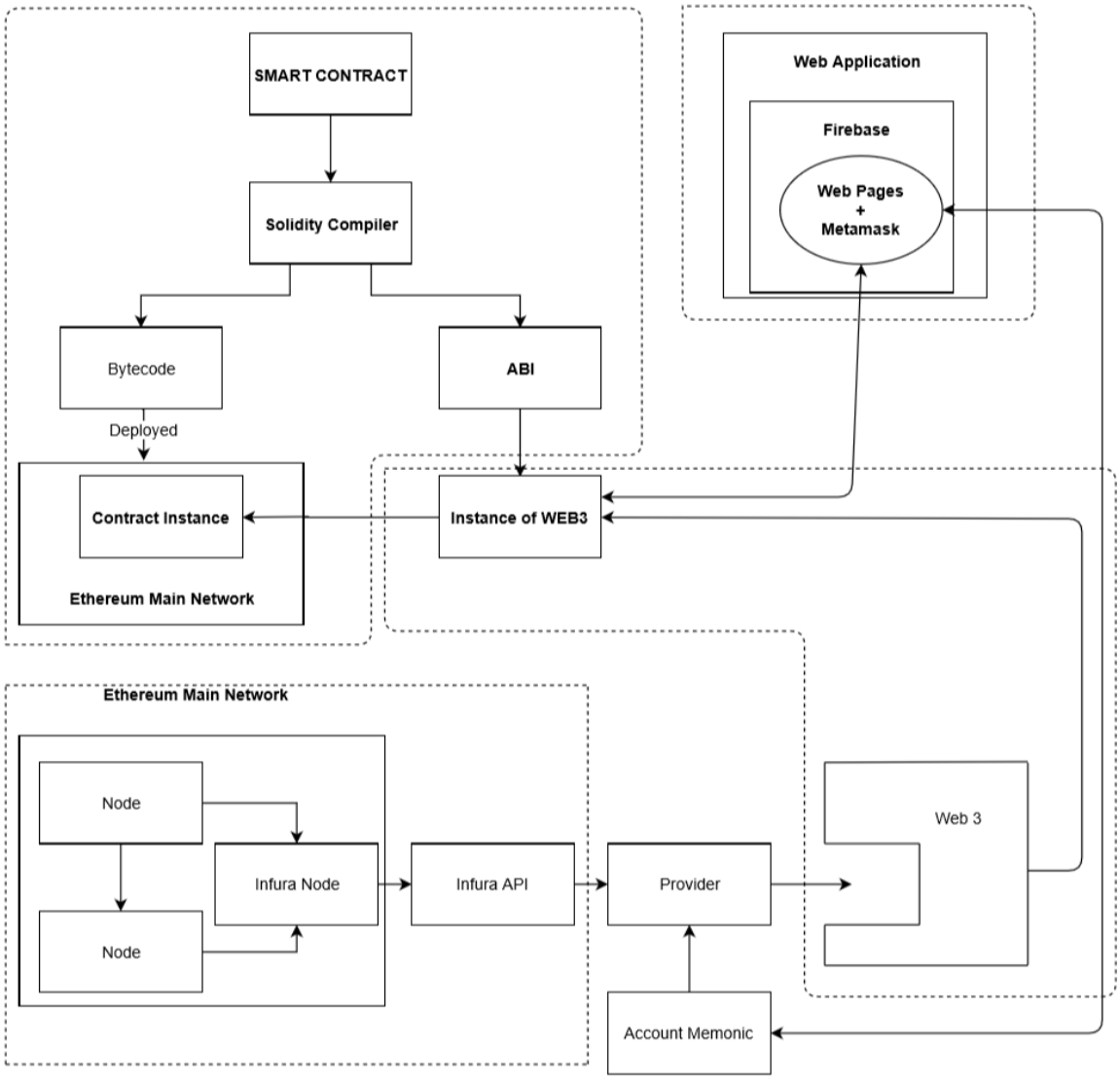

Рисунок 3.6 - Архітектурна діаграма системи

Діаграма розділена на три основні логічні блоки:

1. Блок розробки та розгортання смарт-контракту:

- SMART CONTRACT: Це вихідний код розумного контракту, написаний мовою Solidity.

- Solidity Compiler: Компілює вихідний код смарт-контракту.

- Bytecode (Байт-код): Результат компіляції, машинно-зрозумілий код для EVM. Цей байт-код розгортається (Deployed) в Ethereum Main Network, що призводить до створення Contract Instance (Екземпляра контракту).

- ABI (Application Binary Interface): Також є результатом компіляції. ABI — це JSON-файл, який описує інтерфейс смарт-контракту (доступні функції, їхні параметри, типи даних). ABI не розгортається на блокчейні, а використовується для взаємодії з розгорнутим контрактом.

2. Блок веб-додатка:

- Web Application (Веб-додаток): Це клієнтська частина системи.

- Firebase: частина веб-додатка, що передбачає використання сервісів Firebase (наприклад, хостинг, Realtime Database для додаткових даних, аутентифікація).

- Web Pages + Metamask (Веб-сторінки + Metamask): Це інтерфейс користувача. Веб-сторінки відображають контент, а Metamask інтегрується з ними як гаманець, що дозволяє користувачеві підписувати транзакції та керувати Ether.

- Стрілка від "Instance of Web3" до "Web Pages + Metamask" вказує на те, що Web3.js бібліотека інтегрується у веб-сторінки через Metamask, забезпечуючи взаємодію з блокчейном.

3. Блок взаємодії з мережею Ethereum та допоміжними сервісами:

- Ethereum Main Network (Основна мережа Ethereum):

- Node, Node: Представляють вузли блокчейну Ethereum, що формують мережу.

- Infura Node: Спеціалізований вузол, який надається сервісом Infura для зручного доступу до мережі Ethereum без необхідності запуску власного повного вузла.

- Infura API: Програмний інтерфейс, який надає Infura для взаємодії з їхніми вузлами.

- Provider: Компонент, який служить інтерфейсом для взаємодії з блокчейн-мережею через Infura API. Він отримує запити від Web3.js та передає їх до Infura API.

- Account Mnemonic: Відноситься до Seed Phrase гаманця, яка використовується для генерації ключів облікового запису. Хоча це не є частиною мережі в прямому сенсі, вона є критично важливою для управління рахунками, які взаємодіють з мережею.

4. Web 3 (Web3.js):

- Instance of Web3: Це об'єкт бібліотеки Web3.js, який використовується для взаємодії зі смарт-контрактом. Він отримує ABI від компілятора та взаємодіє з Contract Instance через провайдера.

Розробник пише SMART CONTRACT (Solidity), який компілюється в Bytecode та ABI. Байт-код розгортається в Ethereum Main Network, створюючи Contract Instance. Веб-додаток, використовуючи Web Pages + Metamask, ініціює запити. Ці запити обробляються Instance of Web3, який використовує отриманий ABI для правильного формування викликів до Contract Instance.

Для фактичного надсилання запитів до Ethereum Main Network використовується Infura API через Infura Node. Provider служить інтерфейсом між Web3.js та Infura API, забезпечуючи підключення. Account Mnemonic (гаманець Metamask) керує криптографічними ключами, необхідними для підписання транзакцій, які потім передаються через Web3.js та провайдера до мережі Ethereum.

Діаграма надає уявлення про архітектуру децентралізованого додатка (DApp). Вона демонструє, як смарт-контракти розробляються та розгортаються, як клієнтський веб-додаток взаємодіє з цими контрактами через бібліотеку Web3.js, використовуючи сервіс Infura для доступу до мережі Ethereum, і як Metamask забезпечує безпечне управління гаманцем та підписання транзакцій для користувача.

РОЗДІЛ 4. ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ ВІДСЛІДКОВУВАННЯ ІНФОРМАЦІЇ ПРО ТРАНСПОРТНІ ЗАСОБИ

4.1. Імплементації розумного контракту для системи відслідковування транспортних засобів

Цей підрозділ зосереджений на специфікації та реалізації ключового компонента системи – розумного контракту (Smart Contract), який виконує функцію протоколу для збереження та управління даними у блокчейні Ethereum.

Розумний контракт – це самовиконуваний код, що зберігається та працює в децентралізованій мережі блокчейну. Він є основним механізмом для забезпечення прозорості та незмінності інформації про транспортні засоби.

