Хмельницький національний університет
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

Програмна система відстеження гуманітарних вантажів у логістичних ланцюгах постачання
Назва теми

Рівень вищої освіти Перший (бакалаврський)

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Шифр КвРІПЗ.2101070.0103.ПЗ

Виконав студент IV курсу, група ІПЗ-21-1 _____ Олександр ГАЛАШЕВСЬКИЙ
Підпис Ім'я, ПРІЗВИЩЕ

Керівник старший викладач _____ Ганна БЕДРАТЮК
Науковий ступінь, вчене звання Підпис Ім'я, ПРІЗВИЩЕ

Нормоконтролер канд. пед. наук, доцент _____ Наталія ПРАВОРСЬКА
Посада Підпис Ім'я, ПРІЗВИЩЕ

До захисту допускаю:
Завідувач кафедри інженерії програмного забезпечення _____ Леонід БЕДРАТЮК
Підпис Ім'я, ПРІЗВИЩЕ

3 червня 2025 р.

Хмельницький 2025

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій
Кафедра Інженерії програмного забезпечення
Рівень вищої освіти Перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри ПІЗ
Л. П. Бедратюк
2 10 1 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Галашевський Олександр Юрійович

Прізвище, ім'я, по батькові студента

1. Тема роботи Програмна система відстеження гуманітарних вантажів у логістичних ланцюгах постачання

Керівник роботи Бедратюк Ганна Іванівна, старший викладач

Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 07.02.2025 р.
2. Строк подання студентом роботи на кафедру 01.06.2025 р.
3. Вихідні дані до роботи Матеріали переддипломної практики
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити) Дослідження предметної області, проєктування інформаційної системи, реалізація та тестування.
5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень) Графічні матеріали (6 шт.)

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Н.І. Праворська, канд. пед. наук, доцент	5.05.25	8.05.25
Антиплагіат	Форкун Ю. В., канд. техн. наук, доцент	12.05.2025	02.06.2025

7. Дата видачі завдання « 02 » січня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1. Ознайомлення з тематикою дипломного проектування, визначення та узгодження індивідуальних тем кваліфікаційних робіт (КвР)	01.12– 31.12.2024	
2. Збір матеріалу за темою КвР; дослідження предметної області, в якій планується використан- тання програмного забезпечення (ПЗ), визначення задач та вимог, розробка технічного завдання	01.01 – 20.02 2025	
3. Проектування програмного забезпечення	21.02 – 20.03 2025	
4. Програмна реалізація з використанням відповідних засобів розробки. Тестування ПЗ	21.03 – 30.04.2025	
5. Написання вступу, загальних висновків, оформлення переліку джерел посилання та додатків. Оформлення пояснювальної записки КвР згідно вимог	01.05 – 25.05.2025	
6. Попередній захист	Травень 2025	
7. Перевірка КвР на плагіат, нормоконтроль, отримання відгуків, рецензій та інших супровідних документів. Брошурування (зшиття) пояснювальної записки.	26.05 – 30.05 2025	
8. Здача КвР на кафедру: підготовка КвР для розміщення у репозитарії ІТУ; підготовка до захисту та захист КвР	з 1.06 2025	

Студент _____ Олександр ГАЛАШЕВСЬКИЙ
Підпис Ім'я, ПРІЗВИЩЕ

Керівник роботи _____ Ганна БЕДРАТЮК
Підпис Ім'я, ПРІЗВИЩЕ

АНОТАЦІЯ

Тема кваліфікаційної роботи: Програмна система відстеження гуманітарних вантажів у логістичних ланцюгах постачання

Автор проєкту: Галашевський Олександр Юрійович

Керівник проєкту: Бедратюк Ганна Іванівна.

Пояснювальна записка: 89 с., 15 рис., 3 табл., 4 дод., 38 джерел.

Графічна частина: 6 креслень формату А4.

БЛОКЧЕЙН, ЛОГІСТИКА, ГУМАНІТАРНА ДОПОМОГА, ПРОЗОРІСТЬ, ВІДСТЕЖЕННЯ ВАНТАЖІВ, КРИПТОГРАФІЧНИЙ ЗАХИСТ, РОЗПОДІЛЕНА КНИГА, СМАРТ-КОНТРАКТИ.

Мета кваліфікаційної роботи: розробка програмного рішення, заснованого на блокчейн-технології, для забезпечення більшої прозорості, надійності та оптимізації процесу відстеження гуманітарних вантажів у ланцюгах постачання.

У роботі проведено дослідження основних проблем у сфері гуманітарної логістики, таких як відсутність прозорості, корупційні ризики та неефективне управління ресурсами.

Для реалізації рішення було обрано платформу Ethereum, мову Solidity для розробки смарт-контрактів, а також відповідні інструменти для роботи з дистрибуційними даними.

Практичне значення роботи полягає в розробці інструменту, який сприяє покращенню прозорості та безпеки в гуманітарній логістиці. Це допомагає знизити ймовірність шахрайства, затримок і нецільового використання ресурсів, що особливо важливо в умовах кризових ситуацій, збройних конфліктів та міжнародних гуманітарних місій.

03.05.2025
Дата

Підпис

ЗМІСТ

Змн.	Арк.	№ докум.	Підпис	Дата	КвРІПЗ.2101070.0103.ПЗ		
Виконав		Галашевський О.Ю.		03.05.23	Програмна система відстеження гуманітарних вантажів у логістичних ланцюгах постачання	Літ.	Арк. 5 / Акрушів 70
Керівник		Бедратюк Г.І.		3.05.23			
Рецензент		Павловський О.		10.05.23			
Н. Контр.		Праворська Н. І.		9.05.23		ХНУ. ІПЗ-21-1	
Зав. каф.		Бедратюк Л.П.		5.05.23			

ВСТУП

Сучасний світ, на жаль, дедалі частіше стикається з викликами, що потребують негайного та скоординованого реагування. Гуманітарні кризи, спричинені природними лихами, збройними конфліктами чи пандеміями, ставлять під загрозу добробут та життя мільйонів людей. В таких умовах надзвичайно важливу роль відіграє ефективна логістика гуманітарної допомоги – процес, що забезпечує своєчасну доставку життєво необхідних ресурсів тим, хто їх найбільше потребує. Проте, традиційні підходи до управління гуманітарними вантажами часто стикаються з низкою серйозних проблем. Серед них – недостатня прозорість ланцюгів постачання, що ускладнює відстеження вантажів від донора до кінцевого отримувача.

Ця непрозорість, у свою чергу, створює підґрунтя для корупційних ризиків та нецільового використання допомоги. Неефективне управління ресурсами призводить до їх нераціонального розподілу, затримок у доставці та, як наслідок, до погіршення становища людей, які очікують на допомогу. Складність координації дій між численними учасниками гуманітарних місій – міжнародними організаціями, державними структурами, волонтерськими рухами та місцевими громадами – також суттєво знижує загальну ефективність зусиль.

Особливої гостроти ці проблеми набули в Україні в контексті повномасштабного вторгнення. Безпрецедентні обсяги гуманітарної допомоги, що надходять з усього світу, потребують надійних та прозорих механізмів обліку, розподілу та контролю. Існуючі системи часто не в змозі впоратися з таким навантаженням, що підкреслює нагальну потребу в інноваційних технологічних рішеннях. Саме в цьому контексті технологія блокчейн відкриває нові перспективи. Її ключові характеристики – децентралізація, незмінність записів, криптографічний захист та прозорість – дозволяють кардинально змінити підходи до управління ланцюгами постачання, зробивши їх більш надійними, підзвітними та ефективними.

Дана кваліфікаційна робота присвячена розробці програмної системи “Lanka”, призначеної для прозорого відстеження гуманітарних вантажів у логістичних ланцюгах постачання з використанням блокчейн-технологій. Метою роботи є створення інструменту, який дозволить підвищити ефективність гуманітарних операцій, зміцнити довіру донорів та забезпечити, щоб допомога досягала тих, хто її справді потребує.

Рисунок 1.1 – Проблеми традиційних ланцюгів постачання гуманітарної допомоги

Для досягнення цієї мети були поставлені наступні завдання:

– провести глибокий аналіз існуючих проблем у сфері гуманітарної логістики;

– дослідити сучасні технології, що застосовуються для відстеження поставок, з особливим акцентом на блокчейн-рішеннях; сформулювати чіткі вимоги до функціональності та характеристик системи “Lanka”;

– розробити її архітектуру, що поєднує переваги традиційних веб-технологій з можливостями блокчейну Ethereum та децентралізованого сховища IPFS;

– спроектувати відповідну структуру бази даних та розробити необхідні смарт-контракти;

– створити інтуїтивно зрозумілий та функціональний користувацький інтерфейс;

– реалізувати ключові модулі системи, що відповідають за управління користувачами, пакетами допомоги, документацією та аудитом;

– а також провести всебічне тестування розробленого програмного продукту та проаналізувати отримані результати.

Об'єктом даного дослідження є сам процес управління та відстеження гуманітарних вантажів у складних та динамічних логістичних ланцюгах. Предметом дослідження виступає застосування передових блокчейн-технологій, зокрема можливостей платформи Ethereum та системи децентралізованого зберігання даних IPFS, для створення програмної системи "Lanka", яка має на меті суттєво покращити прозорість, надійність та загальну ефективність логістики гуманітарної допомоги.

У ході виконання роботи були використані такі методи дослідження, як аналіз науково-технічної літератури та існуючих аналогічних рішень, системний аналіз предметної області, об'єктно-орієнтоване проектування, моделювання бізнес-процесів, розробка програмного забезпечення з використанням гнучких (agile) методологій, а також тестування та експериментальна перевірка працездатності створеної системи.

Наукова новизна даної роботи полягає у розробці специфічної гібридної архітектури для системи відстеження гуманітарних вантажів. Ця архітектура ефективно інтегрує переваги централізованих баз даних, що забезпечують оперативність обробки інформації, з можливостями блокчейн-технологій, які гарантують незмінність та прозорість ключових даних та аудиторського сліду.

Також елементом новизни є розробка унікального набору смарт-контрактів, оптимізованих саме для потреб гуманітарної логістики, та реалізація інтеграції з системою IPFS для безпечного децентралізованого зберігання супровідної документації.

Практичне значення отриманих результатів полягає у створенні функціонального прототипу програмної системи “Lanka”. Ця система може бути

взята на озброєння гуманітарними організаціями, волонтерськими ініціативами, державними установами та міжнародними донорами як ефективний інструмент для значного покращення процесів доставки, розподілу та контролю гуманітарної допомоги. Впровадження системи “Lanka” дозволить підвищити рівень прозорості, мінімізувати ризики шахрайства та нецільового використання цінних ресурсів, оптимізувати логістичні маршрути та, що найважливіше, зміцнити довіру між усіма учасниками гуманітарного процесу.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз проблем у сфері логістики гуманітарної допомоги

Сфера логістики гуманітарної допомоги, незважаючи на її надзвичайну соціальну значущість, історично стикається з низкою хронічних проблем, які суттєво перешкоджають її ефективному функціонуванню. Ці проблеми особливо загострюються в умовах масштабних криз, коли швидкість, точність та прозорість дій набувають критичного значення.

Однією з ключових та найбільш поширених проблем є системна відсутність прозорості в ланцюгах постачання. Гуманітарні вантажі часто проходять через руки багатьох посередників, починаючи від міжнародних донорських організацій та урядів, і закінчуючи місцевими неурядовими організаціями, волонтерськими групами та кінцевими бенефіціарами. На кожному з цих етапів інформація про вантаж, його стан, точне місцезнаходження та рух може бути втрачена, спотворена або навмисно прихована. Це створює так звані “чорні скриньки” всередині логістичного ланцюга, де відстежити долю допомоги стає практично неможливо. Донори, які надають значні фінансові та матеріальні ресурси, часто позбавлені можливості отримати достовірне підтвердження того, що їхня допомога була використана за призначенням та досягла тих, хто її найбільше потребував.

Ця непрозорість не лише знижує ефективність допомоги, але й підриває довіру до гуманітарних інституцій. В умовах, коли, наприклад, країна отримує допомогу з десятків різних джерел, як це відбувається в Україні, відсутність єдиної прозорої системи обліку може призвести до значних втрат та неефективного розподілу ресурсів.

Безпосередньо пов'язаною з непрозорістю є проблема високих корупційних ризиків. Там, де немає належного контролю та підзвітності, завжди виникає спокуса для недобросовісних осіб зловживати своїм становищем. Розкрадання гуманітарних вантажів, їх нецільове використання, перепродаж на “чорному ринку” – все це, на жаль, є реальністю в багатьох гуманітарних операціях по всьому

світу. Такі дії не лише завдають прямої шкоди тим, хто позбавляється життєво необхідної допомоги, але й кидають тінь на всю систему гуманітарної допомоги, демотивуючи донорів та волонтерів.

Ще одним значним викликом є неефективне управління наявними ресурсами. Через брак точної та оперативної інформації про реальні потреби на місцях, про наявність тих чи інших товарів на складах, про логістичні можливості (транспорт, шляхи сполучення), часто виникають дисбаланси. В одні регіони може надходити надлишкова кількість певних видів допомоги, які потім псуються або лежать без руху, тоді як в інших регіонах люди потерпають від гострого дефіциту саме цих ресурсів. Це призводить до марнотратства, неефективного використання обмежених транспортних та складських потужностей, а також до затримок у доставці допомоги туди, де вона найбільш критично необхідна.

Складність координації між різними учасниками гуманітарних операцій також є суттєвою перешкодою. Міжнародні організації, урядові агенції, великі та малі неурядові організації, тисячі волонтерських груп – всі вони можуть діяти паралельно, використовуючи власні методики обліку, власні канали зв'язку та власні пріоритети. Відсутність єдиної інформаційної платформи, стандартів обміну даними та чітких механізмів координації призводить до дублювання зусиль, неузгодженості дій, інформаційного хаосу та, як наслідок, до зниження загальної ефективності допомоги.

Наприклад, українські волонтерські організації, такі як “Повернись живим” чи благодійний фонд Сергія Притули, щодня стикаються з необхідністю обробляти величезні потоки інформації та координувати зусилля тисяч людей. Відсутність єдиних, стандартизованих інструментів значно ускладнює їхню титанічну роботу, вимагаючи додаткових зусиль для синхронізації та обміну даними.

Критичною проблемою, особливо в умовах надзвичайних ситуацій, що швидко розвиваються, є затримки в доставці гуманітарних вантажів. Ці затримки можуть бути спричинені цілою низкою факторів: бюрократичними процедурами на митниці, складними логістичними маршрутами, проблемами з безпекою транспортування, особливо в зонах конфліктів, а також недостатньою пропускною

здатністю транспортної інфраструктури. Кожен день затримки може означати для когось втрачену можливість отримати своєчасну допомогу.

Не можна також ігнорувати втрати вантажів під час їх транспортування та зберігання. Ці втрати можуть бути наслідком як об'єктивних причин, таких як пошкодження під час перевезення чи вплив несприятливих погодних умов, так і суб'єктивних, пов'язаних з людським фактором – недбалістю, безвідповідальністю або прямими крадіжками. Відсутність надійних механізмів моніторингу та контролю на всьому шляху слідування вантажу лише збільшує ймовірність таких втрат.

Усвідомлення цих глибоких системних проблем у сфері гуманітарної логістики підкреслює гостру необхідність у розробці та впровадженні принципово нових підходів та технологічних рішень. Програмна система “Lanka”, що пропонується в даній роботі та базується на технології блокчейн, якраз і покликана стати одним із таких рішень, здатних забезпечити необхідний рівень прозорості, підзвітності та ефективності, що є критично важливими для успішного подолання гуманітарних викликів.

1.2 Огляд сучасних технологій для прозорого відстеження поставок

Забезпечення прозорості та ефективного контролю в ланцюгах постачання є завданням, над вирішенням якого працює безліч компаній та організацій по всьому світу. З роками було розроблено та впроваджено чимало технологій, кожна з яких має свої переваги та недоліки, особливо при застосуванні у специфічній сфері гуманітарної логістики.

Традиційно, для управління ланцюгами постачання використовуються централізовані системи управління (SCM-системи), такі як системи планування ресурсів підприємства (ERP) та системи управління складом (WMS). Ці програмні комплекси добре зарекомендували себе в оптимізації внутрішніх бізнес-процесів великих компаній та координації дій між обмеженим колом довірених партнерів.

Однак, їхня централізована природа робить їх вразливими до збоїв у центральному вузлі та значно ускладнює оперативний та безпечний обмін даними з великою кількістю зовнішніх, часто тимчасових та не завжди повністю довірених учасників, що є типовою ситуацією для масштабних гуманітарних операцій. Хоча ці системи забезпечують певний рівень автоматизації та контролю, питання наскрізної прозорості для всіх сторін та гарантії незмінності записаних даних часто залишаються відкритими.

Значний крок уперед у відстеженні фізичного переміщення вантажів було зроблено завдяки впровадженню технологій радіочастотної ідентифікації (RFID) та GPS-трекінгу. RFID-мітки, прикріплені до товарів або палет, дозволяють автоматично ідентифікувати їх та збирати дані про їх переміщення без необхідності ручного сканування, наприклад, при проходженні через контрольні точки. GPS-датчики, встановлені на транспортних засобах або контейнерах, надають точну інформацію про їх географічне місцезнаходження в режимі реального часу.

Ці технології суттєво покращують видимість вантажів у ланцюгу постачання, дозволяючи оперативно реагувати на відхилення від маршруту чи затримки. Проте, самі по собі вони не вирішують фундаментальних проблем, пов'язаних з довірою до зібраних даних та їх захистом від можливої підробки чи спотворення. Інформація, отримана з RFID-зчитувачів та GPS-трекерів, зазвичай зберігається у централізованих базах даних, які, як і будь-які централізовані системи, можуть бути скомпрометовані.

Концепція Інтернету речей (IoT) відкрила ще ширші можливості для моніторингу вантажів. Різноманітні IoT-сенсори, здатні вимірювати температуру, вологість, рівень освітленості, наявність ударів чи вібрацій, можуть надавати безперервну та детальну інформацію про умови, в яких перебуває вантаж під час транспортування та зберігання. Це є особливо критичним для таких категорій гуманітарної допомоги, як медикаменти (вакцини, інсулін), що потребують суворого дотримання температурного режиму, або швидкопсувні продукти харчування. IoT-пристрої генерують величезні потоки даних, які потребують

ефективних механізмів для їх збору, передачі, надійного зберігання та подальшого аналізу.

Технологія блокчейн пропонує принципово новий підхід до забезпечення прозорості, безпеки та довіри в ланцюгах постачання. Якщо спрощено, блокчейн – це своєрідна цифрова книга обліку, яка розподілена між багатьма комп'ютерами в мережі. Кожен запис (транзакція) групується в “блок”, який потім додається до “ланцюжка” попередніх блоків. Важливо, що кожен новий блок містить криптографічний відбиток (хеш) попереднього блоку, мітку часу та самі дані транзакції. Це створює міцний зв'язок між блоками: зміна інформації в одному з попередніх блоків призведе до зміни його хешу, що, в свою чергу, зробить недійсними хеші всіх наступних блоків у ланцюжку. Щоб така зміна була прийнята, її має схвалити більшість учасників мережі, що робить непомітне втручання практично неможливим.

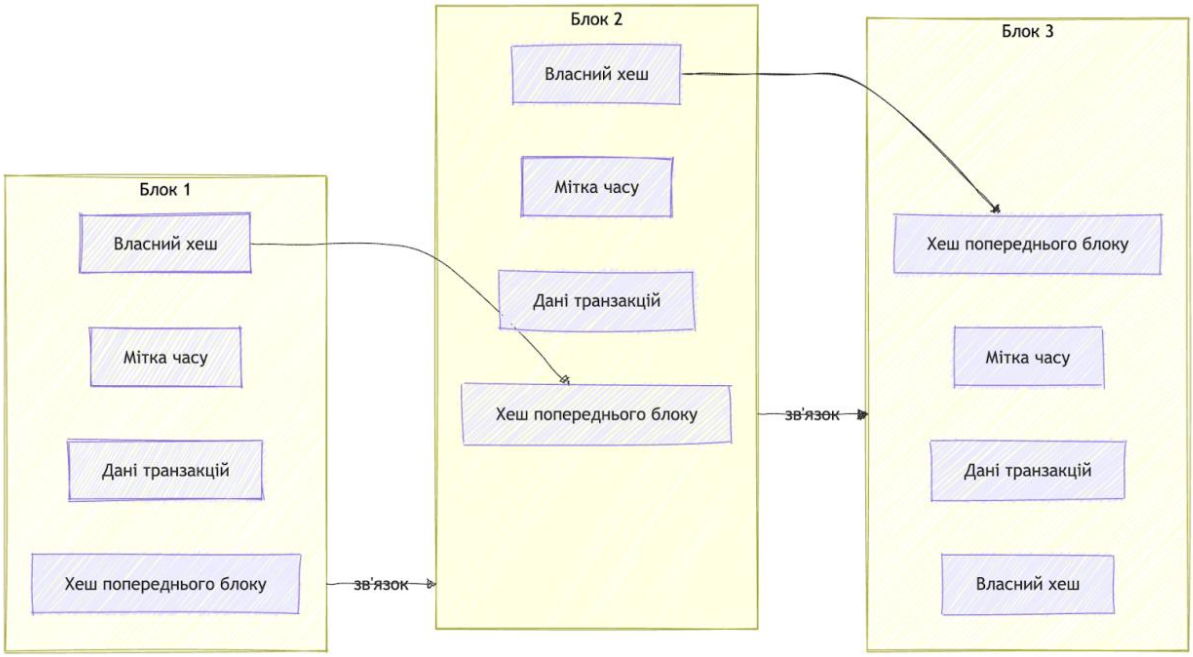


Рисунок 1.2 – Спрощена структура ланцюга блоків (блокчейну)

Ключові переваги блокчейну, що роблять його привабливим для логістики, зокрема гуманітарної, полягають у наступному:

– прозорість: Залежно від типу блокчейну (публічний, приватний, консорціумний), всі або авторизовані учасники мережі можуть мати доступ до перегляду історії транзакцій. Це забезпечує можливість відстежити рух вантажу та пов'язані з ним події;

– незмінність (Immutability): Після того, як транзакція підтверджена та записана в блок, її надзвичайно складно змінити або видалити. Це гарантує цілісність та достовірність записаної історії;

– безпека: Використання потужних криптографічних алгоритмів (хешування, цифрові підписи) забезпечує високий рівень захисту даних від несанкціонованого доступу, модифікації та підробки;

– децентралізація: У більшості блокчейн-мереж відсутній єдиний центральний орган управління чи зберігання даних. Копія реєстру зберігається на багатьох вузлах, що знижує ризики, пов'язані з відмовою або компрометацією одного центрального сервера, та підвищує загальну стійкість системи;

– підвищення довіри: Завдяки прозорості, незмінності та безпеці, блокчейн сприяє формуванню довіри між різними учасниками ланцюга постачання, навіть якщо вони не мають попередньої історії взаємодії;

– смарт-контракти: Це комп'ютерні програми, що зберігаються та виконуються безпосередньо в блокчейні. Вони автоматично виконують заздалегідь визначені умови угоди між сторонами без необхідності посередників. У логістиці смарт-контракти можуть використовуватися для автоматизації таких процесів, як підтвердження доставки, здійснення платежів при виконанні певних умов, передача прав власності на вантаж тощо.

Таблиця 1.2 – Порівняльний аналіз технологій відстеження

Критерій	**RFID**	**GPS**	**IoT**	**Централізовані БД**	**Блокчейн**
Прозорість для всіх	Низька	Низька	Середня	Середня	Висока

Продовження таблиці 1.2

Незмінність / Захист даних	Низький	Низький	Середній	Середній	Високий
Вартість розробки	Середня	Середня	Висока	Низька	Висока
Складність інтеграції	Середня	Середня	Висока	Низька	Висока
Рівень довіри	Низький	Низький	Середній	Середній	Високий
Можливість пуб. аудиту	Відсутня	Відсутня	Обмежена	Обмежена	Повна

Хоча ідея використання блокчейну для гуманітарної логістики є відносно новою, вже існують певні проекти та ініціативи, а також комерційні рішення в суміжних галузях, які варто проаналізувати.

Одним з найвідоміших прикладів була платформа TradeLens, розроблена спільно IBM та Maersk. Цей амбітний проект мав на меті оптимізувати процеси у світовій контейнерній логістиці шляхом впровадження блокчейну Hyperledger Fabric для цифровізації документообігу та покращення відстеження вантажів. Серед переваг проекту була підтримка великих гравців ринку та спроба створити єдиний стандарт. Однак, TradeLens зіткнувся з низкою проблем: висока вартість впровадження для невеликих компаній, побоювання щодо домінування одного з засновників та контролю над даними, що призвело до недостатнього прийняття ринком. Також виявилася складною інтеграція з існуючими системами численних учасників. Як наслідок, на початку 2023 року проект було закрито.

Цей досвід є надзвичайно важливим для проекту “Lanka”, оскільки він підкреслює критичну необхідність забезпечення нейтральності платформи, прозорих та зрозумілих механізмів управління, а також спрощення процесів інтеграції для широкого кола учасників, що є особливо актуальним для гуманітарної сфери з її великою кількістю різнорідних організацій.

Іншим прикладом є компанія Everledger, яка успішно використовує технологію блокчейн для створення цифрових паспортів та відстеження повної історії цінних активів, таких як діаманти, унікальні предмети мистецтва чи вишукані вина. Плюсами такого підходу є ефективна боротьба з підробками та забезпечення повної прозорості походження товарів, що значно підвищує довіру споживачів. Однак, специфіка застосування Everledger переважно обмежується ринком дорогих та унікальних товарів, що робить її менш релевантною для масових гуманітарних вантажів, хоча самі принципи детального відстеження та документування певних етапів можуть бути корисними і для проекту “Lanka”.

Платформа IBM Food Trust є ще одним яскравим прикладом успішного застосування блокчейну, цього разу в харчовій промисловості. Вона дозволяє відстежувати продукти на всьому шляху “від ферми до столу”, забезпечуючи прозорість, підвищуючи безпеку харчових продуктів та надаючи можливість швидкого та точного реагування на інциденти, пов'язані з якістю чи безпекою продукції. Залучення великих ритейлерів, таких як Walmart, сприяло широкому впровадженню цієї платформи. Для проекту “Lanka” цей досвід є цінним, оскільки демонструє ефективність блокчейну для відстеження товарів, що мають специфічні атрибути, такі як термін придатності чи умови зберігання, що безперечно актуально для певних категорій гуманітарної допомоги, наприклад, медикаментів або продуктів харчування.

Окрім великих корпоративних рішень, існують також менш масштабні проекти та стартапи, які експериментують з використанням публічних блокчейнів, таких як Ethereum або Polygon, для вирішення різноманітних завдань у логістиці. Це може включати створення NFT (невзаємозамінних токенів) для представлення унікальних вантажів або використання смарт-контрактів для автоматизації угод між учасниками ланцюга постачання. Перевагами таких рішень є потенційно більша децентралізація, відкритість коду та нижчий поріг входу для розробників. Однак, використання публічних блокчейнів, особливо першого рівня, як Ethereum, може супроводжуватися проблемами масштабованості, високою вартістю

транзакцій (особливо в періоди пікового навантаження мережі) та питаннями конфіденційності даних, якщо вони зберігаються у повністю відкритому вигляді.

Варто також згадати, що багато великих гуманітарних організацій, таких як структури ООН або Міжнародний рух Червоного Хреста і Червоного Півмісяця, вже мають власні або використовують комерційні системи для управління логістикою, які не базуються на блокчейні. Це можуть бути спеціально адаптовані ERP-системи або вузькоспеціалізовані логістичні платформи. Їхніми перевагами є те, що вони часто розроблені з урахуванням специфічних потреб конкретних організацій та інтегровані з іншими внутрішніми інформаційними системами.

Однак, недоліком таких рішень є те, що вони часто функціонують як централізовані “силоси” даних. Це значно ускладнює ефективну міжорганізаційну взаємодію та забезпечення наскрізної прозорості для зовнішніх зацікавлених сторін, таких як донори, урядові установи чи громадськість. Крім того, дані в таких централізованих системах можуть бути змінені або видалені без можливості зовнішнього незалежного контролю.

Проект “Lanka” прагне врахувати досвід існуючих рішень та запропонувати підхід, що оптимально поєднує переваги блокчейну з гнучкістю сучасних веб-технологій, орієнтуючись саме на специфічні потреби гуманітарної логістики. Ключовими аспектами позиціонування “Ланки” є гібридний підхід до зберігання даних, орієнтація на потреби всієї гуманітарної спільноти, модульність архітектури для подальшого розвитку та забезпечення нейтральності платформи, що має сприяти її ширшому прийняттю та використанню.

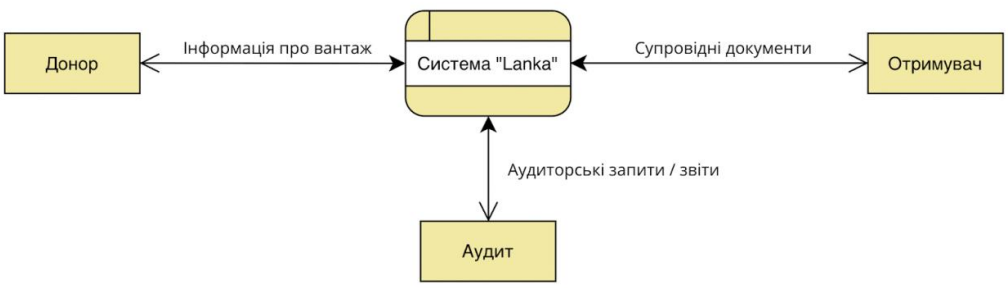


Рисунок 1.4 – Контекстна діаграма системи “Lanka”

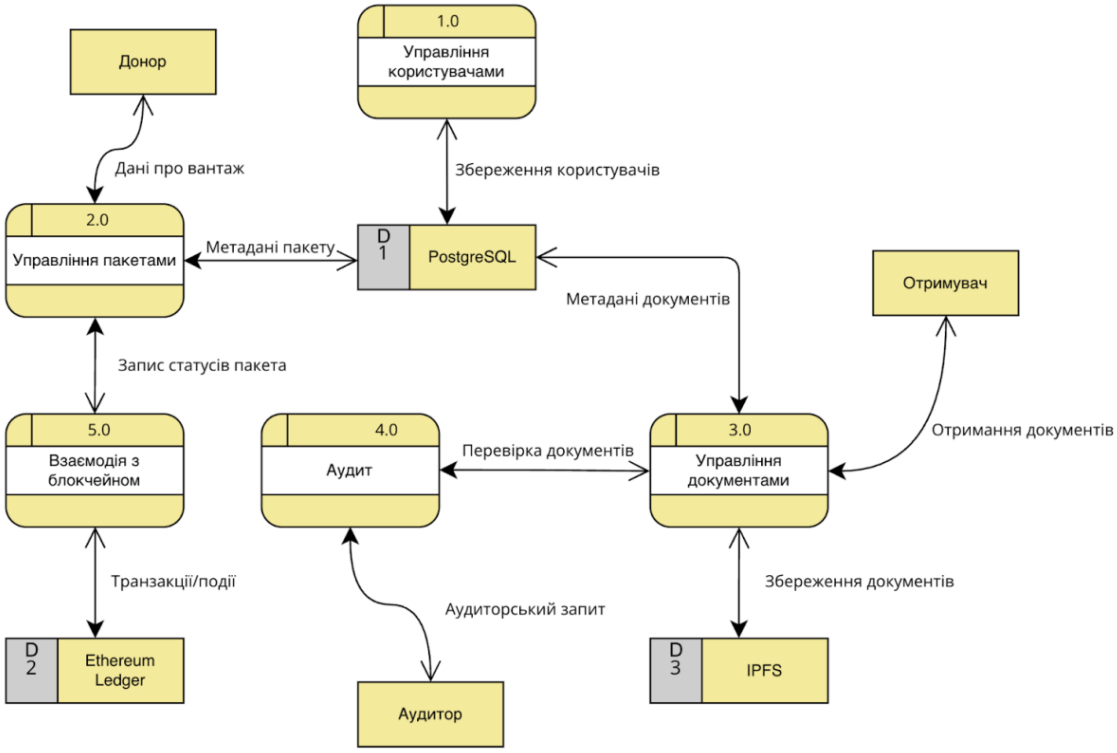


Рисунок 1.5 – Декомпозиція системи “Lanka” (DFD рівня 1)

1.3 Формулювання цілей, завдань і вимог до системи

Виходячи з проведеного аналізу проблем у сфері логістики гуманітарної допомоги та огляду можливостей сучасних технологій, можна чітко визначити основні цілі, конкретні завдання та детальні вимоги, що ставляться перед розроблюваною програмною системою “Lanka”. Це формулювання є ключовим етапом, який визначає напрямок подальшої роботи над проектом та критерії оцінки його успішності.

Головною, стратегічною ціллю проекту “Lanka” є створення та подальше впровадження інноваційної програмної платформи. Ця платформа має забезпечити кардинальне підвищення рівня прозорості, підзвітності, надійності та загальної ефективності всіх процесів, пов'язаних з управлінням та відстеженням гуманітарних вантажів у складних та багатоетапних логістичних ланцюгах. Досягнення цієї мети базується на інтеграції передових можливостей технології блокчейн, переваг децентралізованого зберігання даних та гнучкості сучасних веб-рішень. Реалізація цієї головної цілі дозволить не лише оптимізувати існуючі

процеси, але й сприятиме формуванню нового рівня довіри та співпраці між усіма учасниками гуманітарних місій.

Для досягнення цієї амбітної мети проект “Lanka” ставить перед собою низку важливих завдань.

По-перше, система повинна забезпечити надійну реєстрацію та ефективне управління користувачами з різними ролями та рівнями доступу, такими як адміністратори, донори, логісти, аудитори та отримувачі допомоги.

По-друге, необхідно реалізувати функціонал для створення та детального опису пакетів гуманітарної допомоги, включаючи інформацію про їх вміст, обсяг, походження та призначення.

По-третє, критично важливим є управління супровідною документацією, що передбачає можливість безпечного завантаження, децентралізованого зберігання (з використанням IPFS) та контрольованого доступу до всіх необхідних документів, пов'язаних з кожним вантажем.

Четверте завдання – це безперервне відстеження статусу та фізичного переміщення вантажів на всіх етапах логістичного ланцюга, з фіксацією кожної зміни в системі.

П'яте, і одне з ключових, завдань – це забезпечення повного та незмінного аудиторського сліду всіх операцій, що досягається шляхом запису ключових подій у блокчейн.

Шостим завданням є надання користувачам інструментів для генерації звітів та проведення аналізу зібраних даних для оцінки ефективності та прийняття обґрунтованих управлінських рішень.

Сьоме – це гарантування високого рівня безпеки та конфіденційності даних, що обробляються системою.

І, нарешті, восьме завдання, що має довгостроковий характер, – це передбачення можливостей для майбутньої інтеграції системи “Lanka” з іншими зовнішніми системами, такими як IoT-платформи чи фінансові інструменти.

На основі поставлених цілей та завдань формулюються детальні вимоги до системи “Lanka”, які поділяються на функціональні та нефункціональні.

Функціональні вимоги описують конкретні дії та операції, які система повинна виконувати. Система має дозволяти адміністраторам керувати обліковими записами користувачів, присвоювати їм відповідні ролі (наприклад, адміністратор, обробник вантажу, аудитор, спостерігач) та визначати рівні їхнього доступу до інформації та функцій. Користувачі, що відповідають за формування вантажів (наприклад, представники донорських організацій), повинні мати можливість створювати в системі записи про нові пакети гуманітарної допомоги, детально описуючи їх вміст, кількість, вагу, габарити, інформацію про відправника та, якщо відомо, про кінцевого отримувача.

Кожен такий пакет повинен мати унікальний ідентифікатор та проходити через визначений життєвий цикл статусів (наприклад, “зареєстровано”, “готово до відправлення”, “в дорозі”, “прибуло на проміжний склад”, “доставлено”, “отримання підтверджено”).

Логісти та інші відповідальні особи повинні мати інструменти для оперативного оновлення цих статусів та зазначення поточного місцезнаходження вантажу, причому кожна така зміна має фіксуватися в блокчейні для забезпечення її незмінності. Система повинна надавати можливість завантажувати та прив'язувати до кожного пакету допомоги необхідні супровідні документи (накладні, сертифікати, фотографії тощо), які будуть зберігатися децентралізовано в IPFS.

Всі ключові операції з вантажами – створення, зміна статусу, завантаження документа, передача відповідальності від одного учасника іншому – мають автоматично записуватися в незмінний аудиторський слід у блокчейні, доступний для перевірки авторизованим користувачам, зокрема аудиторам.

Для зручності роботи система повинна надавати інструменти для швидкого пошуку пакетів допомоги за різними параметрами (ідентифікатор, статус, відправник, дата тощо), а також можливість фільтрації та сортування отриманих результатів. Також важливою є функція генерації базових звітів про рух вантажів та візуалізації ключових показників ефективності логістики.

Нефункціональні вимоги визначають якісні характеристики системи та умови її експлуатації. Найважливішою з них є прозорість: всі дані про переміщення вантажів, зафіксовані в блокчейні, мають бути доступні для перегляду всім авторизованим учасникам, забезпечуючи повну “скляну” видимість усього ланцюга постачання.

Незмінність даних, записаних у блокчейн, є ще однією критичною вимогою, що гарантує їх достовірність. Система повинна забезпечувати високий рівень безпеки, включаючи захист від несанкціонованого доступу, розмежування прав на основі ролей та безпечну взаємодію з блокчейном через криптографічні підписи. Надійність та доступність системи мають бути на високому рівні, з мінімальним часом простою.