4.1.1. Створення контракту та успадкування

Для створення контракту в Solidity використовується ключове слово contract, аналогічно оголошенню класу в об'єктно-орієнтованих мовах програмування. У даному випадку, контракт VehicleVerification успадковує функціональність від контракту Ownable, імпортованого з файлу AccessControl.sol.

```
3 import "./AccessControl.sol";

4

5 contract VehicleVerification is Ownable {

6 }
```

Рисунок 4.1 - Створення контракту в Solidity

4.1.2. Контроль доступу (Ownable контракт)

Контракт Ownable реалізує базовий механізм контролю доступу, надаючи функціональність для визначення власника контракту. Це забезпечує, що певні критичні функції можуть бути викликані лише авторизованим власником (адміністратором).

```
/**
 * @title Ownable
 * @dev Контракт Ownable має адресу власника та надає базові функції авторизації
 * функції, це спрощує реалізацію "дозволів користувача".
 */
contract Ownable {
    address private _owner;

    /**
     * @dev Конструктор Ownable встановлює оригінального `власника` контракту на облікo
     * обліковий запис відправника.
     */
    constructor() public {
        _owner = msg.sender;
    }

    /**
     * @dev Видає помилку, якщо викликається будь-яким обліковим записом, крім власника.
     */
    modifier onlyOwner() {
        require(isOwner());
        _;
    }

    /**
     * @return true, якщо `msg.sender` є власником контракту.
     */
    function isOwner() public view returns (bool) {
        return msg.sender == _owner;
    }
}
```

Рисунок 4.2 - Код контролю доступу

4.1.2. Визначення структур даних

Для організації та зберігання інформації про транспортні засоби використовується чотири структурні типи даних, що відповідають різним джерелам даних (автосалон, страхова компанія, реєстраційний відділ та механічна майстерня).

а) Структура showroomData:

Ця структура призначена для зберігання інформації, наданої автосалонами.

```
7 struct showroomData {
8 bytes32 vin;
9 bytes32 ownerDetail;
10 bytes32 modelDetail;
11 bytes32 carCost;
12 bytes32 carLoanOrBankDetail;
13 bytes32 showroomName;
14 bytes32 showroomAddress;
15 bytes32 showroomId;
16 uint256 issuedDate;
17 bytes32 additionalDetail;
18 }
```

Рисунок 4.3 - Структура типу даних Showroom

Пояснення полів структури showroomData:

- vin (Vehicle Identification Number): Унікальний ідентифікатор транспортного засобу.

- ownerDetail: Деталі про власника транспортного засобу.

- modelDetail: Деталі моделі транспортного засобу.

- carCost: Вартість транспортного засобу.

- carLoanOrBankDetail: Інформація про позику або банківські деталі, пов'язані з придбанням автомобіля.

- showroomName: Назва авторизованого автосалону.

- showroomAddress: Адреса авторизованого автосалону.

- showroomId: Ідентифікатор автосалону, згенерований системою аутентифікації (наприклад, Firebase).

- issuedDate: Час запису даних.

- additionalDetail: Додаткова інформація, що може бути додана автосалоном.

б) Типи даних Solidity:

У таблиці нижче представлені основні типи даних, що використовуються в Solidity.

Рисунок 4.1 - Основні типи даних в Solidity

Назва	Примітки	Приклади
string	Послідовність символів	"Привіт!", "Шоколад"
bool	Булеве значення	true, false
int	Ціле число, позитивне або негативне. Не має десяткового	0, -30000, 59158
uint	Беззнакове' ціле число, позитивне. Не має десяткового	0, 30000, 999910
fixed/ufixed	Число з 'фіксованою' точкою. Число з десятковою крапкою	20.001, -42.4242, 3.14
address	Має методи, пов'язані з ним, для відправки грошей	0x18bae199c8dbae199c8d

в) Структура insuranceData:

Ця структура призначена для зберігання інформації, що надається страховими компаніями.

```
30 struct insuranceData {
31 bytes32 vin;
32 bytes32 insuranceNumber;
33 bytes32 numOfInsurance;
34 bytes32 insuranceDetail;
35 bytes32 insuranceId;
36 bytes32 insuranceCompanyName;
37 bytes32 insuranceCompanyAddress;
38 uint256 issuedDate;
39 bytes32 additionalDetail;
40 }
```

Рисунок 4.4 - Тип даних для зберігання даних страхування

Пояснення полів структури insuranceData:

- vin: VIN-номер транспортного засобу.

- insuranceNumber: Номер страхового поліса.

- numOfInsurance: Кількість страхових випадків/полісів, пов'язаних з ТЗ.

- insuranceDetail: Тип страхування.

- insuranceId: Ідентифікатор страхової події, згенерований Firebase.

- insuranceCompanyName: Назва авторизованої страхової компанії.

- insuranceCompanyAddress: Адреса авторизованої страхової компанії.

- issuedDate: Дата видачі полісу.

- additionalDetail: Додаткові деталі.

г) Структура motorDeptData:

Ця структура призначена для зберігання інформації від реєстраційних органів.

```
42 struct motorDeptData {
43 bytes32 vin;
44 bytes32 ownerName;
45 bytes32 regId;
46 bytes32 milesOnCar;
47 bytes32 modelDetail;
48 bytes32 numberPlate;
49 bytes32 motorDeptCity;
50 bytes32 motorDeptAddress;
51 uint256 issuedDate;
52 bytes32 additionalDetail;
53 }
```

Рисунок 4.5 - Тип даних для зберігання даних DMV

Пояснення полів структури motorDeptData:

- vin: VIN-номер транспортного засобу.

- ownerName: Ім'я власника транспортного засобу.

- regId: Ідентифікатор реєстрації транспортного засобу.

- milesOnCar: Пробіг транспортного засобу (в милях).

- modelDetail: Деталі моделі ТЗ (назва, рік, кількість дверей).

- numberPlate: Номерний знак ТЗ.

- motorDeptCity: Місто реєстраційного органу.

- motorDeptAddress: Адреса реєстраційного органу.

- issuedDate: Час збереження даних.

- additionalDetail: Додаткові деталі.

г) Структура mechanicData:

Ця структура призначена для зберігання інформації про виконані ремонтні роботи та обслуговування.

```
54 struct mechanicData {
55 bytes32 vin;
56 bytes32 shopId;
57 bytes32 shopName;
58 bytes32 shopAddress;
59 bytes32 carDetail;
60 bytes32 workDescription;
61 bytes32 moneyCharged;
62 bytes32 typeOfWork;
63 uint256 issuedDate;
64 bytes32 additionalDetail;
65 }
```

Рисунок 4.6 - Тип даних для зберігання даних механіка

Пояснення полів структури mechanicData:

- vin: VIN-номер транспортного засобу.
- shopId: Ідентифікатор механічної майстерні.
- shopName: Назва механічної майстерні.
- shopAddress: Адреса майстерні.
- carDetail: Деталі моделі ТЗ (назва, рік, кількість дверей).
- workDescription: Опис виконаних робіт.
- moneyCharged: Сума за ремонт.
- typeOfWork: Тип виконаних робіт (наприклад, заміна масла).
- issuedDate: Дата виконання робіт.
- additionalDetail: Додаткові деталі.

4.2. Реалізація функціональності розумного контракту

4.2.1. Функції контролю доступу

Ці функції призначені для управління дозволами на додавання даних у блокчейн. Їх може викликати лише власник контракту, що забезпечується модифікатором onlyOwner.

```
// Додавання функцій власника
function addShowroomOwner(address _addr) public onlyOwner {
    ShowroomOwners[_addr] = true;
}
function addInsuranceOwner(address _addr) public onlyOwner {
    InsuranceOwners[_addr] = true;
}
function addMotorDeptOwner(address _addr) public onlyOwner {
    MotorDeptOwners[_addr] = true;
}
function addMechanicOwner(address _addr) public onlyOwner {
    MechanicOwners[_addr] = true;
}
// Видалення функцій власника
function removeShowroomOwner(address _addr) public onlyOwner {
    ShowroomOwners[_addr] = false;
}
function removeInsuranceOwner(address _addr) public onlyOwner {
    InsuranceOwners[_addr] = false;
}
function removeMotorDeptOwner(address _addr) public onlyOwner {
    MotorDeptOwners[_addr] = false;
}
function removeMechanicOwner(address _addr) public onlyOwner {
    MechanicOwners[_addr] = false;
}
```

Рисунок 4.7 - Функція аутентифікації в розумному контракті

Ці функції поділяються на два типи:

1. Функції додавання прав (add...Owner).

Надають вказаній адресі Ethereum дозвіл на додавання даних відповідного типу. Це реалізується шляхом встановлення булевого значення true для цієї адреси у відповідному відображенні (mapping).

2. Функції видалення прав (remove...Owner).

Видаляють дозволи для певної адреси, встановлюючи булеве значення false у відображенні.

Модифікатор функції onlyOwner гарантує, що ці функції можуть бути викликані лише адресою, яка є власником контракту. Він виконується перед основною логікою функції та перевіряє право доступу. Параметром цих функцій є адреса Ethereum (_addr), якій надаються або відкликаються права.

```
// Збереження адрес творців даних [аутентифікація адміністратором]
mapping(address => bool) public ShowroomOwners;
mapping(address => bool) public InsuranceOwners;
mapping(address => bool) public MotorDeptOwners;
mapping(address => bool) public MechanicOwners;

// Додавання функцій власника
function addShowroomOwner(address _addr) public onlyOwner {
    ShowroomOwners[_addr] = true;
}
```

Рисунок 4.8 - Виклик відображення з функції (авторизація адміністратором)

```
// Видалення функцій власника
function removeShowroomOwner(address _addr) public onlyOwner {
    ShowroomOwners[_addr] = false;
}
```

Рисунок 4.9 - Виклик функції відображення для видалення аутентифікації адреси Ethereum

4.2.2. Функції додавання даних (для авторизованих організацій)

Ці функції використовуються авторизованими організаціями для додавання інформації до блокчейну Ethereum. Всі дані, що додаються, передаються як аргументи функції.

```
// Приклад функції додавання даних
function addMechanicData(
    bytes32 _vin,
    bytes32 _shopId,
    bytes32 _shopName,
    bytes32 _shopAddress,
    bytes32 _carDetail,
    bytes32 _workDescription,
    bytes32 _moneyCharged,
    bytes32 _typeOfWork,
    uint256 _issuedDate,
    bytes32 _additionalDetail
) public onlyMechanicOwner { // або інший відповідний модифікатор
    require(MechanicOwners[msg.sender] == true, "Caller is not an authorized mecha
    bytes32 dataHash = keccak256(
        _vin,
        _shopId,
        _shopName,
        _shopAddress,
        _carDetail,
        _workDescription,
        _moneyCharged,
        _typeOfWork,
        _issuedDate,
        _additionalDetail
```

Рисунок 4.10 - Функція, яку буде використовувати авторизована організація (початок)

```
    ); // Створення унікального хешу
    idToMechanicData[dataHash] = mechanicData(
        _vin,
        _shopId,
        _shopName,
        _shopAddress,
        _carDetail,
        _workDescription,
        _moneyCharged,
        _typeOfWork,
        _issuedDate,
        _additionalDetail
    ); // Призначення даних хешу
    vinToMechanicDataId[_vin].push(dataHash); // Зберігання хешу в масиві, пов'язаном
}
```

Рисунок 4.10 - Функція, яку буде використовувати авторизована організація (закінчення)

Кожна така функція починається з перевірки авторизації за допомогою функції require, яка гарантує, що лише авторизована організація може викликати цю функцію (наприклад, onlyMechanicOwner для даних механіка). Далі, створюється унікальний хеш з наданої інформації за допомогою функції keccak256. Цей хеш використовується як ключ у відображенні для зберігання структури даних. Крім того, хеш додається до масиву, індексованого за VIN-номером (vinToMechanicDataId), що дозволяє згодом отримати всі записи, пов'язані з конкретним транспортним засобом.

4.2.3. Функції отримання даних (для користувачів)

Ці функції дозволяють кінцевим користувачам отримувати інформацію з блокчейну Ethereum. У даному контракті реалізовано вісім таких функцій (по дві для кожної з чотирьох організацій), що зумовлено обмеженням на кількість значень, які можуть бути повернуті однією функцією в Solidity.

Розглянемо функцію getMechanicDataByVin1 як приклад. Вона приймає VIN-номер (_vin) як аргумент. Функція спочатку отримує масив ідентифікаторів даних (mechanicDataIds), пов'язаних з цим VIN-номером. Оскільки дані зберігаються у вигляді масивів хешів, кожен хеш вказує на унікальний запис даних через відображення (mapping).

```
function getMechanicDataByVin1(bytes32 _vin) public view
returns (bytes32[] memory vin, bytes32[] memory shopId, bytes32[] memory shopName,
    bytes32[] storage mechanicDataIds = vinToMechanicDataId[_vin]; // Отримання маси
    vin = new bytes32[](mechanicDataIds.length);
    shopId = new bytes32[](mechanicDataIds.length);
    shopName = new bytes32[](mechanicDataIds.length);
    shopAddress = new bytes32[](mechanicDataIds.length);
    carDetail = new bytes32[](mechanicDataIds.length);
    for (uint i = 0; i < mechanicDataIds.length; i++) {
        mechanicData storage data = idToMechanicData[mechanicDataIds[i]]; // Отрима
        vin[i] = data.vin;
        shopId[i] = data.shopId;
        shopName[i] = data.shopName;
        shopAddress[i] = data.shopAddress;
        carDetail[i] = data.carDetail;
    }
    return (vin, shopId, shopName, shopAddress, carDetail);
}
```

Рисунок 4.11 - Функція, яку викликатимуть користувачі

Далі, за допомогою циклу for, функція ітерує по кожному ідентифікатору, витягує відповідну структуру даних та зберігає її поля в окремі масиви, які потім повертаються як результат функції. Такий підхід забезпечує ефективне отримання всіх історичних даних, пов'язаних з конкретним транспортним засобом.

4.3. Імплементація сервісів Firebase для аутентифікації та бази даних у веб-додатку

Цей підрозділ присвячений детальному опису інтеграції сервісів Firebase — зокрема, функціоналу аутентифікації та бази даних реального часу — у розроблений веб-додаток. Загальний огляд Firebase вже був представлений у другому розділі. У цьому розділі зосереджено увагу на аспектах його практичного застосування.

4.3.1. Конфігурація Firebase у проекті

Першим кроком для інтеграції Firebase є його конфігурація у додатку. Цей процес передбачає створення нового проекту на веб-сайті Firebase, після чого надаються необхідні конфігураційні облікові дані. Ці дані, що

включають apiKey, authDomain, databaseURL, projectId, storageBucket та messagingSenderId, є унікальними для кожного проекту Firebase.

```
import firebase from "firebase/app";
import "firebase/database";
import "firebase/auth";

// Облікові дані, отримані з Firebase
var config = {
    apiKey: "AIzaSyB_qJZKjxYfJceOQzzUFUDEfenbKAASLOg",
    authDomain: "vehical-information-syatem.firebaseapp.com",
    databaseURL: "https://vehical-information-syatem.firebase.com",
    projectId: "vehical-information-syatem",
    storageBucket: "vehical-information-syatem.app.com",
    messagingSenderId: "820256655207"
};
firebase.initializeApp(config);
// Експорт екземпляра бази даних реального часу Firebase
export const db = firebase.database();
// Експорт екземпляра сервісу аутентифікації Firebase
export const auth = firebase.auth();
```

Рисунок 4.12 - Файл конфігурації Firebase

Як видно з наведеного лістингу, відбувається імпорт необхідних модулів Firebase (firebase/app, firebase/database, firebase/auth). Після ініціалізації додатку за допомогою об'єкта config, експортуються екземпляри сервісів бази даних (db) та аутентифікації (auth), що дозволяє використовувати їхні методи в інших частинах кодової бази.

4.3.2. Використання сервісу аутентифікації Firebase

У цьому проекті для аутентифікації користувачів реалізовано механізм на основі електронної пошти та пароля. Firebase Authentication підтримує різні методи аутентифікації, однак для поточного рішення обрано автентифікацію за допомогою електронної пошти.

Використані методи з інтерфейсу сервісу аутентифікації Firebase включають: createUserWithEmailAndPassword, sendPasswordResetEmail, signOut та signInWithEmailAndPassword.

а) Метод createUserWithEmailAndPassword

Цей метод застосовується в адміністративному інтерфейсі для створення облікових записів для різних організацій. Адміністратор використовує його для реєстрації нових користувачів у системі.

```
auth.createUserWithEmailAndPassword(this.state.email, this.state.password)
    .then((res) => {
        this.submitTransaction(); // Виклик функції для подачі транзакції (можливо, до
        let uid = auth.currentUser.uid; // Отримання унікального ідентифікатора корист
        console.log('work');
        let obj = this.state;
        obj.address = obj.street + ", " + obj.city + ", " + obj.states + ", " + obj
        delete obj.street;
        delete obj.city;
        delete obj.states;
        delete obj.country;
        delete obj.zip;
        console.log("obj.address", obj.address);
        obj.type = 'insurance'; // Призначення типу користувача (наприклад, "страхова"
        db.ref('/users/' + uid).set(obj) // Збереження даних користувача в Realtime D
            .then((res) => {
                // Успішне збереження даних
            })
            .catch(err => alert(err.message)); // Обробка помилок збереження даних
    })
    .catch(err => {
        alert(err.message); // Обробка помилок створення користувача
    });
```

Рисунок 4.13 - Метод createUserWithEmailAndPassword

Для успішного виконання цього методу необхідно передати адресу електронної пошти та пароль як аргументи. У разі успішного створення користувача виконується блок then(), який, окрім внутрішньої логіки (такої як виклик submitTransaction та обробка даних користувача), зберігає додаткову інформацію про новостворeного користувача (наприклад, адресу та тип організації) у Firebase Realtime Database за унікальним ідентифікатором користувача (UID). У випадку помилки, виконується блок catch(), що відображає повідомлення про помилку.