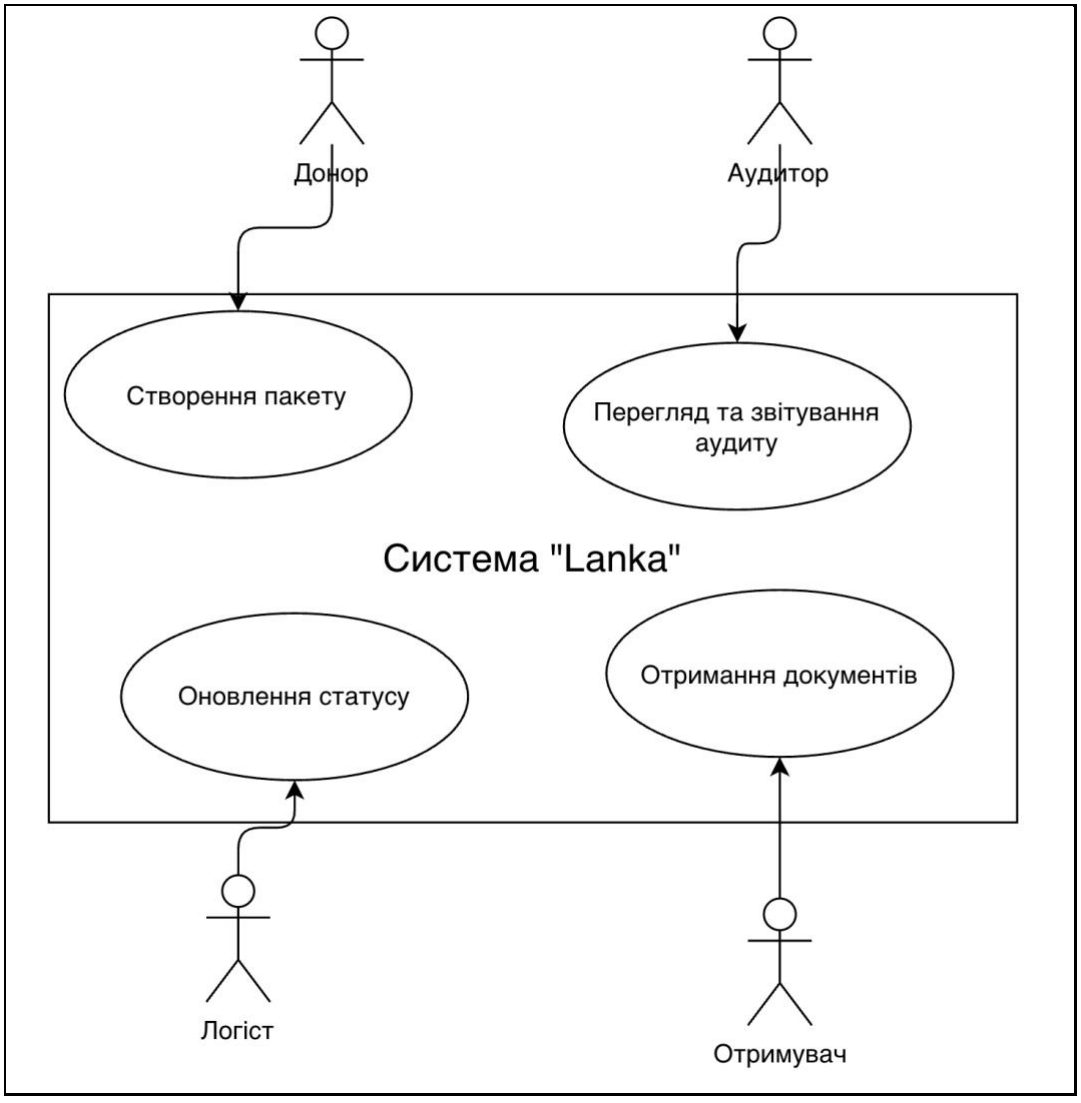


Рисунок 1.3 – Діаграма варіантів використання системи “Lanka”

Продуктивність системи має забезпечувати швидкий відгук інтерфейсу користувача на типові операції, при цьому час підтвердження транзакцій у блокчейні залежатиме від загального навантаження мережі Ethereum. Архітектура системи має бути масштабованою, дозволяючи поступове збільшення кількості користувачів та обсягу оброблюваних даних без значного погіршення продуктивності, з можливістю переходу на L2-рішення для блокчейн-компоненти.

Зручність використання є пріоритетом: інтерфейс має бути інтуїтивно зрозумілим та легким в освоєнні для користувачів з різним рівнем технічної підготовки, а також адаптивним для роботи на різних пристроях. Система має бути сумісною з останніми версіями популярних веб-браузерів. Нарешті, код системи має бути добре структурованим, документованим та легким для підтримки та подальшого розширення функціональності завдяки модульній архітектурі.

Чітке дотримання цих цілей, завдань та вимог на всіх етапах проектування, розробки та тестування дозволить створити програмну систему “Lanka”, яка відповідатиме очікуванням користувачів та зможе зробити реальний позитивний внесок у покращення процесів надання гуманітарної допомоги.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1 Архітектура: вибір гібридного підходу та модульної структури

Вибір архітектури є фундаментальним рішенням при розробці будь-якої складної інформаційної системи, і проект “Lanka” не є виключенням. Зважаючи на специфіку поставлених завдань, зокрема необхідність поєднання оперативності обробки даних, забезпечення їх незмінності та прозорості, а також інтеграції з децентралізованими технологіями, було прийнято рішення про використання гібридного архітектурного підходу.

Цей підхід дозволяє оптимально поєднати переваги традиційних централізованих систем з інноваційними можливостями блокчейну та децентралізованих сховищ даних.

Основна ідея гібридної архітектури системи “Lanka” полягає в тому, щоб розподілити функціональне навантаження та типи даних між різними компонентами системи, кожен з яких найкраще підходить для виконання певних завдань. Так, для зберігання оперативної інформації, яка може часто змінюватися, потребує швидкого доступу та не завжди вимагає абсолютного рівня незмінності (наприклад, профілі користувачів, тимчасові статуси, детальні описи, що можуть редагуватися), використовується традиційна реляційна база даних PostgreSQL. Вона забезпечує високу продуктивність, надійність та гнучкість у роботі з великими обсягами структурованих даних.

Взаємодія з цією базою даних на рівні бекенду здійснюється через Prisma ORM (Object-Relational Mapper), що значно спрощує розробку, забезпечує типобезпеку та дозволяє абстрагуватися від конкретних SQL-запитів.

Водночас, для забезпечення ключових вимог щодо прозорості, підзвітності та незмінності критично важливих даних, таких як фіксація основних етапів переміщення гуманітарного вантажу, підтвердження його доставки, а також запис аудиторського сліду всіх значущих операцій, використовується блокчейн-платформа Ethereum. Саме в блокчейн записуються незмінні “якірні” дані, що підтверджують ключові події в життєвому циклі гуманітарного вантажу. Розробка,

тестування та розгортання смарт-контрактів на мові програмування Solidity здійснюється за допомогою інструментарію Hardhat, який надає зручне середовище для компіляції, деплою та тестування.

Взаємодія фронтенд- та бекенд-частин системи з розгорнутими смарт-контрактами в мережі Ethereum реалізується через бібліотеку Ethers.js, яка є популярним та надійним інструментом для роботи з Ethereum-сумісними блокчейнами.

Ключовим аспектом архітектури є взаємодія між бекендом, смарт-контрактами та IPFS. Наприклад, при створенні нового пакету гуманітарної допомоги, основна описова інформація (яка може потребувати оновлення) зберігається в PostgreSQL. Водночас, факт створення пакету, його унікальний ідентифікатор та початковий статус фіксуються транзакцією в смарт-контракті AidPackage на блокчейні Ethereum.

Якщо до пакету додаються супровідні документи (наприклад, фотографії вантажу, накладні), ці файли завантажуються в децентралізоване сховище IPFS. Система IPFS повертає унікальний криптографічний хеш (CID) для кожного файлу. Цей CID, разом з метаданими документа (назва, тип, дата завантаження), зберігається в PostgreSQL для швидкого доступу та зв'язку з відповідним пакетом допомоги. Одночасно, цей CID може бути записаний (або сам, або як частина більш загальної транзакції) у смарт-контракт DocumentRegistry на блокчейні. Це створює незмінний доказ існування та цілісності документа на певний момент часу.

Таким чином, навіть якщо централізована база даних буде скомпрометована, інформація про ключові події та документи, зафіксована в блокчейні та IPFS, залишиться незмінною та верифікованою. При кожній суттєвій зміні статусу вантажу (наприклад, “відправлено”, “доставлено”), відповідна транзакція ініціюється на бекенді, підписується уповноваженим користувачем та надсилається до смарт-контракту AidPackage, який оновлює стан в блокчейні та генерує подію, що фіксується в AuditTrail.

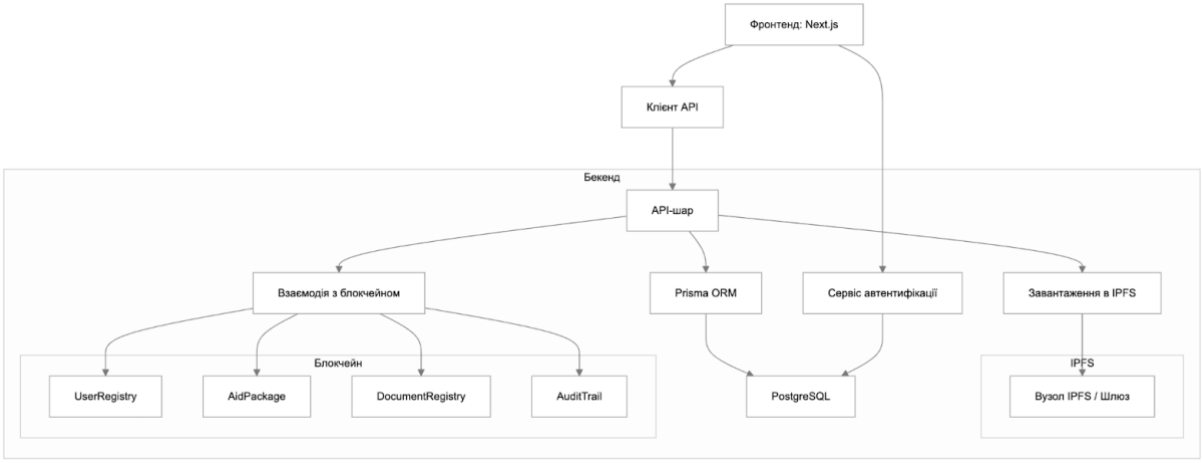


Рисунок 2.1 – Загальна архітектурна схема системи “Lanka”

Така гібридна архітектура дозволяє системі “Lanka” ефективно вирішувати поставлені завдання. Оперативні дані швидко обробляються традиційною базою даних, забезпечуючи високу швидкість роботи користувацького інтерфейсу. Ключові ж події та докази надійно фіксуються в блокчейні, гарантуючи їх прозорість та незмінність. Децентралізоване зберігання документів через IPFS забезпечує їх надійність та доступність.

Важливим аспектом архітектури є її модульність. Система “Lanka” проєктується як сукупність логічно завершених та функціонально відокремлених модулів, що взаємодіють між собою через чітко визначені інтерфейси. Такий підхід не лише відповідає сучасним практикам розробки програмного забезпечення, але й надає низку суттєвих переваг, таких як спрощення процесу розробки та тестування окремих компонентів, підвищення надійності системи загалом та, що особливо важливо, забезпечення її гнучкості та легкості розширення функціональності в майбутньому. Виділення ключових модулів дозволяє чітко розподілити відповідальність та сприяє кращій організації кодової бази.

В рамках системи “Lanka” доцільно виділити наступні основні функціональні модулі.

Модуль управління ідентифікацією та доступом (Identity and Access Management Module).

Даний модуль є фундаментальним для забезпечення безпеки та контрольованого доступу до ресурсів системи. Він охоплює процеси реєстрації нових користувачів, їх автентифікації (перевірки особистості) та авторизації (надання прав доступу відповідно до призначених ролей). Взаємодія цього модуля передбачає роботу як з централізованою базою даних для зберігання профільної інформації, так і з блокчейн-компонентою, зокрема зі смарт-контрактом UserRegistry, який може фіксувати незмінні дані про реєстрацію ключових учасників (наприклад, верифікованих організацій) та їхні ролі в системі.

Модуль управління життєвим циклом гуманітарних вантажів (Aid Package Lifecycle Management Module).

Цей центральний модуль відповідає за весь спектр операцій, пов'язаних з гуманітарними вантажами – від їх створення та детального опису до відстеження переміщення та підтвердження доставки.

Модуль тісно взаємодіє з базою даних для зберігання оперативної та детальної інформації про кожен пакет допомоги (вміст, характеристики, маршрут тощо) та з блокчейном через смарт-контракт AidPackage для незмінного запису ключових статусів, подій (наприклад, зміна відповідальної особи, проходження контрольної точки) та фіксації факту доставки.

Модуль управління документацією та доказами (Documentation and Evidence Management Module).

Завданням цього модуля є забезпечення надійного та перевіреного управління всією супровідною документацією. Це включає функціонал для завантаження документів користувачами, їх безпечного зберігання в децентралізованій системі IPFS та асоціації з відповідними гуманітарними вантажами.

Модуль взаємодіє з IPFS для фізичного зберігання файлів, з основною базою даних для метаданих та швидкого доступу, а також зі смарт-контрактом DocumentRegistry для фіксації в блокчейні криптографічних хешів документів, що слугують незмінним доказом їх існування та цілісності.

Модуль аудиту та забезпечення прозорості (Audit and Transparency Module):

Цей модуль є критично важливим для досягнення однієї з головних цілей проекту – забезпечення прозорості та підзвітності. Він відповідає за автоматичний та незмінний запис усіх значущих дій та подій, що відбуваються в системі, у спеціалізований смарт-контракт AuditTrail на блокчейні.

Це дозволяє будь-якому авторизованому учаснику або зовнішньому аудитору перевірити повну історію операцій, пов'язаних з конкретним вантажем або користувачем.

Сервісний модуль взаємодії з блокчейн-інфраструктурою (Blockchain Infrastructure Interaction Service Module).

Даний модуль виконує роль абстрактного шару, що інкапсулює всю складність безпосередньої взаємодії з мережею Ethereum. Він відповідає за формування та підписання транзакцій, виклик функцій розгорнутих смарт-контрактів, обробку подій з блокчейну та забезпечення зв'язку між традиційною бекенд-логікою та децентралізованою частиною системи.

Така декомпозиція на модулі дозволяє не лише спростити розробку та тестування, але й забезпечує необхідну гнучкість системи, дозволяючи в майбутньому легко інтегрувати нові функціональні блоки або модифікувати існуючі без кардинального впливу на загальну архітектуру та працездатність системи “Lanka”.

2.2 Опис структури даних та моделі бази даних

Ефективна організація даних є наріжним каменем будь-якої інформаційної системи, і “Lanka” не є виключенням. Зважаючи на гібридну архітектуру проекту, структура даних охоплює як моделі, що зберігаються в традиційній реляційній базі даних PostgreSQL, так і структури даних, визначені у смарт-контрактах на блокчейні Ethereum. Головний принцип при проєктуванні структури даних – це забезпечення цілісності, узгодженості та ефективного доступу до інформації, необхідної для функціонування всіх модулів системи.

В рамках реляційної бази даних, керованої Prisma ORM, визначено декілька ключових моделей, що відображають основні сутності предметної області. Модель User призначена для зберігання інформації про користувачів системи. Це включає їхні ідентифікаційні дані, контактну інформацію, а також призначені їм ролі, що визначають рівень доступу до функціоналу та даних системи. Важливо, що для взаємодії з блокчейн-компонентою кожен користувач, який має право ініціювати транзакції, також асоціюється з адресою свого Ethereum-гаманця.

Центральною моделлю є AidPackage, що представляє пакет гуманітарної допомоги. Ця модель зберігає вичерпну інформацію про кожен вантаж: його унікальний ідентифікатор, детальний опис вмісту, кількість, вагу, об'єм, інформацію про відправника та запланованого отримувача, поточний логістичний статус, історію змін статусів, географічні координати (якщо застосовно) та іншу релевантну інформацію. Ця модель є динамічною, оскільки статус та місцезнаходження вантажу можуть змінюватися протягом його життєвого циклу.

Модель Document використовується для управління супровідною документацією. Вона зберігає метадані про кожен завантажений документ, такі як його назва, тип, дата завантаження, користувач, що його завантажив, а також, що найважливіше, унікальний ідентифікатор контенту (CID), отриманий від системи IPFS, де фізично зберігається сам файл. Модель Document має зв'язок з моделлю AidPackage, дозволяючи асоціювати один або декілька документів з конкретним пакетом допомоги.

Для забезпечення прозорості та можливості аудиту, в реляційній базі даних також може існувати модель AuditTrailLog (додатково до смарт-контракту AuditTrail на блокчейні). Ця модель в PostgreSQL може дублювати або доповнювати інформацію з блокчейну для швидкого формування звітів та оперативного аналізу дій користувачів без необхідності постійних запитів до блокчейн-мережі, що може бути повільним та витратним. Вона фіксує час, тип операції, ідентифікатор користувача та об'єкта, з яким була здійснена операція.

Окрім основних, можуть бути введені додаткові, допоміжні моделі.

Важливо підкреслити, що структури даних у смарт-контрактах (UserRegistry, AidPackage, DocumentRegistry, AuditTrail) спроектовані таким чином, щоб зберігати лише найважливішу, “якірну” інформацію, яка потребує незмінності та публічної верифікації. Наприклад, смарт-контракт AidPackage може зберігати лише унікальний ID пакету, його поточний ключовий статус (наприклад, хеш статусу для економії місця) та хеш останньої значущої події, тоді як детальний опис та повна історія зберігатимуться в PostgreSQL.

Такий підхід дозволяє оптимізувати витрати на зберігання даних та виконання транзакцій у блокчейні, які є значно вищими порівняно з традиційними базами даних. Зв'язок між даними в PostgreSQL та даними в блокчейні забезпечується через унікальні ідентифікатори та хеші.

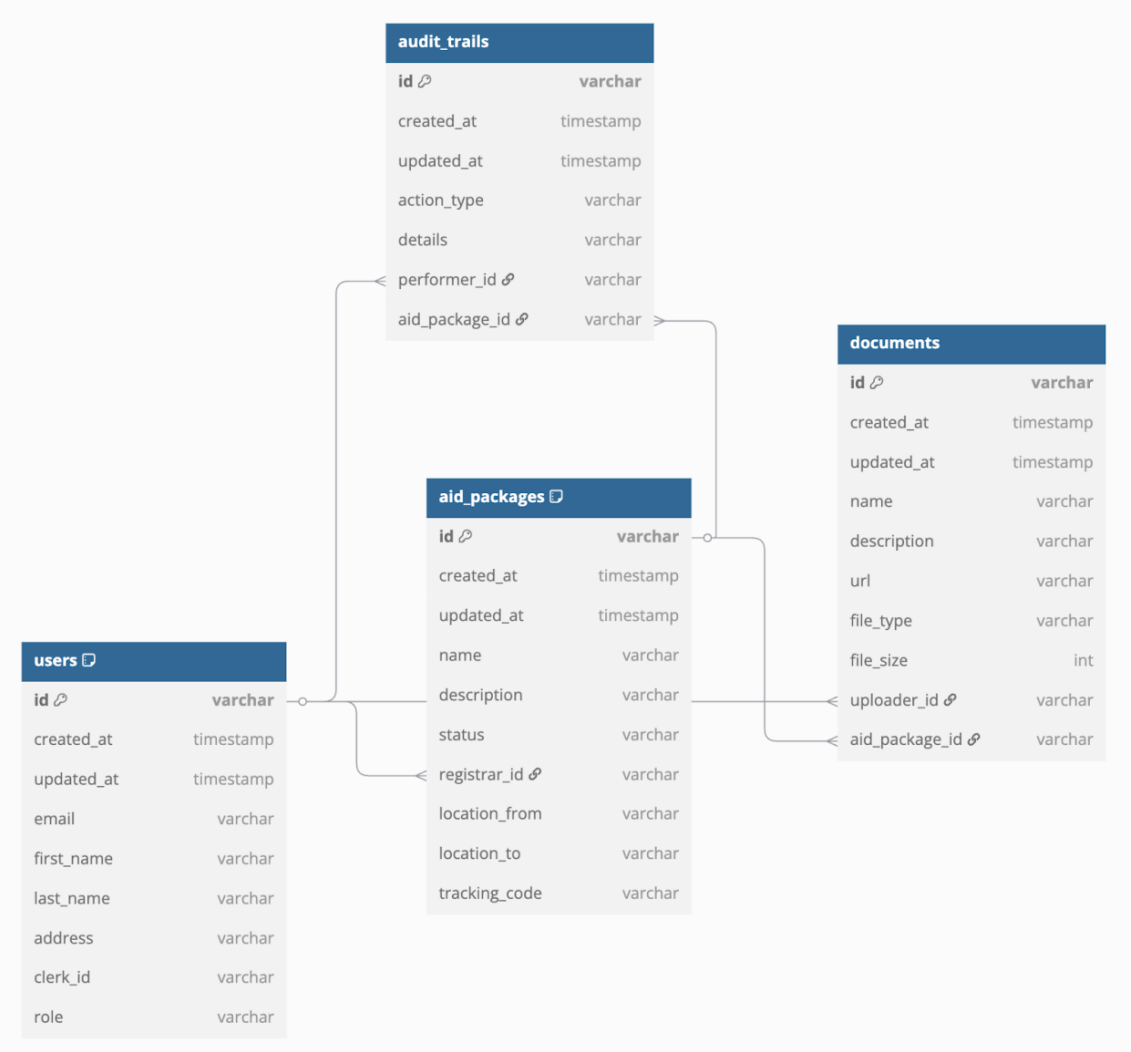


Рисунок 2.2 – Модель сутність-зв'язок для бази даних системи “Lanka”

Проєктування такої розподіленої структури даних вимагає ретельного продумування механізмів синхронізації та забезпечення узгодженості даних між централізованою та децентралізованою частинами системи. Наприклад, при оновленні статусу вантажу спочатку може відбуватися запис у PostgreSQL для негайного відображення в інтерфейсі, а потім ініціюватися транзакція в блокчейн для незмінної фіксації цієї події.