б) Метод sendPasswordResetEmail

Цей метод використовується для функціональності скидання пароля. У разі, якщо користувач забув свій пароль, система дозволяє йому запросити

скидання, надсилаючи електронний лист з інструкціями на зареєстровану адресу.

```
auth.sendPasswordResetEmail(this.props.email)
    .then(function() {
        // Електронний лист надіслано.
        alert("Електронний лист для скидання пароля надіслано");
    })
    .catch(function(error) {
        // Сталася помилка.
    });
```

Рисунок 4.14 - Метод sendPasswordResetEmail

в) Метод signOut

Метод signOut призначений для виходу користувача з веб-додатку. Ця функція доступна як для адміністратора, так і для представників організацій.

```
auth.signOut()
    .then(function () {
        that.props.logout(); // Виклик функції для обробки виходу
        console.log("успішно"); // Вивід повідомлення про успіх
        history.push('/'); // Перенаправлення на головну сторінку
        // Вихід з системи успішний.
    })
    .catch(function (error) {
        // Сталася помилка
    });
```

Рисунок 4.15 - Метод signOut

г) Метод signInWithEmailAndPassword

Цей метод використовується для входу користувачів (адміністраторів та представників організацій) у систему.

Метод приймає електронну пошту та пароль користувача. У разі успішного входу, виконується блок then(). Тут додатково перевіряються права доступу користувача (наприклад, чи є він адміністратором), порівнюючи його UID з UID адміністратора, що зберігається в базі даних Firebase.

```
auth.signInWithEmailAndPassword(this.state.email, this.state.password)
    .then(res => {
        let user_uid = res.user.uid;
        db.ref('/admin').once("value", snapshot => {
            if (user_uid == snapshot.val().uid) {
                this.props.signedIn(); // Виклик функції для перенаправлення авторизов
            } else {
                this.setState({
                    errorTextPass: "цей користувач не має прав адміністратора." // Пов
                });
            }
        });
    })
    .catch(err => {
        console.log("error", err);
        // Деталізована обробка помилок входу
        if (err.message.includes("email")) {
            this.setState({ errorTextEmail: err.message });
        } else if (err.message.includes("password")) {
            this.setState({ errorTextPass: err.message });
        } else {
            this.setState({ error: err.message });
        }
    });
```

Рисунок 4.16 - Метод signInWithEmailAndPassword

Обробка помилок здійснюється у блоці catch(), де надаються специфічні повідомлення залежно від типу помилки (неправильна електронна пошта або пароль).

4.3.3. Використання сервісу бази даних реального часу Firebase

Сервіс Firebase Realtime Database використовується для зберігання додаткової інформації про користувачів та їхні ролі, наданої адміністрацією.

а) Метод once

```
db.ref('/admin').once("value", snapshot => {
    if (user_uid == snapshot.val().uid) {
        this.props.signedIn();
    } else {
        this.setState({
            errorTextPass: "цей користувач не має прав адміністратора."
        });
    }
});
```

Рисунок 4.17 - Метод once з сервісу бази даних Firebase

Метод once використовується для одноразового отримання даних з бази даних за вказаним шляхом. У наведеному прикладі він застосовується для отримання UID адміністратора з шляху /admin для перевірки прав доступу під час входу.

б) Метод set

Метод set використовується для запису даних за вказаним шляхом у базі даних.

```
obj.type = 'department'; // Призначення типу об'єкта (наприклад, "відділ")
db.ref('/users/' + uid).set(obj) // Збереження об'єкта за шляхом '/users/' + uid
    .then((res) => {
        // Успішне завершення операції
    })
    .catch(err => alert(err.message)); // Обробка помилки
```

Рисунок 4.18 - Метод set з сервісу бази даних Firebase

Він перезаписує дані за цим шляхом, якщо вони вже існують. У даному контексті, set використовується для збереження інформації про новостворених користувачів (їхні адреси, тип організації тощо) під їхнім унікальним UID.

РОЗДІЛ 5. РОЗРОБКА ІНТЕРФЕЙСУ КОРИСТУВАЧА ІНФОРМАЦІЙНОЇ СИСТЕМИ

Цей розділ документу представляє візуальні матеріали, що ілюструють інтерфейс користувача розробленої інформаційної системи. Наведені скріншоти демонструють ключові екрани та функціональні можливості системи, розділені за ролями користувачів: адміністратор, організації та кінцевий споживач.

5.1. Візуалізація інтерфейсу входу в систему

Наступні зображення ілюструють сторінки входу, призначені для різних категорій користувачів, забезпечуючи контрольований доступ до системи.

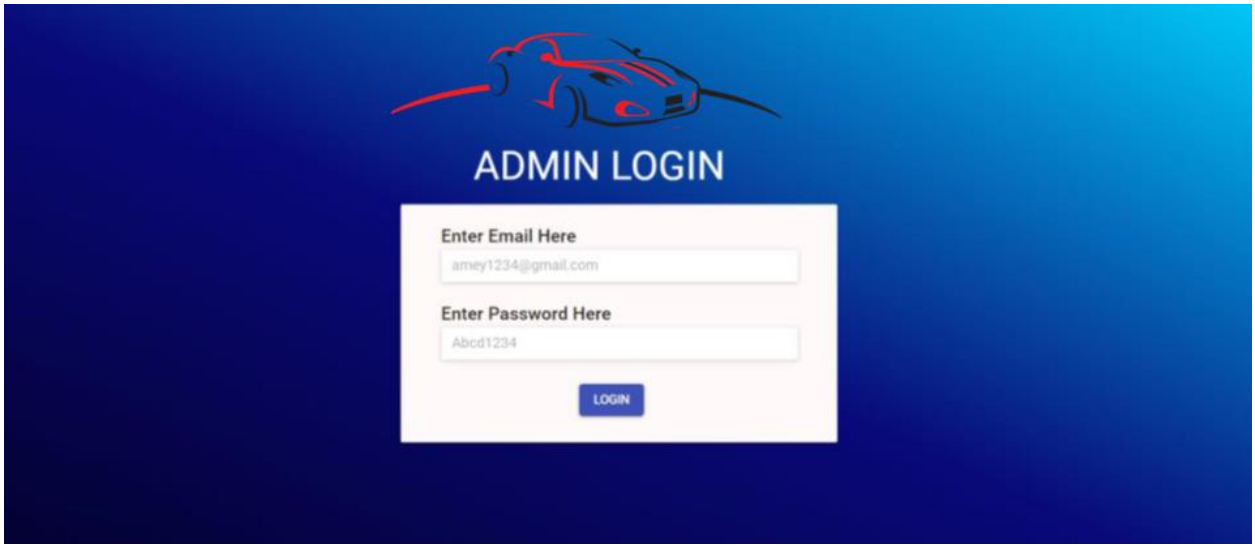

Рисунок 5.1 - Сторінка входу адміністратора

У верхній частині сторінки (рис. 5.1) розміщено стилізоване зображення автомобіля, що є графічним елементом, логотипом, що асоціюється з темою системи відстеження транспортних засобів. Під ним

розташований великий текстовий заголовок "ADMIN LOGIN", чітко вказуючи на призначення цієї сторінки.

Інтерфейс, призначений виключно для адміністраторів системи, які повинні ввести свої облікові дані (логін та пароль) для отримання доступу до адміністративної панелі управління.

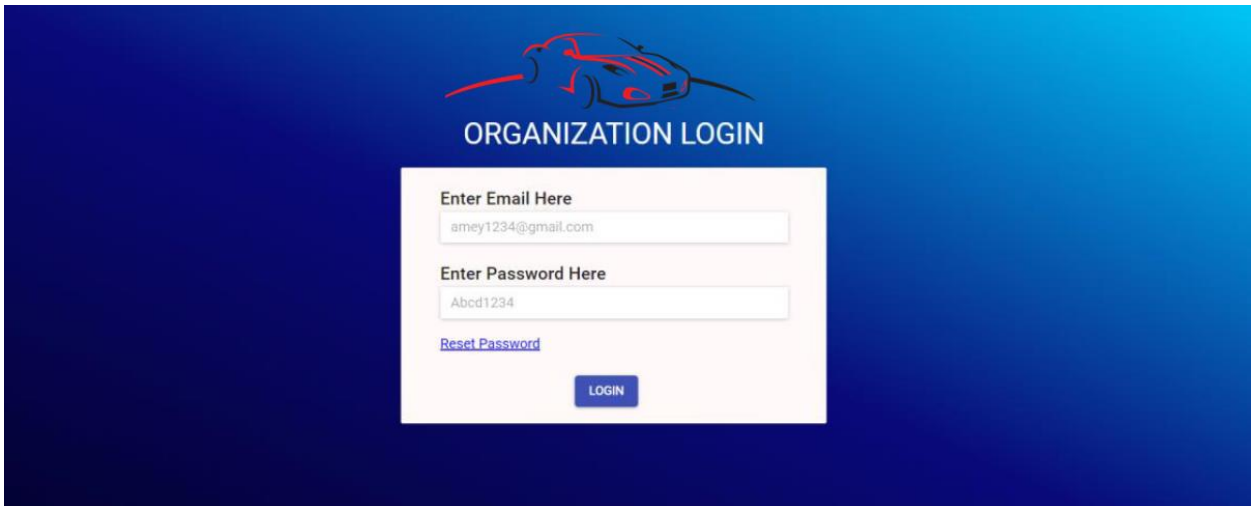

Рисунок 5.2 - Сторінка входу організації

5.2. Інтерфейс адміністратора системи

Представлені скріншоти демонструють функціональні розділи адміністративного проекту, що дозволяють власнику системи керувати даними та доступами для різних організацій.

На рисунку 5.3 показано розділ автосалону, що доступно адміністратору. Ліворуч показано меню з наступними елементами:

- "Insurance" (Страхування),
- "Department Of Motor Vehicle" (Департамент реєстрації ТЗ),
- "Mechanics" (Механіка),
- "All Users" (Усі користувачі),
- "Logout" (Вийти).

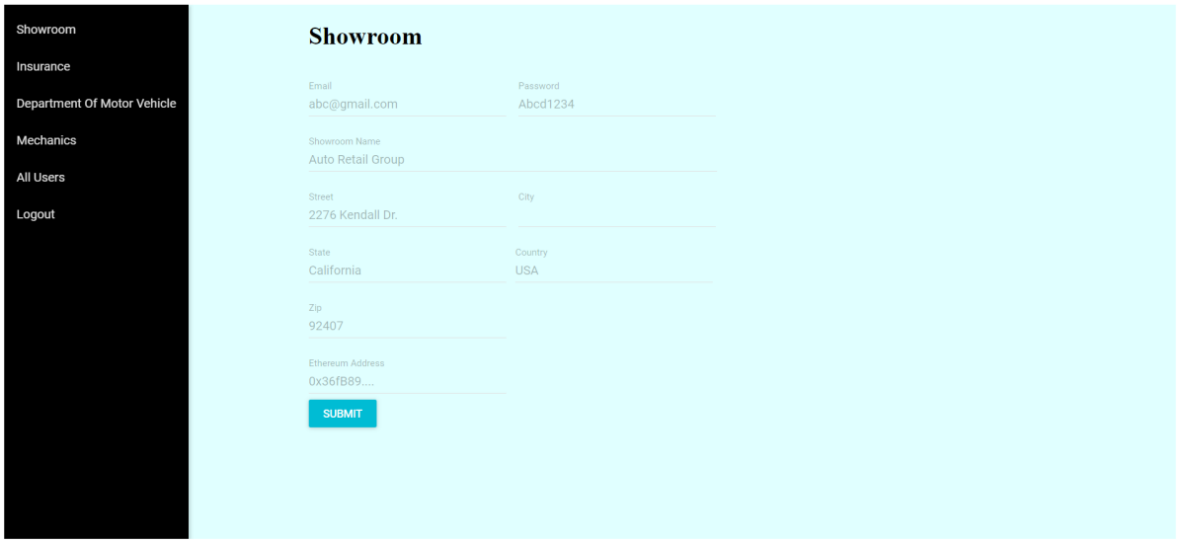

Рисунок 5.3 - Розділ автосалону (для адміністратора)

Поля для заповнення:

- "Showroom Name" - Назва автосалону,
- "Street" – Вулиця,
- "City" – Місто,
- "State" - Штат.
- "Country" – Країна,
- "Zip" - Поштовий індекс.
- "Ethereum Address" - адреса Ethereum, що асоціюється з цим автосалоном для взаємодії зі смарт-контрактом.

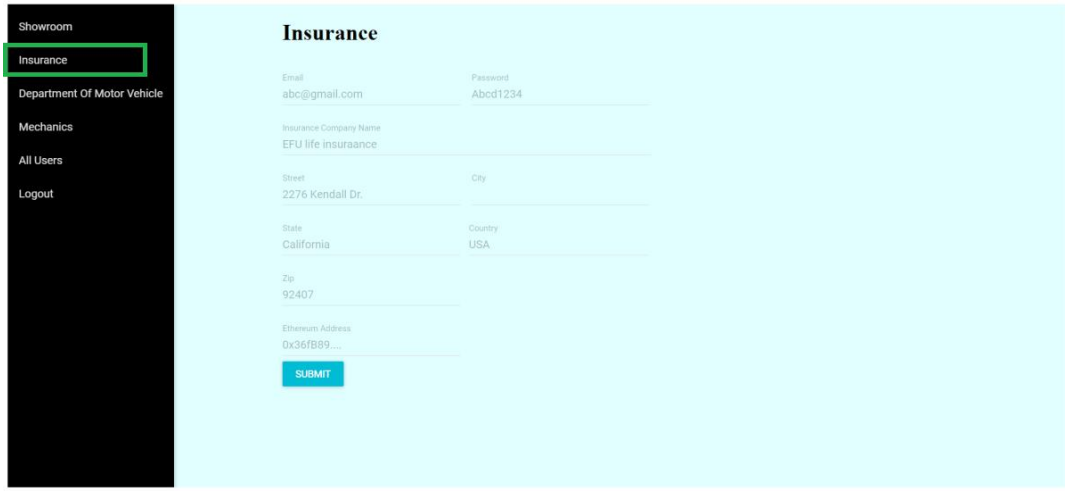

Рисунок 5.4 - Розділ страхування (для адміністратора)

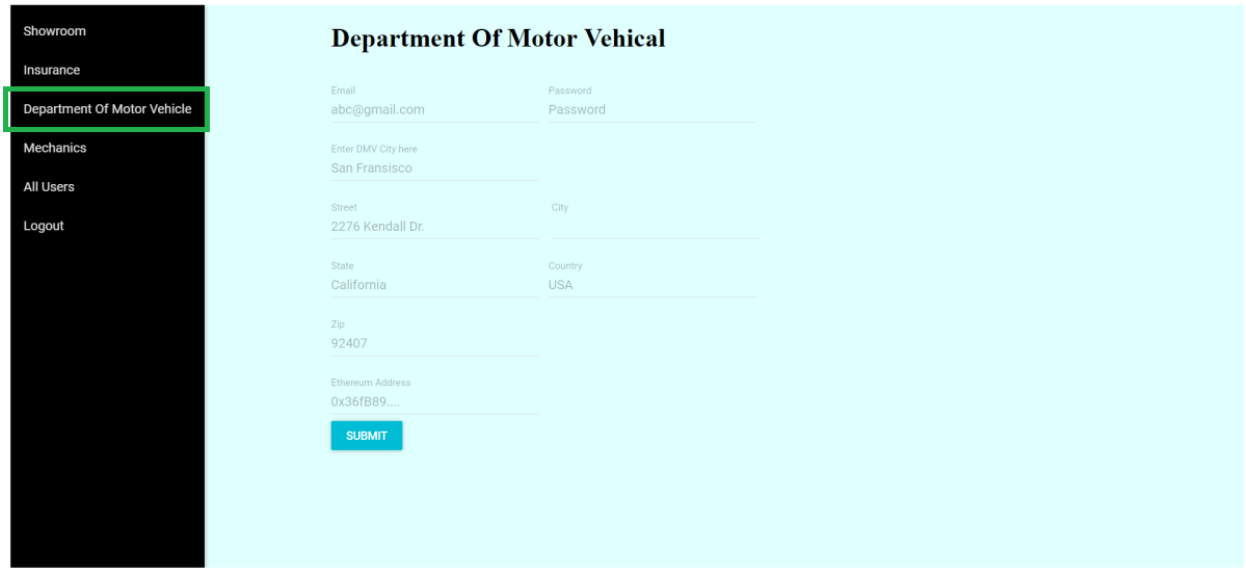

Рисунок 5.4 - Розділ DMW (для адміністратора)

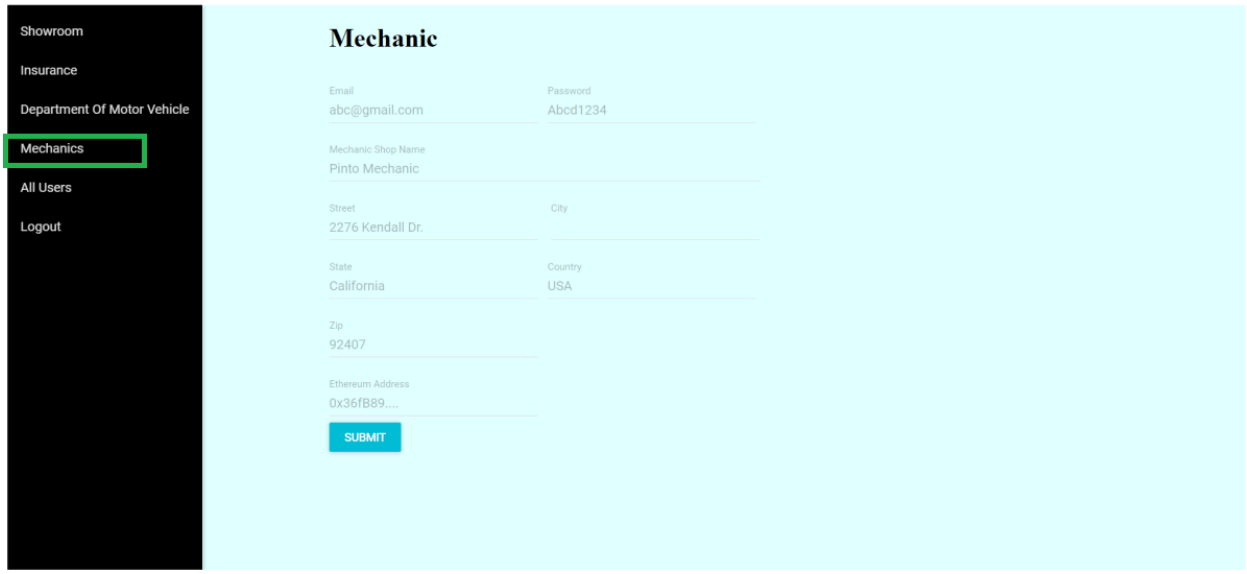

Рисунок 5.5 - Розділ “Mechanic” (для адміністратора)

На рисунку 5.7 показано розділ "All Users" в адміністративній панелі управління системою. Цей розділ призначений для перегляду та управління списком усіх зареєстрованих користувачів/організацій.