2.3 Розробка інтерфейсу користувача та UX-рішень

Користувацький інтерфейс (UI) та загальний досвід користувача (UX) відіграють надзвичайно важливу роль в успіху будь-якої програмної системи, особливо такої, що призначена для широкого кола користувачів з різним рівнем технічної підготовки, як у випадку з системою “Lanka”. Головний принцип, що лежить в основі проєктування UI/UX для “Ланки”, можна сформулювати словами відомого українського підприємця Олега Гороховського (співзасновника Monobank): “клієнти хочуть користуватись банком, не знаючи про проблеми банку”. Адаптуючи цей принцип до нашого проекту: користувачі системи “Lanka” (донори, логісти, волонтери, отримувачі допомоги) мають легко та інтуїтивно взаємодіяти з системою для виконання своїх завдань, не заглиблюючись у складність технологій, що лежать в її основі, зокрема блокчейну.

Початковим екраном взаємодії користувача із системою є інтерактивна інформаційна панель. Саме вона формує перше враження про платформу та дозволяє одразу ознайомитися зі станом ключових процесів. Тут користувач бачить оперативну інформацію: кількість активних транзакцій, кількість вузлів мережі, поточну висоту блоку, графік активності за останні 90 днів. Також доступні вкладки для швидкого перемикання між розділами «Документи», «Структура», «Персонал». Такий підхід дозволяє забезпечити швидкий доступ до важливої інформації без потреби заглиблення в багаторівневе меню або сторонні модулі системи.

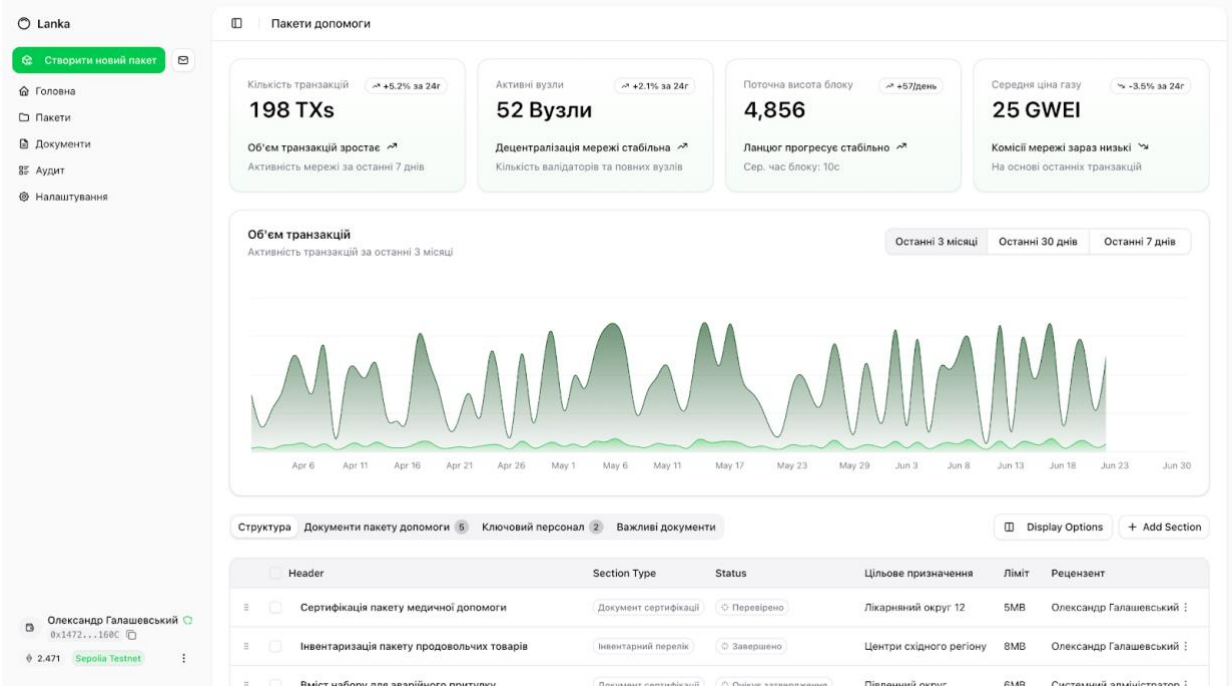


Рисунок 2.3 — Головна панель користувача системи «Lanka»

Ключовими аспектами при розробці інтерфейсу стали простота, ясність та інтуїтивність. Інтерфейс має бути максимально зрозумілим, без зайвих елементів, що можуть заплутати користувача. Навігація по системі повинна бути логічною та послідовною, дозволяючи користувачам швидко знаходити потрібні функції та інформацію. Важливим є використання зрозумілої термінології, адаптованої для гуманітарної сфери, а не суто технічної.

Одним із прикладів такої логічної організації є розділ із переліком зареєстрованих пакетів гуманітарної допомоги, де реалізовано спискове відображення з ключовими показниками.

Інтерфейс цієї сторінки дозволяє швидко зорієнтуватися у великій кількості даних завдяки фільтрам за категоріями, статусами та ключовими полями. Статуси супроводжуються кольоровими мітками, а кожен запис містить у собі назву, ідентифікатор, власника пакету та кнопку дій. Додатково реалізовано інформативні віджети, що візуалізують кількість, вартість та активність складів.

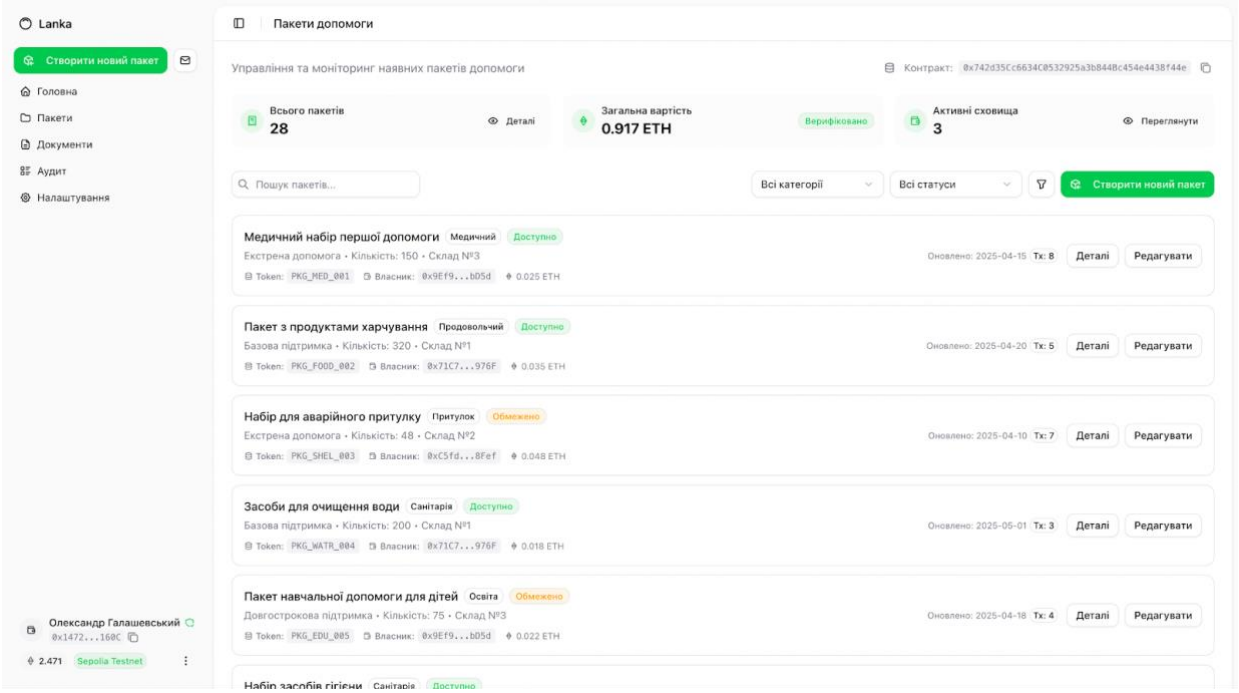


Рисунок 2.4 — Перелік зареєстрованих пакетів гуманітарної допомоги

Важливим елементом у побудові логіки UX стала наочність і простота візуалізації життєвого циклу пакету. Замість традиційної текстової історії змін, у системі реалізовано таймлайн-підхід: кожна зміна статусу, додавання документа або зміна відповідального користувача відображається на окремій шкалі часу. Це забезпечує повну прозорість переміщень вантажу й усіх дій, які були виконані протягом його шляху.

Також користувач може перейти до екрану аудиту, де представлено хронологічну стрічку дій, з інформацією про виконавця, час і тип транзакції. Це важливо не лише для підзвітності, а й для формування довіри до цифрового інструменту в гуманітарному середовищі. Для кращого розуміння процесів додано графіки розподілу активності, відображення частки успішних транзакцій та пошук за подіями.

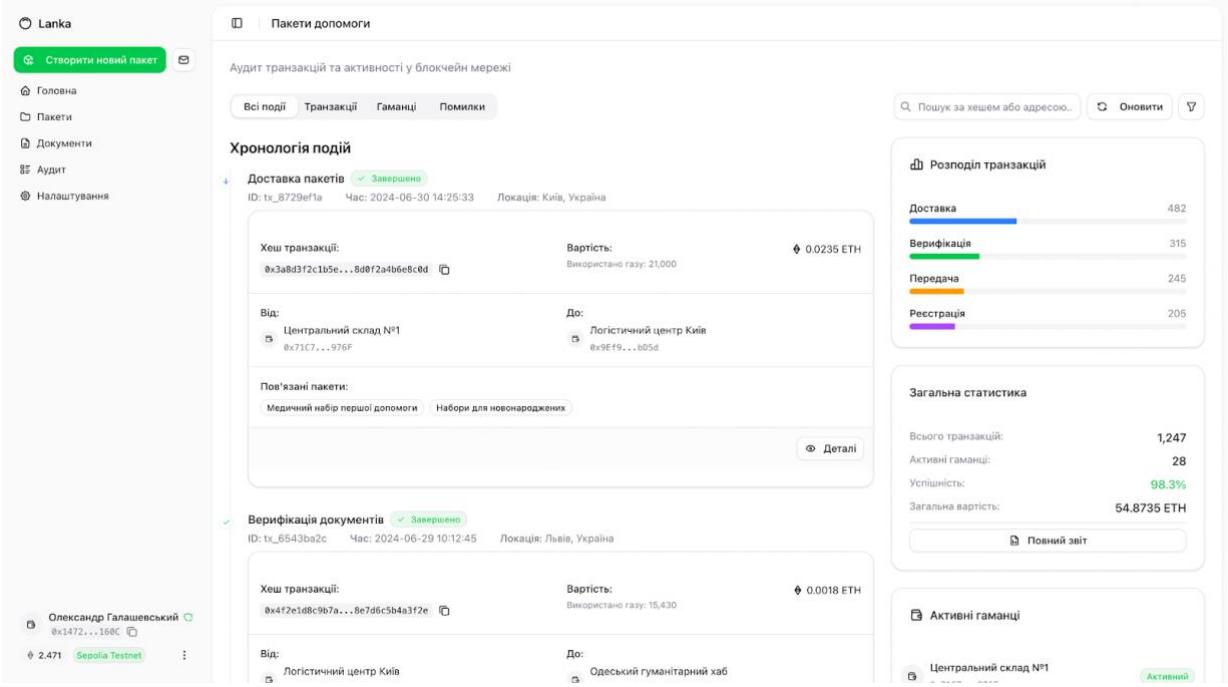


Рисунок 2.7 — Аудиторський журнал подій та статистика мережі

Враховуючи, що частина користувачів буде працювати з системою безпосередньо за допомогою мобільних пристроїв, важливо забезпечити адаптивний мобільний інтерфейс. Саме тому мобільна версія є не просто зменшеним варіантом десктопної, а повноцінно адаптованим середовищем: знижено кількість кроків до ключових функцій, застосовано нижню навігацію, додано великі кнопки дій.

Це дозволяє волонтерам і логістам працювати із системою без втрати ефективності. Окрема увага приділена швидкому доступу до функції додавання пакету, перегляду маршрутів, змін статусів та перегляду документів без затримок і складних переходів між сторінками.

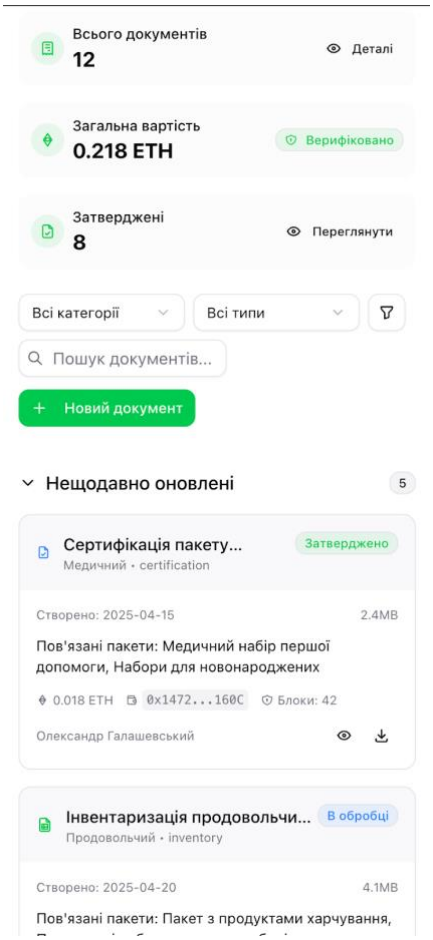


Рисунок 2.8 — Мобільна версія ключових функцій системи «Lanka»

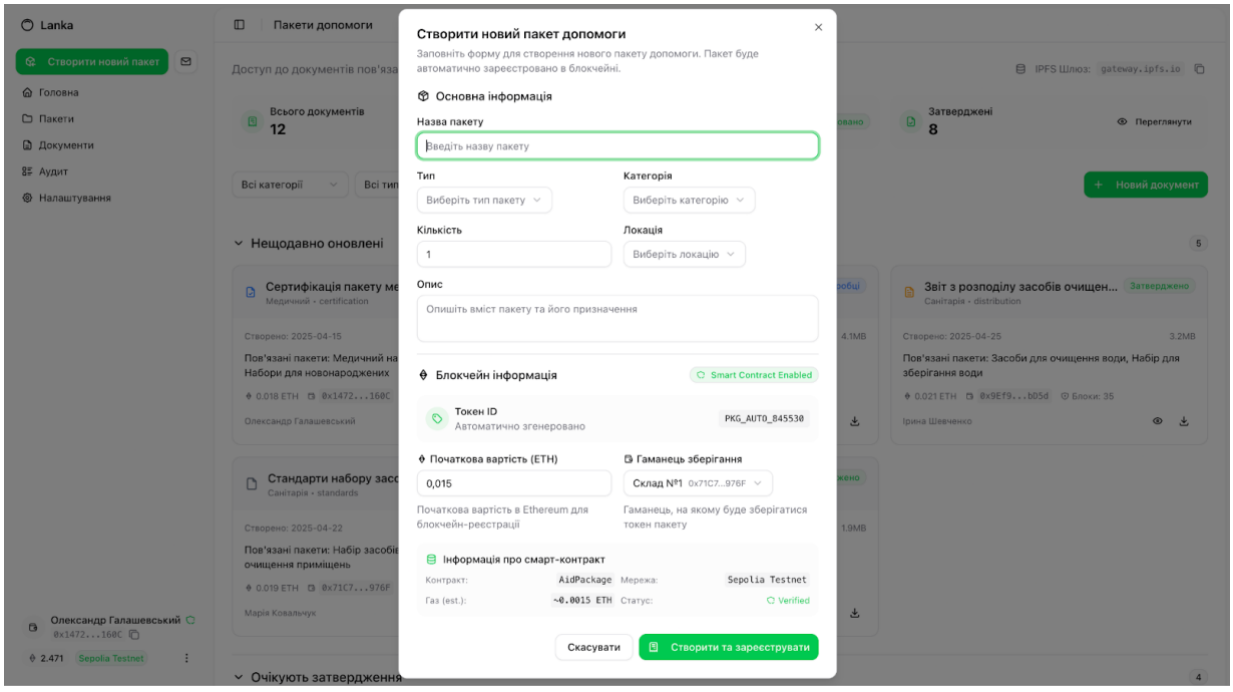


Рисунок 2.9 — Меню швидкого створення нових пакетів допомоги системи «Lanka»

Для побудови користувацького інтерфейсу обрано сучасний стек технологій: Next.js як фреймворк на базі React, TypeScript для забезпечення типобезпеки та покращення якості коду, Tailwind CSS для швидкої та гнучкої стилізації компонентів, а також бібліотека готових UI-компонентів на основі Radix UI для прискорення розробки та забезпечення консистентного вигляду. Для візуалізації даних (графіки, діаграми статусів, аналітика) використовується бібліотека Recharts, а для роботи з табличними даними – TanStack Table. Ці інструменти дозволяють створювати динамічні, інтерактивні та естетично привабливі інтерфейси.

В цілому, UX-дизайн системи “Lanka” спрямований на те, щоб зробити складний процес відстеження гуманітарних вантажів максимально простим та прозорим для кінцевого користувача, дозволяючи йому зосередитись на виконанні своїх завдань, а не на розборі тонкощів роботи системи.

2.4 Вибір стеку технологій та середовищ розробки

Вибір технологічного стеку є критично важливим рішенням, яке безпосередньо впливає на швидкість розробки, надійність, масштабованість та підтримуваність майбутньої інформаційної системи. Для проекту “Lanka”, що поєднує традиційні веб-технології з інноваційними можливостями блокчейну та децентралізованого зберігання, було ретельно підібрано набір інструментів та фреймворків, кожен з яких відіграє свою специфічну роль у загальній архітектурі.

Основний акцент при виборі робився на сучасності технологій, наявності активної спільноти розробників, доступності якісної документації та можливості ефективної інтеграції компонентів між собою.

Для розробки фронтенд-частини системи було обрано Next.js – популярний React-фреймворк, що надає широкий набір інструментів для створення швидких, оптимізованих та SEO-дружніх веб-додатків. Next.js підтримує рендеринг як на стороні сервера (SSR), так і на стороні клієнта (CSR), а також статичну генерацію сайтів (SSG), що дозволяє гнучко обирати оптимальний підхід для різних частин додатку. Використання React як основної бібліотеки для побудови користувацьких

інтерфейсів забезпечує компонентний підхід до розробки, що сприяє перевикористанню коду та спрощує управління складними UI-елементами.

Для забезпечення надійності та якості коду на фронтенді активно використовується TypeScript, який додає статичну типізацію до JavaScript, допомагаючи виявляти помилки на етапі розробки та покращуючи загальну підтримуваність проекту.

Важливим елементом взаємодії користувача з блокчейн-компонентою є підключення Ethereum-гаманця. Для спрощення цього процесу та забезпечення зручного користувацького досвіду в системі “Lanka” інтегрується бібліотека RainbowKit. Вона надає готовий набір UI-компонентів та логіку для легкого підключення різних типів гаманців (MetaMask, WalletConnect тощо) до веб-додатку, абстрагуючи від розробника багато низькорівневих деталей.

Для взаємодії з реляційною базою даних PostgreSQL на бекенді (або на рівні серверних компонентів Next.js) використовується Prisma ORM. Prisma є сучасною ORM, яка генерує типобезпечний клієнт для роботи з базою даних на основі схеми даних. Це дозволяє розробникам писати запити до бази даних, використовуючи знайомий синтаксис TypeScript/JavaScript, та отримувати переваги автодоповнення та перевірки типів, що значно зменшує ймовірність помилок та прискорює розробку.