Сторінка представляє список користувачів і має п'ять стовпців:

1. Ethereum Address - адреса гаманця Ethereum, пов'язана з кожним користувачем/організацією. Це критично важливе поле, що підкреслює інтеграцію з блокчейном.

Рисунок 5.7 - Розділ всіх деталей користувача

2. Email - адреса електронної пошти користувача/організації.

3. Organization Name - назва організації, до якої належить користувач.

4. Organization Type - тип організації (наприклад, "showroom", "department", "mechanic", "insurance"). Це вказує на роль користувача в системі.

4. Кнопка "Delete" дозволяє адміністратору видаляти відповідний запис користувача.

5.3. Представлення інтерфейсів для організацій

Нижче наведено скріншоти інтерфейсів, призначених для автентифікованих організацій, які відповідають за внесення та управління специфічними даними про транспортні засоби.

На рисунку 5.8 представлено сторінку додавання інформації про транспортний засіб, для страхової компанії. Праворуч зверху відображається ID користувача (наприклад, "ID: 0ayIMgOk3BVmeHzQvUJrZPhIC4p1"), а поруч з ним — кнопка "LOGOUT" (Вийти). Це вказує на те, що користувач авторизований і використовується блокчейн.

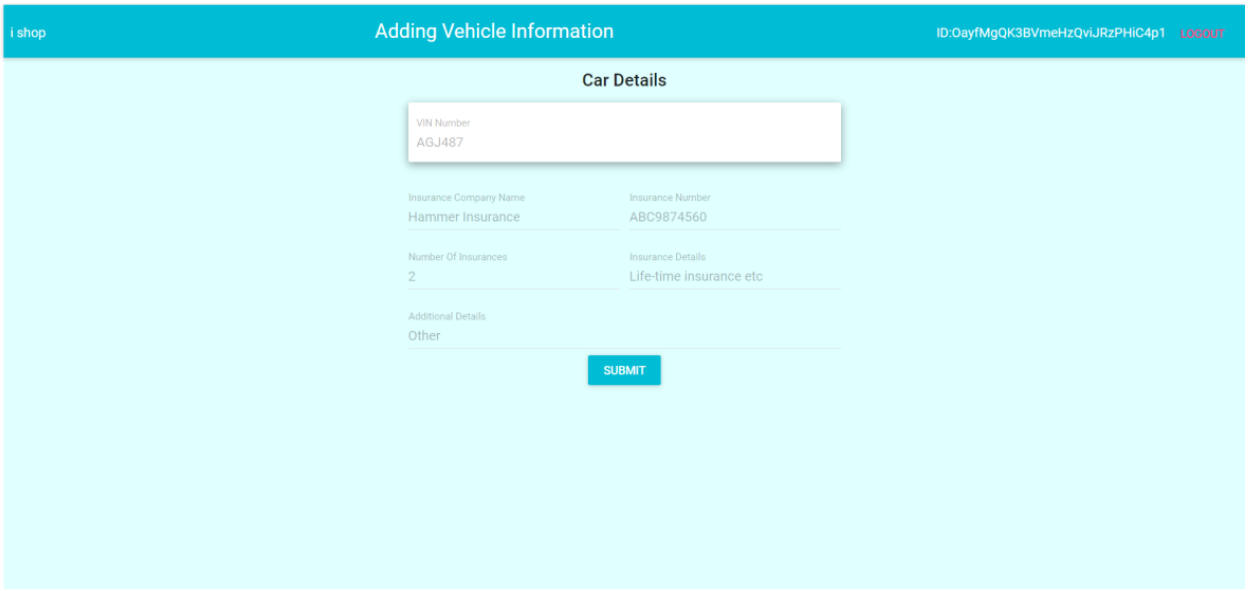

Рисунок 5.8 - Сторінка додавання інформації про транспортний засіб, для страхової компанії

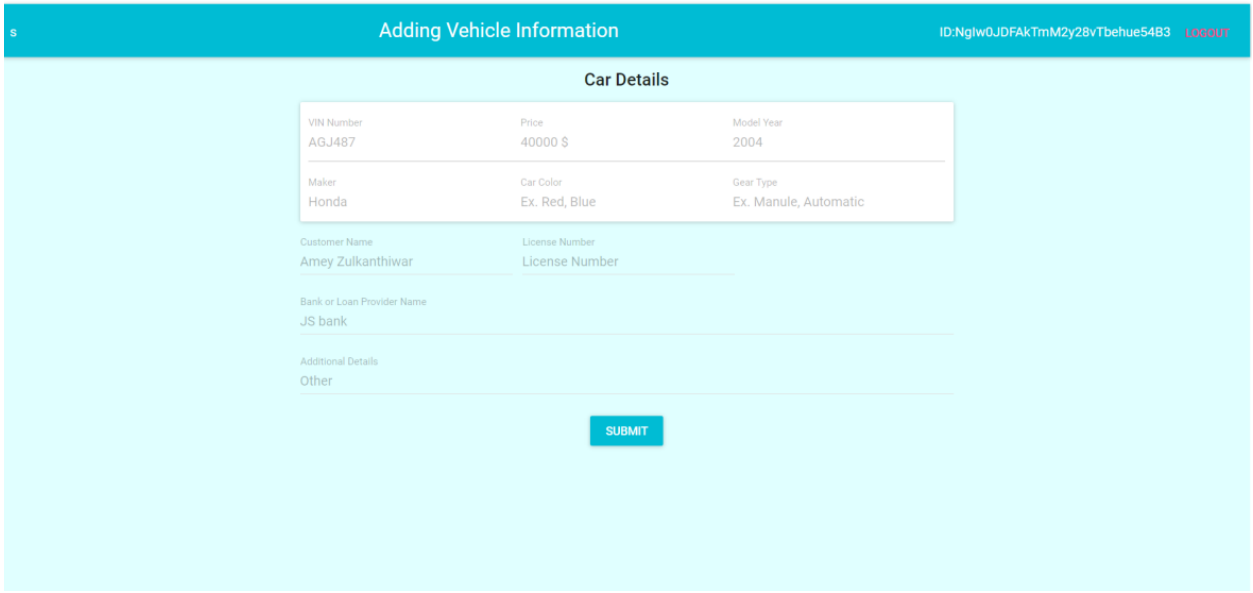

Рисунок 5.9 - Сторінка додавання інформації про транспортний засіб, для організацій (автосалонів)

На рисунку 5.10 представлена сторінка додавання інформації про транспортний засіб, для органів реєстрації.

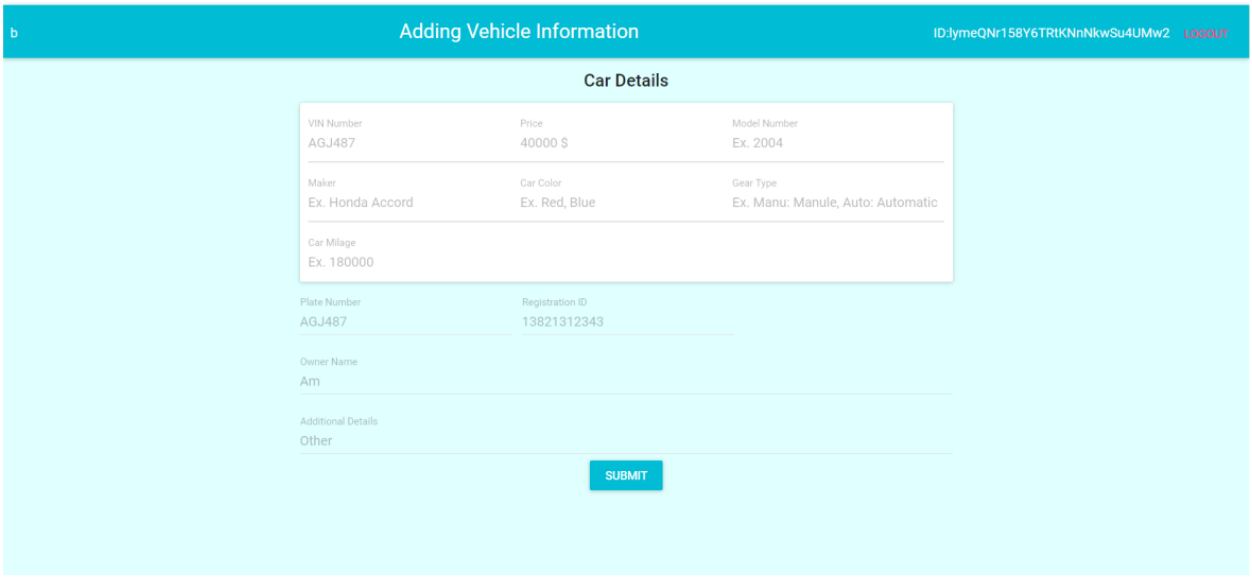

Рисунок 5.10 - Сторінка додавання інформації про транспортний засіб, для сервісних центрів реєстрації

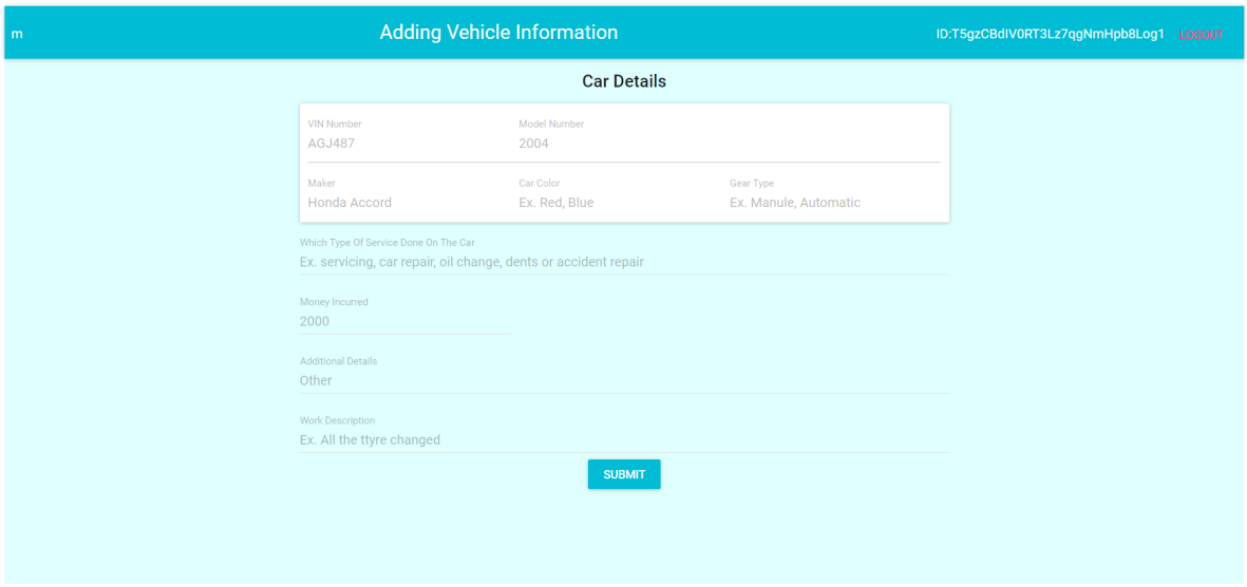

Рисунок 5.11 - Сторінка додавання інформації про транспортний засіб, для СТО

5.4. Реалізація інтерфейсу кінцевого користувача системи

Завершує візуалізацію інтерфейс, призначений для кінцевих користувачів, що дозволяє їм переглядати агреговану інформацію про транспортні засоби.

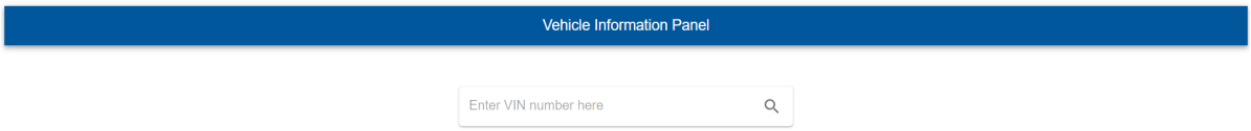

Рисунок 5.12 – Сторінка користувача для введення VIN коду автомобіля

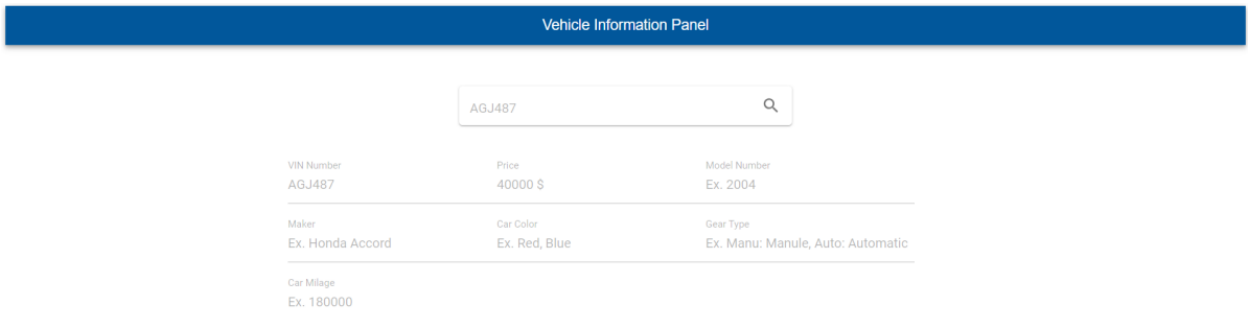

Рисунок 5.13 – Сторінка користувача з інформацією про автомобіль

У поточній імплементації веб-додатку адміністратор має можливість відкликати авторизацію для будь-якої організації зі смарт-контракту та видалити відповідні записи з Firebase Realtime Database. Однак, функціональність видалення облікових даних для входу безпосередньо з Firebase не передбачена на адміністративній стороні веб-додатку. Видалення аутентифікаційних даних можливе або через консоль управління Firebase, або вимагає, щоб сам користувач ініціював процес видалення свого облікового запису після входу до системи. Firebase не надає програмних інтерфейсів (методів) для видалення аутентифікаційних записів на рівні адміністративного інтерфейсу веб-додатку.

У межах даного проєкту було успішно розроблено веб-додаток, функціонал якого забезпечує запис даних до блокчейну Ethereum. Ключовою

особливістю реалізації є неможливість подальшого оновлення або зміни збережених даних, що повністю відповідає фундаментальним принципам блокчейну та визначеній меті проєкту щодо забезпечення незмінності інформації.

Розроблений веб-додаток інтегрований із системою входу, що дозволяє взаємодіяти з блокчейном. Проєкт успішно поєднав централізовані компоненти з децентралізованою та розподіленою архітектурою. Зокрема, реалізовано механізм подвійної аутентифікації, що включає використання Firebase для управління доступом та Metamask для взаємодії з блокчейном. Цей підхід значно підвищує рівень безпеки веб-додатку та гарантує цілісність даних, що записуються у блокчейн.

ВИСНОВКИ

В дипломній роботі було досліджено, спроєктовано та реалізовано прототип інформаційної системи відслідковування транспортних засобів із використанням технологій блокчейну. Проведене дослідження дозволило обґрунтувати доцільність застосування розподілених реєстрів для забезпечення прозорості, достовірності та цілісності інформації, що циркулює в сфері реєстрації та обміну даними про транспортні засоби.

У першому розділі здійснено всебічний аналіз предметної області. Було охарактеризовано функціональність інформаційної системи на основі блокчейну та актуальність її розробки. Визначено головні проблеми в існуючих підходах, зокрема — ризики фальсифікації даних, недовіра до онлайн-оголошень, труднощі перевірки історії транспортного засобу тощо. Розглянуто архітектуру блокчейн-системи, зокрема принципи функціонування платформи Ethereum, специфіку створення смарт-контрактів мовою Solidity та взаємодію з фронтендом через Web3.js. Проведено аналіз провідних українських інформаційних ресурсів (зокрема RST.ua та Auto.ria), що дозволило виявити функціональні обмеження централізованих рішень.

У другому розділі сформульовано функціональні й нефункціональні вимоги до системи, обґрунтовано вибір інструментів розробки — MetaMask, Firebase, Infura, Node.js та інші. Здійснено деталізацію технічних та програмних компонентів, які забезпечують коректне функціонування розподіленої системи обміну інформацією.

У третьому розділі змодельовано процеси взаємодії користувачів із системою. За допомогою діаграм варіантів використання представлено основні сценарії роботи: реєстрація та аутентифікація, взаємодія з розумними контрактами, перегляд інформації про транспортні засоби, внесення змін до бази даних тощо. Також представлено архітектурну діаграму системи, яка відображає логічну структуру її компонентів.