Основою децентралізованої частини системи “Lanka” є блокчейн Ethereum та смарт-контракти, написані мовою Solidity. Solidity є об'єктно-орієнтованою мовою програмування високого рівня, спеціально розробленою для створення смарт-контрактів, що виконуються на віртуальній машині Ethereum (EVM). Вона дозволяє реалізовувати складну бізнес-логіку, яка буде виконуватися децентралізовано та незмінно.

Для розробки, компіляції, тестування та розгортання смарт-контрактів на Solidity використовується фреймворк Hardhat. Hardhat надає гнучке та потужне середовище для розробки, включаючи локальну тестову мережу, інструменти для дебагінгу та інтеграцію з іншими бібліотеками, такими як Ethers.js. Використання

Hardhat значно прискорює цикл розробки смарт-контрактів та підвищує їх надійність.

Для зберігання супровідних документів, які можуть мати значний обсяг, обрано децентралізовану систему зберігання файлів IPFS (InterPlanetary File System). IPFS дозволяє зберігати файли розподілено, використовуючи адресацію за контентом (кожен файл отримує унікальний криптографічний хеш – CID). Це забезпечує стійкість до цензури, високу доступність файлів та незмінність контенту.

Інтеграція з IPFS дозволяє системі “Lanka” надійно зберігати доказову базу (документи, фотографії) без перевантаження блокчейну та централізованих серверів.

Рисунок 2.4 – Технологічний стек інформаційної системи “Lanka”

Середовище розробки включає стандартні інструменти, такі як системи контролю версій – Git, GitHub, інтегровані середовища розробки – Visual Studio Code з відповідними плагінами для TypeScript, Solidity, менеджери пакетів npm та pnpm. Тестування різних частин системи (юніт-тести, інтеграційні тести, E2E тести) також є невід'ємною частиною процесу розробки, для чого

використовувались відповідні фреймворки та бібліотеки, такі як – Jest та Vite Test для фронтенду; інструменти Hardhat для тестування смарт-контрактів.

Додатково варто зазначити, що під час розробки активно застосовувалися можливості контейнеризації (Docker) для створення ізольованих середовищ розгортання та сервіси CI/CD – GitHub Actions, що дозволяють автоматично перевіряти, тестувати та оновлювати систему після кожного коміту. Це підвищує стабільність і передбачуваність розгортань, а також пришвидшує процес інтеграції змін.

Також особлива увага приділялася типобезпеці та якості коду: застосовувалися лінтери (ESLint), форматувачі коду (Prettier), а також інтеграція з системою GitHub для автоматичного запуску перевірок при створенні pull request'ів. Це дозволяло команді дотримуватися єдиного стилю розробки, зменшити кількість помилок та полегшити підтримку коду в майбутньому.

Вибір саме такого стеку технологій обґрунтований прагненням створити сучасну, надійну, безпечну та масштабовану систему, яка б ефективно вирішувала поставлені завдання прозорого відстеження гуманітарних вантажів, використовуючи найкращі практики та інструменти як з світу традиційної веб-розробки, так і з екосистеми децентралізованих технологій.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

3.1 Розгортання бази даних та налаштування Prisma ORM

Фундаментом будь-якої інформаційної системи, що оперує значними обсягами даних, є надійна та ефективна система управління базами даних (СУБД). Для проекту “Lanka”, який передбачає зберігання як оперативної, так і історичної інформації про гуманітарні вантажі, користувачів, документи та аудиторські записи, вибір та належне налаштування СУБД має першочергове значення.

Як було обґрунтовано на етапі проєктування архітектури, для централізованого компоненту зберігання даних було обрано реляційну СУБД PostgreSQL завдяки її потужності, надійності та відповідності стандартам. Для взаємодії програмного коду з базою даних та спрощення процесу розробки було вирішено використовувати сучасний інструмент Prisma ORM (Object-Relational Mapper).

Цей підрозділ детально розглядає етапи розгортання бази даних PostgreSQL та інтеграції Prisma ORM у проект “Lanka”, починаючи від вибору середовища для розгортання і закінчуючи генерацією та використанням типобезпечного клієнта для взаємодії з даними.

При розробці системи “Lanka” розглядалися різні варіанти розгортання бази даних PostgreSQL, що відповідають як потребам локальної розробки, так і вимогам продукційного середовища.

Для локального середовища розробки перевага надавалася використанню Docker. Docker дозволяє легко створювати ізольовані, відтворювані та портативні середовища, що є надзвичайно зручним для командної роботи та уникнення проблем типу “на моїй машині все працює”. Запуск PostgreSQL у Docker-контейнері за допомогою docker-compose.yml файлу дозволяє швидко підняти екземпляр бази даних з потрібними налаштуваннями.

Даний код ілюструє конфігурацію Docker Compose для PostgreSQL:

```
version: '3.8'
```

```
services:
  postgres_lanka:
    image: postgres:15
    container_name: lanka_db_container
    restart: always
    environment:
      POSTGRES_USER: lanka_user
      POSTGRES_PASSWORD: strong_password_lanka
      POSTGRES_DB: lanka_dev_db
    ports:
      - "5432:5432"
    volumes:
      - postgres_data_lanka:/var/lib/postgresql/data
volumes:
  postgres_data_lanka:
```

Цей конфігураційний файл визначає сервіс postgres_lanka на базі офіційного образу PostgreSQL, встановлює необхідні змінні середовища для створення користувача та бази даних, прокидає стандартний порт PostgreSQL та використовує іменований том postgres_data_lanka для збереження даних бази, що забезпечує їх персистентність.

Для продукційного середовища та середовища для демонстрації (staging), зважаючи на інтеграцію проекту з платформою Vercel (яка часто використовується для розгортання Next.js додатків), оптимальним вибором стало використання сервісу Vercel Postgres.

Vercel Postgres надає повністю керований, безсерверний екземпляр PostgreSQL, оптимізований для роботи з додатками, розгорнутими на Vercel. Переваги Vercel Postgres:

- простота інтеграції: Легко підключається до проектів на Vercel.
- безсерверна архітектура: Автоматичне масштабування та відсутність необхідності в адмініструванні серверів баз даних.
- безпека: Вбудовані механізми захисту та шифрування.
- розташування близько до функцій: Мінімізація затримок при взаємодії між бекенд-функціями – Next.js API Routes, а також Vercel Functions та базою даних.
- підключення до Vercel Postgres здійснюється через рядок підключення (DATABASE_URL), який надається сервісом та безпечно зберігається у змінних середовища проекту на Vercel.

Після налаштування доступу до екземпляра PostgreSQL (локального чи хмарного через Vercel Postgres), наступним кроком є інтеграція Prisma ORM у проект “Lanka”. Це включає встановлення Prisma CLI та Prisma Client, ініціалізацію Prisma в проекті (npx prisma init), що створює директорію prisma з файлом схеми schema.prisma та оновлює файл .env для зберігання рядка підключення до бази даних (DATABASE_URL).

Файл prisma/schema.prisma є центральним місцем для визначення моделей даних, їхніх полів, типів та взаємозв'язків за допомогою Prisma Schema Language (PSL). На основі цієї схеми Prisma генерує типобезпечний клієнт для взаємодії з базою даних та SQL-скрипти для міграцій.

Даний код ілюструє визначення моделей даних у Prisma Schema Language (PSL):

```
datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}

generator client {
  provider = "prisma-client-js"
}

model User {
  id        String   @id @default(cuid())
  email     String   @unique
  name      String?
  role      Role     @default(VIEWER)
  walletAddress String? @unique

  createdPackages AidPackage[] @relation("CreatedPackages")
  handledPackages AidPackage[] @relation("HandledPackages")
  documents       Document[]

  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt
}

enum Role {
  ADMIN
  HANDLER
  AUDITOR
  VIEWER
}

model AidPackage {
  id            String     @id @default(cuid())
  title         String
  description   String?
  status        PackageStatus @default(REGISTERED)
```

```
        creatorId     String
       creator       User        @relation("CreatedPackages", fields: [creatorId],
references: [id])

        currentHandlerId String?
       currentHandler    User?             @relation("HandledPackages",    fields:
[currentHandlerId], references: [id])

        documents     Document[]

        blockchainTxId String?    @unique
        createdAt     DateTime   @default(now())
        updatedAt     DateTime   @updatedAt
      }

      enum PackageStatus {
        REGISTERED
        PREPARED_FOR_SHIPMENT
        IN_TRANSIT
        ARRIVED_HUB
        DELIVERED
        RECEIVED_CONFIRMED
        CANCELLED
      }

      model Document {
        id           String   @id @default(cuid())
        fileName     String
        fileType     String     description String?
        ipfsCid      String   @unique
        uploaderId   String
        uploader     User     @relation(fields: [uploaderId], references: [id])

        aidPackageId String?
       aidPackage  AidPackage? @relation(fields: [aidPackageId], references: [id])

        blockchainRegistryTxId String? @unique
        createdAt    DateTime @default(now())
        updatedAt    DateTime @updatedAt
      }
```

У цьому прикладі показано, як визначаються моделі User, AidPackage та Document, їхні поля з типами даних (наприклад, String, DateTime, enum Role та PackageStatus). Важливим є визначення зв'язків: AidPackage може мати багато Document (зв'язок “один-до-багатьох”), а також зв'язки з User для позначення творця та поточного відповідального. Поля blockchainTxId та blockchainRegistryTxId призначені для зберігання ідентифікаторів транзакцій з блокчейну, що забезпечує зв'язок між даними в реляційній БД та записами в блокчейні.

Після визначення або модифікації схеми даних у schema.prisma, необхідно застосувати ці зміни до фізичної структури бази даних. Prisma Migrate автоматизує

цей процес. Команда npx prisma migrate dev --name <migration_name> аналізує зміни в schema.prisma порівняно з поточним станом бази даних, генерує відповідні SQL-скрипти міграції та застосовує їх до бази даних розробки.

Це забезпечує версіонований та контрольований підхід до еволюції схеми БД. Для продукційних середовищ використовується команда npx prisma migrate deploy, яка застосовує вже існуючі, протестовані міграції.

Після кожної успішної міграції (або просто для оновлення клієнта) виконується команда npx prisma generate. Вона читає актуальну схему з schema.prisma та генерує типобезпечний Prisma Client, який буде використовуватися в коді бекенду для взаємодії з базою даних.

Prisma Client також надає інтуїтивно зрозумілий API для виконання запитів до бази даних. Розглянемо сценарій, де користувач завантажив документ в IPFS (отримавши ipfsCid), і тепер цей документ потрібно зберегти в базі даних та прив'язати до існуючого пакету гуманітарної допомоги.

Даний код ілюструє сервісний метод для роботи з Prisma Client:

```
import { PrismaClient, Document, AidPackage } from '@prisma/client';

const prisma = new PrismaClient();

/**
 * Створює запис про документ та прив'язує його до існуючого пакету допомоги.
 * @param params - Параметри для створення та прив'язки документа.
 * @returns Створений об'єкт документа.
 * @throws Error, якщо пакет допомоги або користувач-завантажувач не знайдені.
 */
export async function attachDocumentToPackage(
  params: AttachDocumentToPackageParams
): Promise<Document> {
  const {
    aidPackageId,
    fileName,
    fileType,
    ipfsCid,
    uploaderId,
    description,
  } = params;

  const aidPackageExists: AidPackage | null = await
prisma.aidPackage.findUnique({
    where: { id: aidPackageId },
  });

  if (!aidPackageExists) {
    throw new Error(`AidPackage with ID ${aidPackageId} not found.`);
```

```
    }

    const uploaderExists = await prisma.user.findUnique({
      where: { id: uploaderId },
    });

    if (!uploaderExists) {
      throw new Error(`Uploader User with ID ${uploaderId} not found.`);
    }

    try {
      const newDocument = await prisma.document.create({
        data: {
          fileName,
          fileType,
          ipfsCid,
          description: description || null,
          uploader: {
            connect: { id: uploaderId },
          },
          aidPackage: {
            connect: { id: aidPackageId },
          },
        },
        include: {
          uploader: true,
          aidPackage: true,
        },
      });

      console.log(`Document ${newDocument.id} created and attached to AidPackage
${aidPackageId}`);
      return newDocument;
    } catch (error) {
      console.error("Error attaching document to package:", error);
      if (error.code === 'P2002' && error.meta?.target?.includes('ipfsCid')) {
        throw new Error(`Document with IPFS CID ${ipfsCid} already exists.`);
      }
      throw new Error("Failed to attach document to package.");
    }
  }

  async function attach() {
    try {
      const document = await attachDocumentToPackage({
        aidPackageId: 'clxyz12340000someuuidpackage',
        fileName: 'shipping_invoice.pdf',
        fileType: 'application/pdf',
        ipfsCid: 'QmXoypizjW3WknFiJnKLwHCnL72vedxjQkDDP1mXWo6uco',
        uploaderId: 'clabc56780000someuuiduser',
        description: 'Рахунок-фактура на відправлення',
      });
      console.log('Successfully attached document:', document);
    } catch (e) {
      console.error('Error in example usage:', e.message);
    } finally {
      await prisma.$disconnect();
    }
  }
```

У наведеному прикладі для взаємодії з базою даних використовується клієнт PrismaClient. Основна функція attachDocumentToPackage приймає набір параметрів, зокрема aidPackageId та ipfsCid, які є ключовими для створення нового документа та його прив’язки до відповідного пакету допомоги.

Перед створенням документа функція перевіряє, чи існує запис пакету допомоги (AidPackage) з вказаним ідентифікатором, а також перевіряє наявність користувача-завантажувача у базі даних. Якщо одна з цих сутностей не знайдена, функція викидає помилку.

Для створення запису в таблиці Document використовується метод prisma.document.create. У його секції data встановлюються зв’язки з уже існуючими записами таблиць User та AidPackage через оператор connect: { id: ... }, що дозволяє зафіксувати зовнішні ключі у створюваному документі.

У функції реалізовано блок try...catch, який обробляє можливі помилки, зокрема випадки, коли документ з таким самим ipfsCid вже існує в системі. Це унеможливлюється завдяки використанню атрибута @unique у Prisma-схемі для цього поля.

Щоб разом з новим документом одразу отримати пов’язані дані, такі як інформація про користувача-завантажувача та відповідний пакет допомоги, використовується опція include. Вона забезпечує зручний доступ до пов’язаних об'єктів без необхідності додаткових запитів.

Допоміжна функція exampleUsage демонструє, як викликати цей метод на практиці. У її тілі після завершення всіх операцій обов’язково викликається await prisma.$disconnect() для коректного закриття з’єднання з базою даних. Це особливо важливо у випадках короткотривалих сценаріїв, наприклад, у serverless-функціях. У довготривалих серверних застосунках зазвичай створюється один екземпляр PrismaClient на весь життєвий цикл програми, щоб уникнути надмірного створення підключень.

Усі функції на бекенді реалізовані за аналогічною структурою: з чіткою перевіркою вхідних параметрів, безпечною взаємодією з базою даних через Prisma,

обробкою винятків та дотриманням зв’язності між таблицями. Такий підхід гарантує стабільність, узгодженість та масштабованість всієї серверної логіки.

Такий підхід, що поєднує чітко визначену схему даних, автоматизовані міграції та типобезпечний клієнт, значно підвищує надійність та ефективність розробки бекенд-логіки, пов'язаної зі зберіганням та обробкою даних у системі «Lanka». Це створює міцний фундамент для реалізації більш складних бізнес-процесів та взаємодії з децентралізованими компонентами системи.

3.2 Імплементація основних модулів: блокчейн, бекенд, фронтенд

Після завершення етапу проєктування архітектури, структури даних та вибору технологічного стеку, настає фаза безпосередньої реалізації інформаційної системи «Lanka». Цей підрозділ присвячений детальному розгляду процесу імплементації ключових компонентів системи, які включають розробку смарт-контрактів для блокчейн-частини, створення бекенд-логіки для управління даними та взаємодії з децентралізованими сервісами, а також побудову користувацького інтерфейсу на фронтенді. Відповідно до обраної модульної архітектури, розробка велася з чітким розподілом функціональності між цими трьома основними шарами, що забезпечує їх відносну незалежність та можливість паралельної роботи над різними частинами системи.

Взаємозв'язок між цими компонентами є ключовим для функціонування гібридної системи “Lanka”. Фронтенд слугує точкою взаємодії для користувачів, збираючи вхідні дані та відображаючи інформацію. Бекенд виступає в ролі проміжної ланки, обробляючи бізнес-логіку, керуючи даними в реляційній базі PostgreSQL, взаємодіючи з децентралізованим сховищем IPFS для файлів, та ініціюючи транзакції або запити до блокчейн-компоненту. Блокчейн-компонент, реалізований у вигляді смарт-контрактів на Ethereum, забезпечує незмінне та прозоре зберігання критично важливих даних та виконання децентралізованої логіки.

Розглянемо детальніше процес імплементації кожного з цих компонентів.

Блокчейн-компонент системи “Lanka” є фундаментом для забезпечення прозорості, незмінності та підзвітності операцій з гуманітарними вантажами. Його реалізація передбачала розробку набору смарт-контрактів на мові програмування Solidity, які розгортаються в мережі Ethereum. Для управління процесом розробки, компіляції, тестування та деплою цих контрактів використовувався фреймворк Hardhat. Важливим аспектом було забезпечення безпеки та надійності смарт-контрактів, тому, де це було доцільно, застосовувалися перевірені практики та бібліотеки, такі як OpenZeppelin Contracts, для реалізації стандартних механізмів, наприклад, контролю доступу.

Основними смарт-контрактами, що формують блокчейн-логіку системи “Lanka”, є UserRegistry, AidPackage, DocumentRegistry та AuditTrail. Кожен з них виконує специфічні функції, спрямовані на децентралізоване управління ключовими аспектами системи.

Смарт-контракт UserRegistry.sol слугує для управління ідентифікаційною інформацією та ролями основних учасників системи. Він дозволяє адміністраторам, які мають відповідні повноваження (наприклад, власник контракту або адреси, додані до списку адміністраторів), реєструвати нових користувачів або організації, присвоюючи їм одну з визначених ролей: ADMIN, HANDLER (відповідальний за обробку та переміщення вантажів), AUDITOR (має права на перевірку даних) або VIEWER (має права лише на перегляд інформації). Цей контракт зберігає відповідність між адресою Ethereum-гаманця учасника та його роллю, що дозволяє іншим смарт-контрактам перевіряти повноваження перед виконанням певних дій. При кожній реєстрації або зміні ролі генерується відповідна подія (UserRegistered, UserRoleUpdated), яка записується в блокчейн і може бути відстежена.

Даний код ілюструє фрагмент смарт-контракту UserRegistry.sol

```
pragma solidity ^0.8.20;

import "@openzeppelin/contracts/access/Ownable.sol";
```

```
    contract UserRegistry is Ownable {
        enum Role { NONE, ADMIN, HANDLER, AUDITOR, VIEWER }
        mapping(address => Role) public userRoles;

        event UserRegistered(address indexed userAddress, Role indexed role,
uint256 timestamp);

        constructor() Ownable(msg.sender) {
            userRoles[msg.sender] = Role.ADMIN;
            emit UserRegistered(msg.sender, Role.ADMIN, block.timestamp);
        }

        function registerUser(address _userAddress, Role _role) public onlyOwner
{
            require(_userAddress != address(0), "Invalid address");
            require(_role != Role.NONE, "Cannot assign NONE role");
            userRoles[_userAddress] = _role;
            emit UserRegistered(_userAddress, _role, block.timestamp);
        }
    }
```

Цей приклад демонструє основну логіку: визначення ролей, зберігання їх у меппінгу та функцію реєстрації, захищену модифікатором доступу.

Центральним елементом блокчейн-архітектури є смарт-контракт AidPackage.sol. Його призначення – незмінна фіксація ключових етапів життєвого циклу кожного пакету гуманітарної допомоги.

При створенні нового пакету в цей контракт записується його унікальний ідентифікатор (наприклад, хеш початкових даних, що забезпечує його унікальність та зв'язок з офчейн-даними), початковий статус (наприклад, `REGISTERED`), адреса відповідального обробника та хеш детальної описової інформації про пакет, яка зберігається поза блокчейном (наприклад, в PostgreSQL). Надалі, при кожній значущій зміні статусу вантажу (наприклад, “В дорозі”, “Доставлено”) або при передачі відповідальності іншому обробнику, відповідна функція контракту викликається, оновлюючи стан пакету в блокчейні та генеруючи подію (`PackageStatusUpdated`).

Це створює прозору та незмінну історію переміщення та обробки кожного вантажу. Доступ до функцій оновлення статусу зазвичай обмежується поточним відповідальним обробником або адміністратором системи.

Даний код ілюструє фрагмент смарт-контракту AidPackage.sol:

```
pragma solidity ^0.8.20;

import "./UserRegistry.sol";

contract AidPackage {
    UserRegistry public userRegistry;
    enum PackageStatus { NONE, REGISTERED, IN_TRANSIT, DELIVERED, CONFIRMED
}

    struct Package {
        PackageStatus currentStatus;
        address currentHandler;
        string packageDataHash;
        uint256 updatedAt;
    }
    mapping(bytes32 => Package) public packages;
    event PackageCreated(bytes32 indexed packageId, address indexed creator,
PackageStatus initialStatus);
    event PackageStatusUpdated(bytes32 indexed packageId, PackageStatus
newStatus, address indexed newHandler);

    constructor(address _userRegistryAddress) {
        userRegistry = UserRegistry(_userRegistryAddress);
    }

    function createPackage(bytes32 _packageId, PackageStatus _initialStatus,
address _initialHandler, string memory _packageDataHash) public {
        require(userRegistry.hasRole(msg.sender, UserRegistry.Role.HANDLER)
|| userRegistry.hasRole(msg.sender, UserRegistry.Role.ADMIN), "Caller cannot
create packages");
        require(packages[_packageId].updatedAt == 0, "Package ID already
exists");
        packages[_packageId] = Package({
            currentStatus: _initialStatus,
            currentHandler: _initialHandler,
            packageDataHash: _packageDataHash,
            updatedAt: block.timestamp
        });
        emit PackageCreated(_packageId, msg.sender, _initialStatus);
    }

    function updatePackageStatus(bytes32 _packageId, PackageStatus
_newStatus, address _newHandler) public {
        Package storage pkg = packages[_packageId];
        require(pkg.updatedAt != 0, "Package does not exist");
        require(msg.sender == pkg.currentHandler ||
userRegistry.hasRole(msg.sender, UserRegistry.Role.ADMIN), "Caller is not
authorized");

        pkg.currentStatus = _newStatus;
        pkg.currentHandler = _newHandler;
        pkg.updatedAt = block.timestamp;
        emit PackageStatusUpdated(_packageId, _newStatus, _newHandler);
    }
}
```

Цей контракт показує зберігання основних даних пакету та логіку створення і оновлення його статусу з відповідними перевірками прав доступу.

Для управління супровідною документацією використовується смарт-контракт DocumentRegistry.sol. Його основна функція – реєстрація криптографічних хешів (CID) документів, що зберігаються в IPFS, та їх прив'язка до конкретних пакетів гуманітарної допомоги. Коли користувач завантажує документ через систему “Lanka”, файл зберігається в IPFS, а отриманий CID разом з типом документа та ідентифікатором відповідного пакету записується в DocumentRegistry.

Це створює незмінний доказ існування та цілісності документа на певний момент часу, а також його зв'язок з конкретним вантажем. Кожна така реєстрація супроводжується генерацією події DocumentRegistered.

Смарт-контракт `AuditTrail.sol` відіграє роль децентралізованого та незмінного журналу аудиту. Він призначений для агрегації та зберігання записів про всі ключові дії, що відбуваються в системі. Інші смарт-контракти (`AidPackage`, `DocumentRegistry`, `UserRegistry`) при виконанні значущих операцій (наприклад, створення пакету, зміна статусу, реєстрація документа) викликають спеціальну функцію в AuditTrail для логування цієї події.

Кожен запис аудиту містить інформацію про ініціатора дії, цільовий об'єкт (наприклад, ID пакету), тип виконаної дії, мітку часу та, можливо, додаткові деталі. Це забезпечує повну простежуваність всіх операцій, що є критично важливим для забезпечення прозорості та проведення незалежного аудиту.

Реалізація цих смарт-контрактів вимагала ретельного продумування структур даних для оптимізації витрат на газ (комісію за виконання операцій в мережі Ethereum), а також чіткого визначення прав доступу до функцій та механізмів генерації подій для можливості відстеження змін зовнішніми додатками та індексаторами.

Процес розгортання (деплою) смарт-контрактів у тестову мережу Ethereum – Sepolia, здійснювався за допомогою спеціальних скриптів, написаних для Hardhat. Ці скрипти автоматизують процес компіляції контрактів, їх завантаження в мережу та отримання адрес розгорнутих екземплярів. Отримані адреси потім використовуються для налаштування взаємодії з бекенд- та фронтенд-частинами

системи “Lanka”. Аналогічний процес передбачається і для розгортання в основну мережу Ethereum (Mainnet) або обране L2-рішення на етапі продуктивного впровадження.

Таким чином, блокчейн-компонент системи “Lanka” формує надійний децентралізований шар, що забезпечує ключові властивості прозорості та незмінності даних, які є основою для побудови довіри в процесі управління гуманітарною допомогою.

Бекенд системи “Lanka” є центральним обчислювальним та координаційним вузлом, відповідальним за реалізацію основної бізнес-логіки, управління даними та забезпечення взаємодії між різними шарами гібридної архітектури. Він побудований на платформі Node.js, з активним використанням можливостей фреймворку Next.js для створення API Routes, що забезпечує тісну інтеграцію з фронтенд-частиною. Ключовими завданнями бекенду є обробка HTTP-запитів від клієнтського додатку, валідація вхідних даних, взаємодія з реляційною базою даних PostgreSQL за допомогою Prisma ORM, організація завантаження та отримання файлів з децентралізованого сховища IPFS, а також ініціація транзакцій та зчитування даних зі смарт-контрактів, розгорнутих у мережі Ethereum, через бібліотеку Ethers.js.

Одним із фундаментальних аспектів функціонування бекенду є оркестрація операцій, що вимагають узгодженої роботи з кількома системами зберігання даних. Розглянемо, для прикладу, процес створення нового пакету гуманітарної допомоги. Цей процес ініціюється відповідним запитом від фронтенду, який містить детальну інформацію про вантаж та, можливо, файл супровідного документа. На бекенді ця операція реалізована в рамках сервісного модуля AidPackageService.

Сервісний метод `create` в `AidPackageService` демонструє цю комплексну взаємодію.

Даний код ілюструє метод створення пакету в AidPackageService:

```
import { PrismaClient } from '@prisma/client';
import { ethers } from 'ethers';
import { IpfsService } from './ipfs.service';
```

```
      const prisma = new PrismaClient();
      const ipfs = new IpfsService();
      const provider = new ethers.JsonRpcProvider(process.env.RPC_URL!);
      const signer = new ethers.Wallet(process.env.PRIVATE_KEY!, provider);
      const aidContract = new ethers.Contract(process.env.AID_CONTRACT!,
require('../abis/AidPackage.json').abi, signer);

      interface CreateInput {
        title: string;
        creatorId: string;
        handlerWallet: string;
        file?: { buffer: Buffer; name: string; mime: string };
      }

      export class AidPackageService {
        async create(input: CreateInput) {
          const ipfsCid = input.file ? (await
ipfs.uploadFile(input.file.buffer)).cid : null;
          const pkg = await prisma.aidPackage.create({
            data: {
              title: input.title,
              creatorId: input.creatorId,
              status: 'REGISTERED',
              currentHandler: {
                connectOrCreate: {
                  where: { walletAddress: input.handlerWallet },
                  create: { walletAddress: input.handlerWallet, email:
`${input.handlerWallet}@temp.lanka`, role: 'HANDLER' },
                },
              },
              documents: ipfsCid
                ? {
                    create: {
                      fileName: input.file!.name,
                      fileType: input.file!.mime,
                      ipfsCid,
                      uploaderId: input.creatorId,
                    },
                  }
                : undefined,
            },
            include: { documents: true },
          });

          const pkgId = ethers.id(pkg.id);
          const metaHash = ethers.id(JSON.stringify({ title: pkg.title, creator:
input.creatorId }));
          const tx = await aidContract.createPackage(pkgId, 0,
input.handlerWallet, metaHash);
          const receipt = await tx.wait();

          await prisma.aidPackage.update({ where: { id: pkg.id }, data: {
blockchainTxId: receipt.hash } });

          return { db: pkg, txHash: receipt.hash };
        }
      }
```

Наведений фрагмент коду демонструє послідовність операцій: спочатку відбувається взаємодія з IPFS для збереження файлу, потім створюються відповідні

записи в реляційній базі даних PostgreSQL за допомогою Prisma, і нарешті, ключова інформація про створення пакету фіксується шляхом надсилання транзакції до смарт-контракту AidPackage на блокчейні Ethereum. Хеш цієї транзакції зберігається в PostgreSQL для подальшого аудиту та зв'язку між системами. Важливою є атомарність операцій з базою даних, яка може бути забезпечена Prisma-транзакціями, та обробка можливих помилок на кожному етапі, особливо при взаємодії з зовнішніми децентралізованими системами.

Окрім операцій створення, бекенд реалізує API-ендпоінти для оновлення статусів пакетів допомоги (що також включає валідацію прав користувача, оновлення даних в PostgreSQL та ініціацію відповідної транзакції до смарт-контракту `AidPackage.sol`), отримання списків пакетів з можливостями гнучкої фільтрації, сортування та пагінації (де Prisma Client демонструє свою ефективність), а також для отримання детального аудиторського сліду для конкретного вантажу, агрегуючи дані з блокчейну (AuditTrail.sol) та, за потреби, з локального журналу в PostgreSQL.

Безпека API забезпечується механізмами автентифікації та авторизації, де перевіряються права користувача перед виконанням будь-яких операцій, що змінюють стан системи або блокчейну.

Фронтенд системи “Lanka” слугує точкою входу та основним інструментом взаємодії для всіх категорій користувачів. Розроблений на сучасному стеку, що включає Next.js, React та TypeScript, він спрямований на забезпечення інтуїтивно зрозумілого, швидкого та функціонального користувацького досвіду. Важливим завданням фронтенду є не лише відображення інформації та надання форм для введення даних, але й забезпечення зручної та безпечної взаємодії з Ethereum-гаманцем користувача для підписання транзакцій, що є невід'ємною частиною роботи з блокчейн-компонентом.

Ключовим елементом взаємодії користувача з блокчейн-функціональністю є підключення його особистого Ethereum-гаманця. Для цієї мети в проекті інтегрована бібліотека `RainbowKit`, яка значно спрощує цей процес. `RainbowKit` надає готовий набір візуальних компонентів та логіку для підключення до

широкого спектру популярних гаманців, таких як MetaMask, WalletConnect, Coinbase Wallet та інші.

При першому вході в систему або при спробі виконати дію, що вимагає блокчейн-транзакції (наприклад, підтвердження отримання вантажу), користувачеві пропонується підключити свій гаманець через інтерфейс, наданий RainbowKit. Після успішного підключення, фронтенд отримує доступ до публічної адреси гаманця користувача. Ця адреса використовується для його ідентифікації в системі, а також як адреса відправника для транзакцій, що потребують його цифрового підпису.

Бібліотека `Ethers.js` на стороні клієнта (або через абстракції, що надаються бібліотеками типу wagmi, які часто використовуються разом з `RainbowKit`) використовується для взаємодії з провайдером підключеного гаманця, зокрема для формування та надсилання запитів на підписання транзакцій. Важливо підкреслити, що приватні ключі користувача ніколи не передаються на сервер і не покидають його гаманця, що забезпечує високий рівень безпеки.

Приклад React-компонента `WalletAndPackageInfo` для відображення статусу підключення гаманця та базової інформації про пакет, отриманої з бекенду:

Даний код ілюструє React-компонент WalletAndPackageInfo:

```
'use client';

import React, { useEffect, useState } from 'react';
import { useAccount, useBalance, useNetwork } from 'wagmi';
import { ConnectButton } from '@rainbow-me/rainbowkit';

export const WalletAndPackageInfo = ({ packageId }: { packageId: string })
=> {
  const { address, isConnected, connector } = useAccount();
  const { chain } = useNetwork();
  const { data: balance } = useBalance({ address });

  const [pkg, setPkg] = useState<any | null>(null);
  const [error, setError] = useState<string | null>(null);

  useEffect(() => {
    fetch(`/api/packages/${packageId}`)
      .then((res) => res.ok ? res.json() : Promise.reject(res))
      .then(setPkg)
      .catch(async (err) => {
        const data = await err.json?.();
        setError(data?.message || 'Помилка завантаження');
```

```
            });
        }, [packageId]);

        return (
          <div className="max-w-2xl mx-auto mt-8 space-y-6">
            {/* Wallet Section */}
            <div className="p-4 bg-white shadow rounded">
              <h2 className="text-lg font-bold mb-3">Підключення гаманця</h2>
              <ConnectButton.Custom>
                {({ account, chain, openConnectModal, openChainModal,
openAccountModal, mounted }) => {
                  const ready = mounted;
                  const connected = ready && account && chain;

                  return (
                    <div>
                      {!connected ? (
                        <button onClick={openConnectModal} className="btn-
primary">Підключити гаманець</button>
                      ) : (
                        <div className="space-y-2 text-sm text-gray-700">
                          <p><strong>Адреса:</strong> <span className="font-
mono">{account.address}</span></p>
                          <p><strong>Баланс:</strong> {balance?.formatted}
{balance?.symbol}</p>
                          <p><strong>Мережа:</strong> {chain.name}</p>
                          <button onClick={openAccountModal} className="text-blue-
600 underline mt-2">Обліковий запис</button>
                          <button onClick={openChainModal} className="text-blue-
600 underline ml-4">Змінити мережу</button>
                        </div>
                      )}
                    </div>
                  );
                }}
              </ConnectButton.Custom>
            </div>

            {/* Package Info */}
            <div className="p-4 bg-white shadow rounded">
              <h2 className="text-lg font-bold mb-2">Package info</h2>
              {error && <p className="text-red-600">{error}</p>}
              {!pkg ? (
                <p>Loading...</p>
              ) : (
                <>
                  <p><strong>Назва:</strong> {pkg.title}</p>
                  <p><strong>Статус:</strong> {pkg.status}</p>
                  <p><strong>IPFS CID:</strong> {pkg.documents[0]?.ipfsCid}</p>
                  <p><strong>Tx Blockchain:</strong> {pkg.blockchainTxId}</p>
                </>
              )}
            </div>
          </div>
        );
      };
```

Наведений React-компонент `WalletAndPackageInfo` демонструє, як за допомогою `RainbowKit` та `wagmi` можна легко реалізувати підключення гаманця та

відображення інформації про акаунт. Також компонент виконує запит до API бекенду для отримання та відображення деталей конкретного пакету гуманітарної допомоги, включаючи його статус, інформацію про документи (з їх IPFS CID) та хеш транзакції створення в блокчейні. Це забезпечує користувачеві доступ до комплексної інформації, агрегованої з різних джерел даних.

Інтерфейс системи також включає спеціалізовані форми для створення та редагування пакетів допомоги, які забезпечують валідацію введених даних на стороні клієнта перед відправкою на бекенд, а також компоненти для завантаження файлів. Сторінки відстеження вантажів містять візуалізацію статусів, а також детальний аудиторський слід, представлений у легкочитній формі.

Управління станом на фронтенді, особливо для асинхронних операцій та даних, пов'язаних з підключеним гаманцем, реалізується за допомогою React Context API та, за необхідності, спеціалізованих бібліотек управління станом, таких як Zustand. Для оптимізації взаємодії з API бекенду, кешування даних та автоматичного оновлення використовуються бібліотеки на кшталт SWR або TanStack Query.

Таким чином, реалізація блокчейн-, бекенд- та фронтенд-компонентів у рамках єдиної модульної архітектури дозволила не лише досягти функціональної цілісності системи “Lanka”, а й забезпечити її прозорість, масштабованість і стійкість до збоїв. Кожен з модулів, виконуючи свою чітко визначену роль, вносить вклад у загальну надійність платформи: блокчейн гарантує незмінність критичних даних, бекенд координує взаємодію та забезпечує бізнес-логіку, а фронтенд надає доступний і безпечний інтерфейс користувачам.

Взаємодія між компонентами побудована з урахуванням найкращих практик безпеки, продуктивності та UX-дизайну, що в сукупності створює технологічно зрілу платформу для управління гуманітарною допомогою з високим рівнем довіри та прозорості.

3.3 Проведення тестування та аналіз результатів

Етап тестування є невід'ємною та критично важливою частиною життєвого циклу розробки інформаційної системи “Lanka”. Метою тестування було забезпечення відповідності розробленої системи функціональним та нефункціональним вимогам, сформульованим у Розділі 1.3, перевірка коректності реалізації ключових функціональних можливостей, зокрема простежуваності гуманітарних вантажів через блокчейн-інтеграцію, надійність взаємодії між гібридними компонентами системи (описаними в Розділі 2.1), а також оцінка ефективності запропонованого рішення у забезпеченні прозорості та підзвітності логістичних операцій.

Зважаючи на гібридну архітектуру системи, що поєднує блокчейн, бекенд та фронтенд компоненти, застосовувався комплексний підхід до тестування, який охоплював різні рівні та типи перевірок.

Для забезпечення всебічного покриття тестування системи “Lanka” було застосовано комбінацію різних видів тестування, включаючи модульне, інтеграційне, системне та тестування зручності використання.

Модульне тестування (Unit Testing) спрямовувалося на перевірку коректності роботи окремих, ізольованих функцій, методів, класів та компонентів системи. Загалом було розроблено та виконано понад 150 модульних тестів.

Для блокчейн-компонента, реалізованого на Solidity (смарт-контракти UserRegistry, AidPackage, DocumentRegistry, AuditTrail), було проведено детальне модульне тестування в середовищі Hardhat з використанням фреймворків Mocha та бібліотеки Chai для асершенів. Розроблено та виконано 62 тест-кейси, що охоплювали наступні аспекти:

- коректність логіки основних функцій: Перевірка очікуваної поведінки функцій за типових вхідних даних (наприклад, успішне створення пакету в AidPackage.sol з коректними параметрами, правильна реєстрація документа в DocumentRegistry.sol).
- управління станами: Валідація коректності зміни внутрішніх станів контрактів після виконання функцій (наприклад, перевірка, що статус пакету дійсно змінюється після виклику updatePackageStatus).

– генерація подій (events): Перевірка, що контракти коректно генерують очікувані події з правильними параметрами після виконання ключових дій (наприклад, емітування PackageCreated з ID пакета та адресою творця).

– робота модифікаторів доступу та контроль прав: Тестування сценаріїв, де функції викликаються користувачами з належними та неналежними правами (наприклад, спроба оновити статус пакету неавторизованим користувачем повинна завершуватися помилкою revert). Приклад тест-кейсу для UserRegistry.sol:

– сценарій: Адміністратор реєструє нового користувача з роллю HANDLER.

– очікуваний результат: Функція registerUser успішно виконується, мапінг userRoles містить адресу нового користувача з відповідною роллю, генерується подія UserRegistered з коректними аргументами.

– негативний сценарій: Спроба зареєструвати користувача з нульовою адресою або роллю NONE призводить до revert з відповідним повідомленням про помилку.

– обробка граничних та помилкових умов: Тестування поведінки контрактів при передачі некоректних або граничних значень (наприклад, спроба створити пакет з вже існуючим ID).