У четвертому розділі описано етапи програмної реалізації. Створено смарт-контракти на мові Solidity, які реалізують основні функції контролю доступу, додавання інформації про транспортні засоби та її перевірки. Впроваджено модулі для взаємодії з Firebase, що забезпечує реєстрацію та аутентифікацію користувачів, а також зберігання додаткових метаданих у базі даних реального часу.

У п’ятому розділі розроблено та реалізовано інтерфейси користувача для трьох основних ролей: адміністратора, представників організацій і кінцевих користувачів. Інтерфейс забезпечує інтуїтивну взаємодію з системою, включаючи функції перегляду, додавання та редагування даних, доступ до перевіреної інформації, яка зберігається у блокчейні.

Таким чином, у результаті дослідження було створено інформаційну систему, що поєднує можливості блокчейн-технології та традиційної веб-архітектури для вирішення актуальної проблеми — забезпечення достовірності інформації про транспортні засоби. Реалізоване рішення демонструє перспективність використання розподілених реєстрів у сфері транспорту, зокрема для боротьби з шахрайством, підвищення рівня довіри між продавцями й покупцями, а також удосконалення державного обліку автотранспорту.

Отримані результати можуть бути використані як основа для подальшої розробки повнофункціональної платформи, а також розширення системи на інші галузі, де актуальне питання прозорого обліку об'єктів із фіксацією історії змін.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. How can I increase or decrease the value when put to a database Firebase RD, javascript - Stack Overflow - https://stackoverflow.com/questions/77121771/how-can-i-increase-or-decrease-the-value-when-put-to-a-database-firebase-rd-jav
2. Remix | Viction - https://docs.viction.xyz/smart-contract-development/ides-and-tools/remix
3. Cebe, M., Erdin, E., Akkaya, K., Aksu, H., & Uluagac, S. (2018). Block4Forensic: An Integrated Lightweight Blockchain Framework for Forensics Applications of Connected Vehicles. arXiv preprint arXiv:1802.00561.
4. Bragadeesh, S. A., & Umamakeswari, A. (2022). Secured Vehicle Life Cycle Tracking using Blockchain and Smart Contract. Computer Systems Science and Engineering, 41(1), 1–18. Tech Science Press, Oct 8.
5. Chauhan, H., Kumar, D., Gupta, D., Gupta, S., & Verma, V. (2021). Blockchain and IoT based Vehicle Tracking System for Industry 4.0 Applications. IOP Conference Series: Materials Science and Engineering, 1022(1), 012051.
6. Alharby, M., & van Moorsel, A. (2017). Blockchain-based Smart Contracts: A Systematic Mapping Study. arXiv preprint arXiv:1710.06372.
7. Li, X., Jiang, P., Chen, T., Luo, X., & Wen, Q. (2018). A Survey on the Security of Blockchain Systems. arXiv preprint arXiv:1802.06993.
8. Lukman, A. (2019). A Secure Tracking Automobile System for Oil and Gas Distribution using Telematics and Blockchain Techniques. Balkan Journal of Electrical and Computer Engineering, 7(3), 257–268.
9. Tomar, R. (2022). A Blockchain-Based Approach to Track Traffic Messages in Vehicular Networks. In Proceedings of Academia-Industry Consortium for Data Science (pp. 345–362).

10. Xing, W., Cheng, X., Li, J., He, Y., & Xiao, K. (2021). A Survey: Applications of Blockchain in the Internet of Vehicles. EURASIP Journal on Wireless Communications and Networking, 2021(77).
11. Reimers, T., Leber, F., & Lechner, U. (2019). Integration of Blockchain and Internet of Things in a Car Supply Chain. In Proceedings of the 2019 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPCON) (pp. 146–151).
12. González-Benito, Ó., et al. (2020). Blockchain Technology for Enhancing Traceability and Efficiency in Automobile Supply Chain — A Case Study. Sustainability, 13(24), 13667.
13. Mollah, M. B., Zhao, J., Niyato, D., Guan, Y. L., Yuen, C., Sun, S., & Lam, K.-Y. (2020). Blockchain for the Internet of Vehicles towards Intelligent Transportation Systems: A Survey. IEEE Internet of Things Journal, 7(5), 4985–5000.
14. Hildebrand, B., Baza, M., Salman, T., Amsaad, F., Razaqu, A., & Alourani, A. (2022). A Comprehensive Review on Blockchains for Internet of Vehicles: Challenges and Directions. IEEE Access, 10, 23456–23489.
15. Fraga-Lamas, P., & Fernandez-Carames, T. M. (2024). A Review on Blockchain Technologies for an Advanced and Cyber-Resilient Automotive Industry. IEEE Transactions on Intelligent Transportation Systems, 25(4), 1234–1256.
16. Li, W., Nejad, M., & Zhang, R. (2019). A Blockchain-Based Architecture for Traffic Signal Control Systems. In Proceedings of the International Conference on Blockchain Methods in Transportation (pp. 45–53). New York, USA: IEEE.
17. Chauhan, H., Kumar, D., Gupta, D., Gupta, S., & Verma, V. (2021). Blockchain and IoT based Vehicle Tracking System for Industry 4.0 Applications. IOP Conference Series: Materials Science and Engineering, 1022(1), 012051.

18. Aksu, H., Akkaya, K., Cebe, M., Erdin, E., & Uluagac, S. (2018). Block4Forensic: An Integrated Lightweight Blockchain Framework for Forensics Applications of Connected Vehicles. arXiv:1802.00561.
19. Lukman, A. (2019). A Secure Tracking Automobile System for Oil and Gas Distribution using Telematics and Blockchain Techniques. Balkan Journal of Electrical and Computer Engineering, 7(3), 257–268.
20. Reimers, T., Leber, F., & Lechner, U. (2019). Integration of Blockchain and Internet of Things in a Car Supply Chain. In Proceedings of IEEE International Conference on Decentralized Applications and Infrastructures (pp. 146–151). IEEE.
21. González-Benito, Ó., et al. (2020). Blockchain Technology for Enhancing Traceability and Efficiency in Automobile Supply Chain — A Case Study. Sustainability, 13(24), 13667.
22. Tomar, R. (2022). A Blockchain-Based Approach to Track Traffic Messages in Vehicular Networks. In Proceedings of the Academia-Industry Consortium for Data Science (pp. 345–362).
23. Ferrag, M. A., Shu, L., & Maglaras, L. (2020). Blockchain for Vehicular Internet of Things: Recent Advances and Open Issues. IEEE Access, 8, 148072–148094.
24. Ali, O., & Wang, L. (2020). Blockchain Assisted Intelligent Transportation System Promoting Data Services with Privacy Protection. Sensors, 20(9), 2483.
25. Li, X., et al. (2019). Blockchain for Smart Mobility—Literature Review and Future Research Agenda. Sustainability, 13(23), 13268.
26. Awad, A. I., Al-Akkad, A. T., & Mohammed, F. (2020). Intelligent Transportation Using Wireless Sensor Networks and Blockchain. Sensors, 23(5), 2670.
27. Hilario, M., & Chen, Y. (2020). A Review of Blockchain-Based Systems in Transportation. Information, 11(1), 21.

28. Caro, M. P., Ali, M. S., Vecchio, M., & Giaffreda, R. (2018). Blockchain–based Ubiquitous Transport and Logistics Monitoring System. MDPI Proceedings, 31(1), 9–19.
29. Yuan, Y., & Wang, F.-Y. (2016). Towards Blockchain-Based Intelligent Transportation Systems. In Proceedings of the IEEE 19th International Conference on ITS (pp. 2663–2668). IEEE.
30. Maksymyuk, T., Gazda, J., Han, L., & Jo, M. (2019). Blockchain-Based Intelligent Network Management for 5G and Beyond. In 3rd Intl Conf on Advanced Information and Communications Tech. (pp. 36–39). IEEE.
31. Sharma, R., & Chakraborty, S. (2018). BlockApp: Blockchain for Authentication and Privacy Preservation in IoV. In IEEE Globecom Workshops (pp. 1–6). IEEE.
32. Deng, X., & Gao, T. (2020). Electronic Payment Schemes Based on Blockchain in VANETs. IEEE Access, 8, 38296–38303.
33. Olumolade, O., Jiang, H., & Forde, A. (2017). Smart Parking Meter for Internet of Vehicle (IoV). In 8th Annual UEMCON (pp. 112–118). IEEE.
34. Lin, X., Wu, J., Mumtaz, S., Garg, S., & Li, J. (2020). Blockchain-Based On-Demand Computing Resource Trading in IoV-Assisted Smart City. IEEE Transactions on Emerging Topics in Computing, 8, 1234–1245.

БІБЛІОГРАФІЧНА ДОВІДКА

Тема дипломної роботи: “ Інформаційна система відслідковування інформації про транспортні засоби ”

Обсяг пояснювальної записки: 77 аркушів.

Дата закінчення роботи: 9 червня 2025 р.

Підпис студента _____