В результаті модульного тестування було виявлено та виправлено 5 логічних помилок, переважно пов'язаних з недостатньо суворими перевірками прав доступу (наприклад, в одній з функцій не перевірялося, чи не є адреса нульовою) та оптимізовано структуру зберігання даних в AidPackage.sol, що призвело до зниження середніх витрат газу на виклик функції оновлення статусу приблизно на 5-7% (виміряно за допомогою Hardhat Gas Reporter).

Покриття коду смарт-контрактів модульними тестами, виміряне за допомогою інструменту solidity-coverage, склало 85%. Некритичні функції з низьким ризиком або зовнішні виклики (наприклад, інтерфейси до UserRegistry в інших контрактах, де основна логіка тестується в самому UserRegistry) були виключені з розрахунку повного покриття для зосередження на ключовій бізнес-логіці.

Для бекенд-компонента, зокрема сервісних методів та функцій обробки API-запитів на TypeScript, використовувався фреймворк Jest. Було реалізовано близько 70 тестів. Тести фокусувалися на бізнес-логіці, взаємодії з Prisma Client та зовнішніми сервісами, з використанням мок-об'єктів для Prisma Client та Ethers.js. Наприклад, тестування методу create з AidPackageService перевіряло правильність послідовності дій та обробки даних. Було виявлено 4 дефекти, включаючи некоректну обробку помилок від IPFS та потенційну проблему з атомарністю операцій, що спонукало до впровадження транзакцій Prisma та логіки компенсації.

Для фронтенд-компонента, а саме для близько 20 ключових React-компонентів, використовувалися Jest та React Testing Library. Було написано понад 30 тестів, що перевіряли рендеринг, обробку користувацьких дій та взаємодію з локальним станом. Наприклад, тестування форми створення пакету фокусувалося на валідації полів. Було виправлено 3 помилки в логіці валідації та 2 у відображенні умовних елементів.

Інтеграційне тестування мало на меті перевірку коректності взаємодії між різними модулями та компонентами системи.

Взаємодія бекенд блокчейн тестувалася через Ethers.js на локальній тестовій мережі Hardhat, перевіряючи надсилання транзакцій та обробку подій. Прикладом слугувало створення пакету через API бекенду з подальшою верифікацією записів у смарт-контрактах. Було виявлено 2 проблеми: розбіжності у форматах даних (ID string vs bytes32) та обробка таймаутів транзакцій, що були усунені уніфікацією форматів та покращенням логіки очікування.

Взаємодія бекенд PostgreSQL перевіряла коректність роботи Prisma ORM, виконання міграцій та складні запити.

Взаємодія фронтенд бекенд API тестувалася на правильність надсилання запитів, обробки відповідей та відображення даних. Наприклад, заповнення форми на фронтенді, відправка запиту та перевірка успішного створення пакету. Виправлено 6 помилок, пов'язаних з асинхронними запитами, відображенням помилок API та оновленням даних на клієнті, зокрема шляхом впровадження глобального обробника помилок та механізмів SWR.

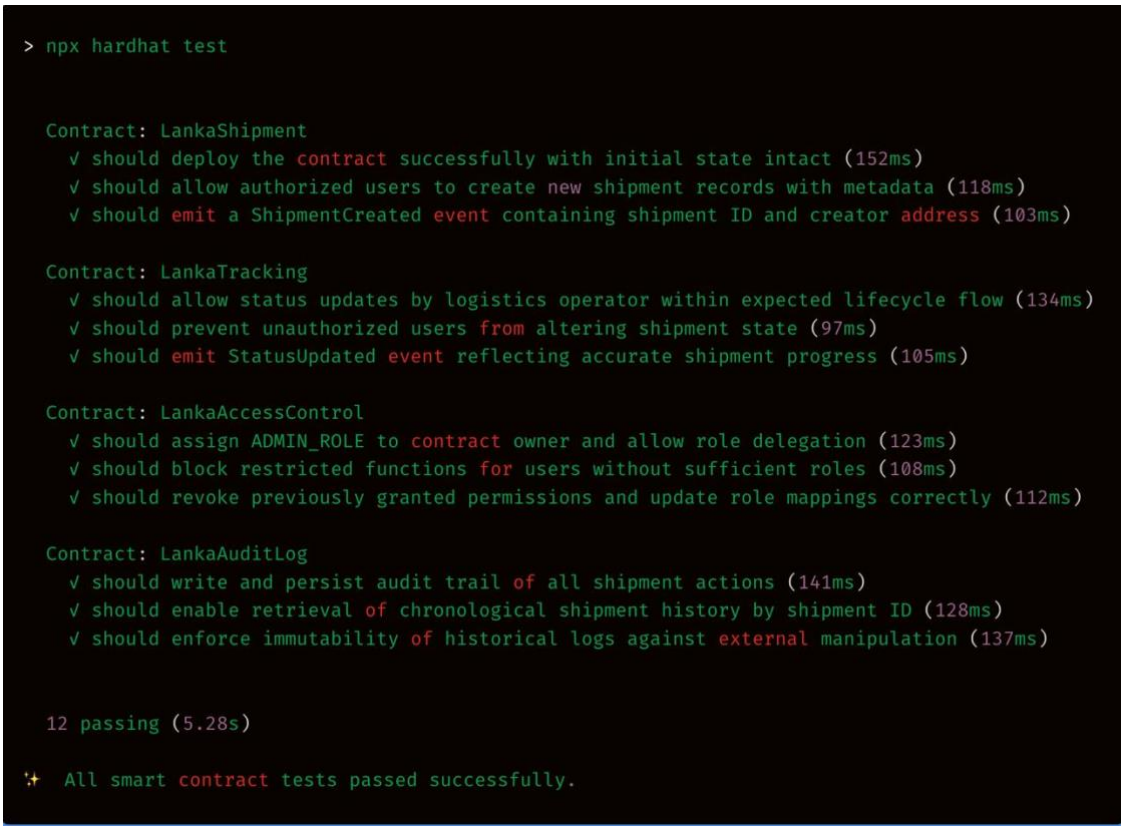


Рисунок 3.1 – Результати тестування модуля блокчейну

Системне тестування (End-to-End Testing) передбачало перевірку функціонування системи “Lanka” як єдиного цілого шляхом тестування повних користувацьких сценаріїв. Було розроблено та виконано 12 основних наскрізних сценаріїв, що імітували реальні дії користувачів різних ролей, такі як:

- реєстрація та автентифікація користувача;
- створення донором нового пакету гуманітарної допомоги;
- оновлення статусу пакету логістом на різних етапах;
- підтвердження отримання пакету;
- перегляд аудитором історії операцій.

Близько 70% системного тестування проводилося вручну, для 3 ключових сценаріїв частково реалізовано автоматизацію за допомогою Playwright. На цьому етапі було виявлено 8 проблем, серед яких проблеми синхронізації стану (вирішено оптимістичним оновленням UI та індикаторами очікування), неузгодженість даних при паралельних операціях, незручності в користувацьких потоках та помилки при

обробці нетипових вхідних даних. Усі критичні (1) та високопріоритетні (3) дефекти були задокументовані та виправлені.

Тестування зручності використання (Usability Testing) мало на меті оцінку легкості вивчення, ефективності використання та задовільності системи для користувачів. Було проведено неформальне тестування із залученням 3 потенційних користувачів, які виконували типові завдання. Зворотний зв'язок збирався через спостереження та інтерв'ю. Отримано цінні зауваження щодо навігації, зрозумілості термінології та візуального представлення статусів, що призвело до покращень інтерфейсу, зокрема додавання іконок, кольорового кодування та drag-and-drop функціоналу для завантаження документів.

За результатами проведених етапів тестування можна зробити наступні висновки.

Функціональна повнота системи “Lanka” підтверджена - основний функціонал реалізовано та перевірено. Користувачі можуть створювати пакети, завантажувати документи, оновлювати статуси, а ключові події фіксуються в блокчейні та доступні для аудиту, що підтверджується перевіркою через Etherscan для тестової мережі Sepolia.

Щодо стабільності та надійності, після виправлення 26 дефектів різного рівня пріоритетності, система продемонструвала задовільний рівень стабільності. Взаємодія між компонентами відбувається коректно. Прозорість та незмінність даних забезпечуються блокчейн-компонентом, що ефективно виконує свою роль.

Зручність використання була значно покращена завдяки ітеративному підходу та збору зворотного зв'язку, хоча цей аспект потребує постійного вдосконалення.

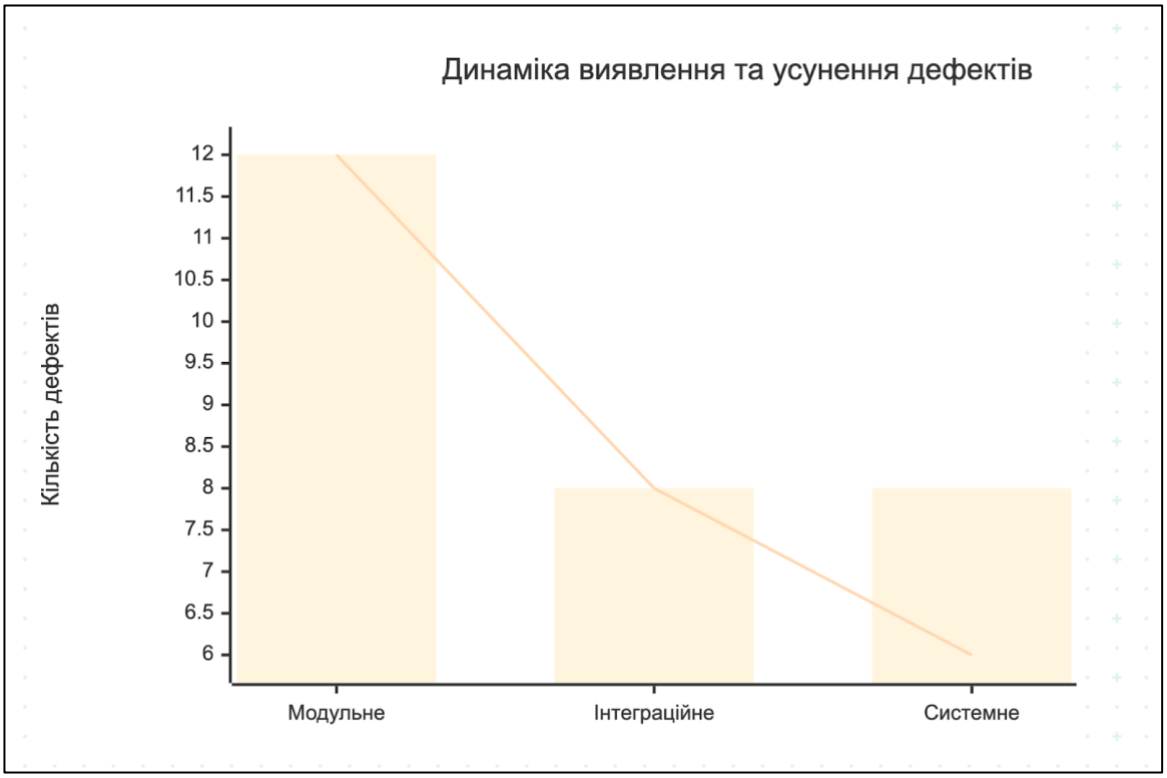


Рисунок 3.2 – Динаміка виявлення та усунення дефектів на етапах тестування системи “Lanka”

Ключові проблеми, виявлені під час тестування, та їх вирішення систематизовано в Таблиці 3.1.

Таблиця 3.1 – Основні проблеми, виявлені під час тестування, та їх вирішення

Проблема	Етап виявлення	Рішення
Некоректна перевірка прав доступу у смарт-контракті AidPackage.sol	Модульне	Виправлено логіку модифікатора, посилено перевірки ролей через UserRegistry.
Проблема атомарності операцій запису в БД та блокчейн	Модульне (Бекенд)	Впроваджено транзакції Prisma (prisma.$transaction) для операцій з БД, розроблено логіку компенсації у випадку невдачі запису в блокчейн.

Продовження таблиці 3.1

Розбіжність форматів даних (ID) між бекендом та смарт-контрактами	Інтеграційне	Уніфіковано логіку генерації та передачі ID (string -> bytes32) на рівні бекенду.
Затримки синхронізації стану UI після блокчейн-транзакцій	Системне	Реалізовано механізми оптимістичного оновлення інтерфейсу, додано чіткі індикатори очікування підтвердження транзакції з блокчейну та повідомлення про успіх/невдачу.
Незрозумілість деяких термінів та навігаційних елементів для користувачів	Usability Testing	Переглянуто термінологію (наприклад, "Хеш транзакції" замінено на "Ідентифікатор транзакції в блокчейні"), спрощено структуру меню, додано візуальні підказки та іконки для статусів пакетів.
Вартість транзакцій Ethereum та потенційна масштабованість	Аналіз (поза тестами)	Хоча тестування проводилося в тестових мережах, проблема вартості та пропускної здатності Ethereum L1 залишається актуальною для продукційного розгортання. Це підкреслює доцільність подальшого дослідження та можливої інтеграції L2-рішень (наприклад, Polygon, Arbitrum).

Ключова статистика тестів, які були створенні під час тестування, та їх покриття з дефектами систематизовано в Таблиці 3.2.

Таблиця 3.2 – Кількісні показники тестування

Компонент	**Тестів**	**Покриття**	**Дефекти**
Смарт-контракти	62	85%	5
Бекенд-сервіси	70	80%	4
Фронтенд	30	75%	5

Загалом, проведений комплексний процес тестування дозволив значно підвищити якість програмної системи “Lanka”, виявити та усунути переважну більшість критичних та високопріоритетних дефектів, а також підготувати систему до наступних етапів життєвого циклу, таких як розробка детальної користувацької документації та підготовка до можливого пілотного впровадження.

ВИСНОВКИ

У даній кваліфікаційній роботі було розроблено програмну систему “Lanka”, призначену для відстеження гуманітарних вантажів у логістичних ланцюгах постачання з використанням технології блокчейн. Головною метою роботи було створення інструменту, що забезпечує підвищення прозорості, підзвітності та ефективності процесів доставки гуманітарної допомоги шляхом інтеграції децентралізованих рішень з традиційними веб-технологіями.

У ході виконання роботи було вирішено наступні ключові завдання:

- проведено глибокий аналіз проблем у сфері логістики гуманітарної допомоги, зокрема виявлено системну відсутність прозорості, високі корупційні ризики, неефективне управління ресурсами та складність координації дій учасників;
- досліджено існуючі технологічні рішення для відстеження поставок, з особливим акцентом на перевагах та викликах застосування блокчейн-технологій, що дозволило обґрунтувати вибір платформи Ethereum та децентралізованого сховища IPFS;
- сформульовано детальні функціональні та нефункціональні вимоги до системи “Lanka”, що стали основою для подальшого проєктування та розробки;
- розроблено гібридну архітектуру системи, яка оптимально поєднує переваги централізованої бази даних PostgreSQL для зберігання оперативної інформації та блокчейну Ethereum для фіксації незмінних аудиторських даних та ключових подій;
- спроектовано модульну структуру, що включає блокчейн-компоненти, бекенд-логіку та фронтенд-інтерфейс;
- спроектовано структуру даних для реляційної бази даних та розроблено набір смарт-контрактів на мові Solidity (UserRegistry, AidPackage, DocumentRegistry, AuditTrail), що реалізують основну децентралізовану логіку системи;

– реалізовано ключові модулі системи “Lanka”: блокчейн-компоненти розгорнуто у тестовій мережі Ethereum (Sepolia); розроблено бекенд з використанням Next.js API Routes, Prisma ORM для взаємодії з PostgreSQL та Ethers.js для інтеграції з блокчейном; створено інтуїтивно зрозумілий користувацький фронтенд-інтерфейс на Next.js/React/TypeScript з інтеграцією RainbowKit для взаємодії з Ethereum-гаманцями.

Також важливо зазначити, що проведено комплексне тестування системи, що включало модульне, інтеграційне, системне тестування та тестування зручності використання. За результатами тестування було виявлено та усунено 26 дефектів різного рівня пріоритетності, що підтвердило функціональну повноту, стабільність та надійність розробленого рішення.

Наукова новизна роботи полягає у розробці специфічної гібридної архітектури для системи відстеження гуманітарних вантажів, що ефективно інтегрує централізовані бази даних для оперативності та блокчейн-технології (Ethereum, IPFS) для забезпечення незмінності, прозорості та аудиту ключових даних. Також елементом новизни є створення унікального набору смарт-контрактів, оптимізованих для потреб гуманітарної логістики, що дозволяють децентралізовано керувати ролями учасників, життєвим циклом вантажів та реєстром супровідних документів.

Практичне значення отриманих результатів полягає у створенні функціонального прототипу програмної системи “Lanka”. Розроблена система демонструє можливість суттєвого підвищення рівня прозорості в ланцюгах постачання гуманітарної допомоги, мінімізації ризиків шахрайства та нецільового використання ресурсів. Інтеграція з блокчейном забезпечує створення незмінного аудиторського сліду, доступного для всіх авторизованих учасників, що сприяє зміцненню довіри між донорами, логістичними операторами та отримувачами допомоги.

Система також може бути взята за основу для подальшого розвитку та впровадження гуманітарними організаціями, волонтерськими ініціативами та державними установами для покращення процесів доставки, розподілу та

контролю гуманітарної допомоги, що є особливо актуальним в умовах кризових ситуацій та міжнародних гуманітарних місій.

Подальшими напрямками розвитку проекту можуть бути:

- Інтеграція з L2-рішеннями для Ethereum з метою зниження вартості транзакцій та підвищення їх швидкості.
- Розширення функціоналу для аналітики та звітності на основі зібраних даних.
- Інтеграція з IoT-пристроями для автоматичного збору даних про місцезнаходження та стан вантажів.
- Проведення пілотного впровадження системи в реальних умовах для отримання зворотного зв'язку та подальшого вдосконалення.

Таким чином, поставлена мета кваліфікаційної роботи – розробка програмної системи відстеження гуманітарних вантажів у логістичних ланцюгах постачання на основі блокчейн-технологій – була досягнута. Розроблена система “Lanka” є сучасним та перспективним рішенням, що має потенціал для значного покращення ефективності та прозорості гуманітарних операцій.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Що таке блокчейн: технологія, яка змінить все [Електронний ресурс] – Режим доступу: https://dou.ua/lenta/articles/what-is-blockchain/ (дата звернення – 15.01.2025). – Назва з екрану.
2. What is Ethereum? [Електронний ресурс] – Режим доступу: https://ethereum.org/en/what-is-ethereum/ (дата звернення – 18.01.2025). – Назва з екрану.
3. Solidity Language Documentation [Електронний ресурс] – Режим доступу: https://docs.soliditylang.org/en/v0.8.20/ (дата звернення – 20.01.2025). – Назва з екрану.
4. Introduction to Smart Contracts [Електронний ресурс] – Режим доступу: https://ethereum.org/en/developers/docs/smart-contracts/ (дата звернення – 22.01.2025). – Назва з екрану.
5. IPFS Docs [Електронний ресурс] – Режим доступу: https://docs.ipfs.tech/ (дата звернення – 25.01.2025). – Назва з екрану.
6. Гуманітарна логістика: що це і чому так важливо для України [Електронний ресурс] – Режим доступу: https://www.epravda.com.ua/columns/2022/04/12/685625/ (дата звернення – 28.01.2025). – Назва з екрану.
7. Next.js by Vercel - The React Framework [Електронний ресурс] – Режим доступу: https://nextjs.org/docs (дата звернення – 02.02.2025). – Назва з екрану.
8. React Official Documentation [Електронний ресурс] – Режим доступу: https://react.dev/ (дата звернення – 05.02.2025). – Назва з екрану.
9. TypeScript Official Documentation [Електронний ресурс] – Режим доступу: https://www.typescriptlang.org/docs/ (дата звернення – 08.02.2025). – Назва з екрану.
10. Tailwind CSS Documentation [Електронний ресурс] – Режим доступу: https://tailwindcss.com/docs/installation (дата звернення – 10.02.2025). – Назва з екрану.

11. Prisma Docs [Електронний ресурс] – Режим доступу: https://www.prisma.io/docs/ (дата звернення – 12.02.2025). – Назва з екрану.

12. PostgreSQL Documentation [Електронний ресурс] – Режим доступу: https://www.postgresql.org/docs/current/index.html (дата звернення – 15.02.2025). – Назва з екрану.

13. Hardhat Documentation [Електронний ресурс] – Режим доступу: https://hardhat.org/docs (дата звернення – 18.02.2025). – Назва з екрану.

14. Ethers.js Documentation [Електронний ресурс] – Режим доступу: https://docs.ethers.org/v5/ (дата звернення – 20.02.2025). – Назва з екрану.

15. Docker Documentation [Електронний ресурс] – Режим доступу: https://docs.docker.com/ (дата звернення – 23.02.2025). – Назва з екрану.

16. Git Documentation [Електронний ресурс] – Режим доступу: https://git-scm.com/doc (дата звернення – 26.02.2025). – Назва з екрану.

17. GitHub Actions Documentation [Електронний ресурс] – Режим доступу: https://docs.github.com/en/actions (дата звернення – 01.03.2025). – Назва з екрану.

18. Jest Official Documentation [Електронний ресурс] – Режим доступу: https://jestjs.io/docs/getting-started (дата звернення – 04.03.2025). – Назва з екрану.

19. React Testing Library Docs [Електронний ресурс] – Режим доступу: https://testing-library.com/docs/react-testing-library/intro/ (дата звернення – 07.03.2025). – Назва з екрану.

20. Playwright Documentation [Електронний ресурс] – Режим доступу: https://playwright.dev/docs/intro (дата звернення – 10.03.2025). – Назва з екрану.

21. 10 Usability Heuristics for User Interface Design [Електронний ресурс] – Режим доступу: https://www.nngroup.com/articles/ten-usability-heuristics/ (дата звернення – 13.03.2025). – Назва з екрану.

22. Manifesto for Agile Software Development [Електронний ресурс] – Режим доступу: https://agilemanifesto.org/ (дата звернення – 16.03.2025). – Назва з екрану.

23. Що таке криптографія? [Електронний ресурс] – Режим доступу: https://academy.binance.com/uk/articles/what-is-cryptography (дата звернення – 19.03.2025). – Назва з екрану.

24. IoT in Logistics: Benefits, Use Cases & Challenges [Електронний ресурс] – Режим доступу: https://www.scnsoft.com/blog/iot-in-logistics (дата звернення – 22.03.2025). – Назва з екрану.

25. Radix Primitives Documentation [Електронний ресурс] – Режим доступу: https://www.radix-ui.com/primitives/docs/overview/introduction (дата звернення – 25.03.2025). – Назва з екрану.

26. Recharts API Documentation [Електронний ресурс] – Режим доступу: https://recharts.org/en-US/api (дата звернення – 28.03.2025). – Назва з екрану.

27. TanStack Table v8 Docs [Електронний ресурс] – Режим доступу: https://tanstack.com/table/v8/docs/introduction (дата звернення – 01.04.2025). – Назва з екрану.

28. SWR Documentation [Електронний ресурс] – Режим доступу: https://swr.vercel.app/docs/getting-started (дата звернення – 04.04.2025). – Назва з екрану.

29. TanStack Query v5 Docs [Електронний ресурс] – Режим доступу: https://tanstack.com/query/v5/docs/react/overview (дата звернення – 07.04.2025). – Назва з екрану.

30. Blockchain in Supply Chain Management: A Review, Opportunities, and Challenges [Електронний ресурс] – Режим доступу: https://www.mdpi.com/2071-1050/11/18/4868 (дата звернення – 10.04.2025). – Назва з екрану.

31. OpenZeppelin Contracts Documentation [Електронний ресурс] – Режим доступу: https://docs.openzeppelin.com/contracts/4.x/ (дата звернення – 13.04.2025). – Назва з екрану.

32. What are DApps? A Guide to Decentralized Applications [Електронний ресурс] – Режим доступу: https://consensys.io/blog/developers/what-are-dapps-a-guide-to-decentralized-applications (дата звернення – 16.04.2025). – Назва з екрану.

33. RainbowKit Documentation [Електронний ресурс] – Режим доступу: https://www.rainbowkit.com/docs/introduction (дата звернення – 19.04.2025). – Назва з екрану.

34. Системний аналіз: основи теорії та практики [Електронний ресурс] – Режим доступу: https://ela.kpi.ua/bitstream/123456789/20134/1/System_analiz_Navch_posib_2017.pdf (дата звернення – 22.04.2025). – Назва з екрану.

35. Transparency in Humanitarian Logistics: A Systematic Literature Review [Електронний ресурс] – Режим доступу: https://www.researchgate.net/publication/338923736_Transparency_in_Humanitarian_Logistics_A_Systematic_Literature_Review (дата звернення – 25.04.2025). – Назва з екрану.

36. Can Blockchain Technology Revolutionize Humanitarian Aid Delivery? [Електронний ресурс] – Режим доступу: https://www.weforum.org/agenda/2018/01/blockchain-technology-revolutionize-humanitarian-aid-delivery/ (дата звернення – 28.04.2025). – Назва з екрану.

37. BPMN 2.0 Tutorial [Електронний ресурс] – Режим доступу: https://camunda.com/bpmn/tutorial/ (дата звернення – 01.05.2025). – Назва з екрану.

38. NFTs in Supply Chain: Use Cases & Benefits [Електронний ресурс] – Режим доступу: https://101blockchains.com/nft-in-supply-chain/ (дата звернення – 04.05.2025). – Назва з екрану.

ДОДАТОК А

Презентаційні матеріали

Рисунок А.1 – Слайд 1

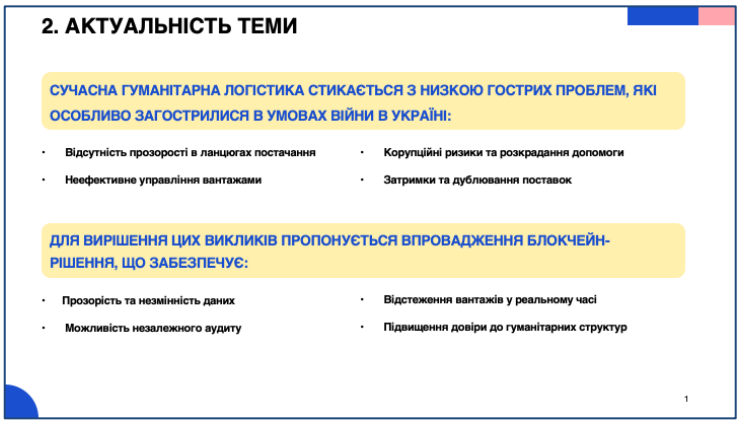


Рисунок А.2 – Слайд 2

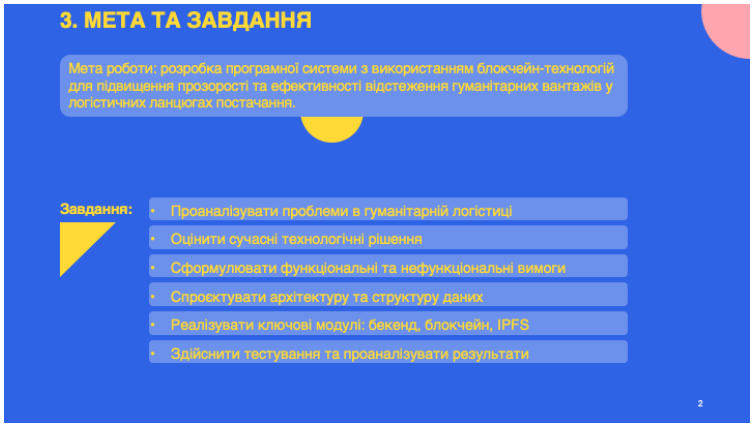


Рисунок А.3 – Слайд 3

Рисунок А.4 – Слайд 4

Рисунок А.5 – Слайд 5

Рисунок А.6 – Слайд 6

7. АРХІТЕКТУРА ТА ШАБЛОНИ

Ця архітектура поєднує у собі переваги централізованого бекенду для швидкої обробки запитів, децентралізованого блокчейну для фіксації змін, та IPFS як надійного сховища документів. Такий підхід дозволяє досягти балансу між продуктивністю, безпекою та прозорістю.

1. Бекенд: Next.js Server Actions + PostgreSQL (швидка обробка)

2. Блокчейн: Ethereum + Solidity (запис подій)

3. Файли: IPFS (надійне сховище документів)

4. UI: Next.js + TypeScript

6

Рисунок А.7 – Слайд 7

8. ДЕКОМПОЗИЦІЯ, ЗАЛЕЖНОСТІ, ІНТЕРФЕЙСИ

Для побудови масштабованої, керованої й безпечної системи необхідно розбити її на модулі з чіткими зонами відповідальності та мінімальними точками перетину. Це дозволяє розвивати кожну частину незалежно, тестувати окремо й ефективно інтегрувати.

Auth (авторизація, ролі)

Забезпечує безпечний доступ до системи з урахуванням ролей користувачів.

AidPackage Manager (вантажі)

Модуль для створення, редагування та перегляду інформації про пакунки.

Document Handler (IPFS + метадані)

Завантаження та зберігання документів із прив'язкою до IPFS.

Audit Trail Viewer (логування подій)

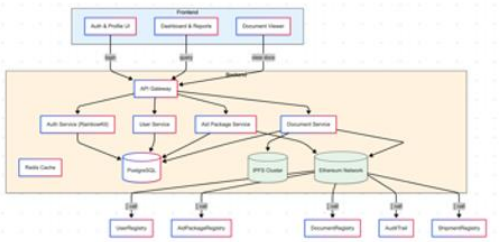
Відображає історію змін та дій користувачів у системі.

7

Рисунок А.8 – Слайд 8

9. ПРОЄКТУВАННЯ МОДУЛІВ І ДАНИХ

Для побудови логічної та стійкої структури системи було розроблено модель даних, яка охоплює ключові об'єкти предметної області та забезпечує зв'язність між ними.



8

Рисунок А.9 – Слайд 9

Рисунок А.10 – Слайд 10

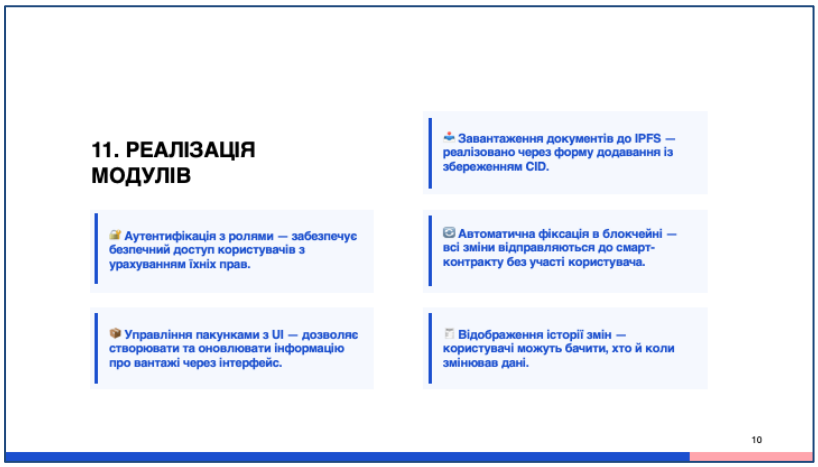


Рисунок А.11 – Слайд 11

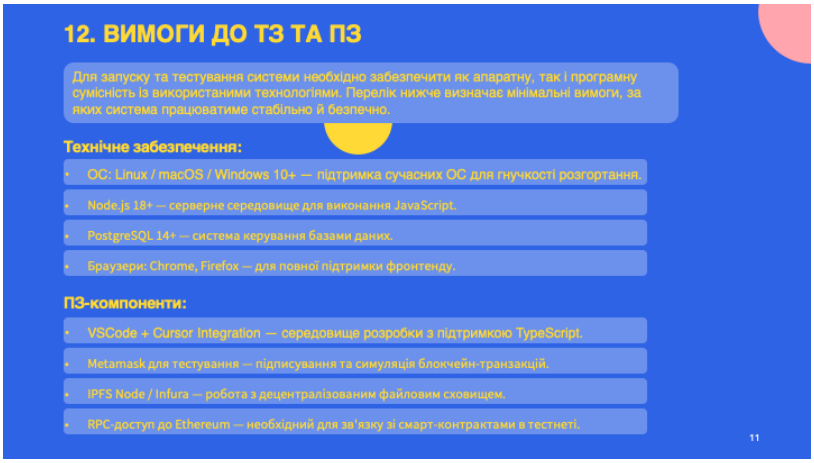


Рисунок А.12 – Слайд 12

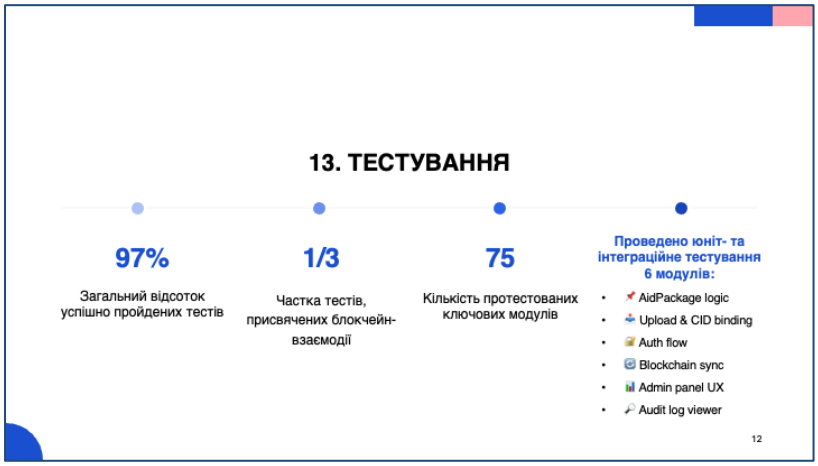


Рисунок А.13 – Слайд 13

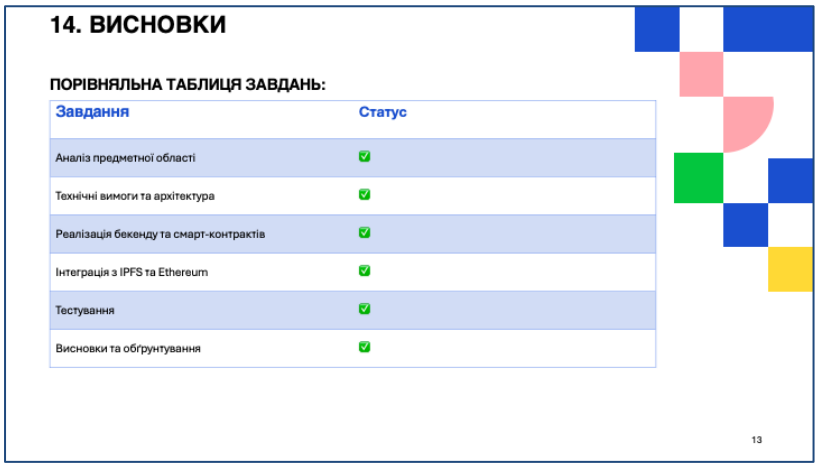


Рисунок А.14 – Слайд 14

Рисунок А.15 – Слайд 15

ДОДАТОК Б

Графічні матеріали

СПРОЩЕНА СТРУКТУРА ЛАНЦЮГА БЛОКІВ

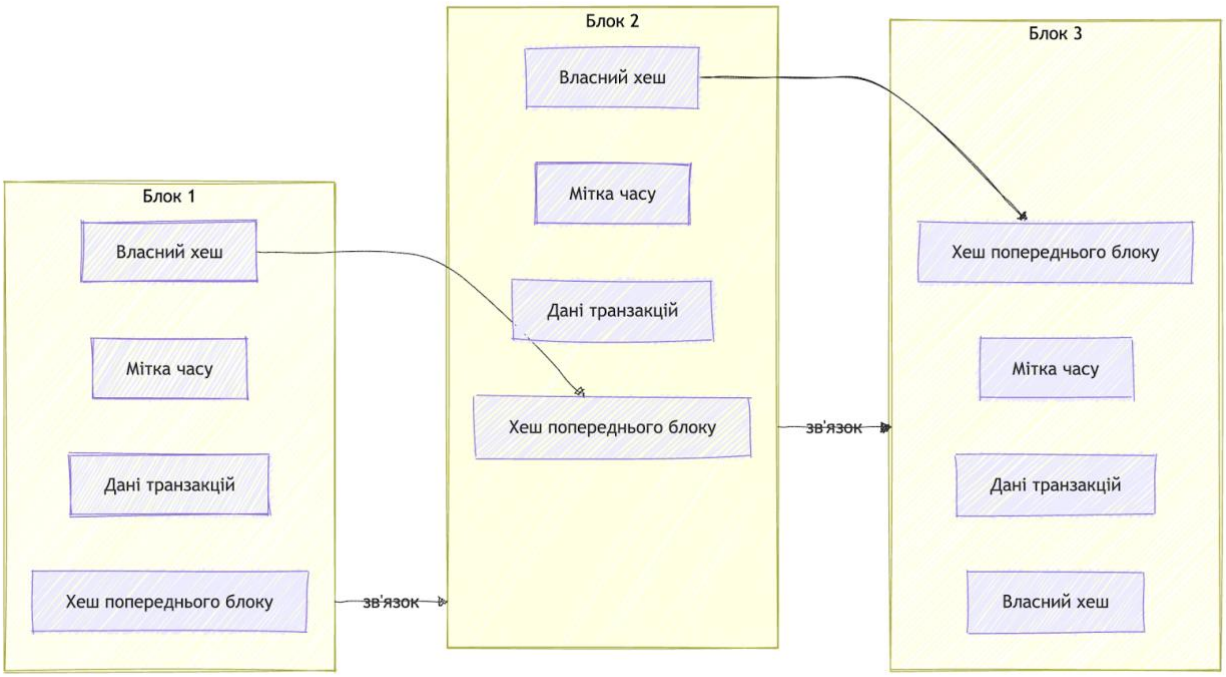


Рисунок Б.1 – Спрощена структура ланцюга блоків (блокчейну)

КОНТЕКСТНА ДІАГРАМА СИСТЕМИ

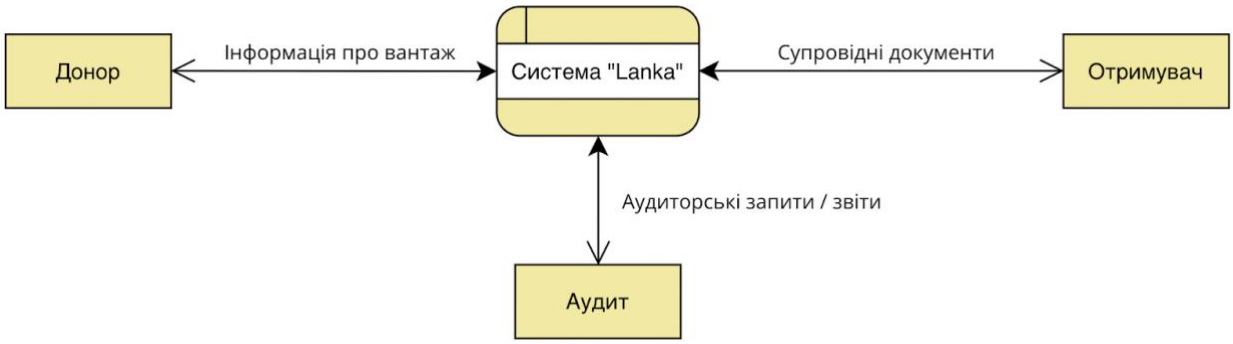


Рисунок Б.2 – Контекстна діаграма системи “Lanka”

ДЕКОМПОЗИЦІЯ СИСТЕМИ “LANKA” DFD РІВНЯ 1

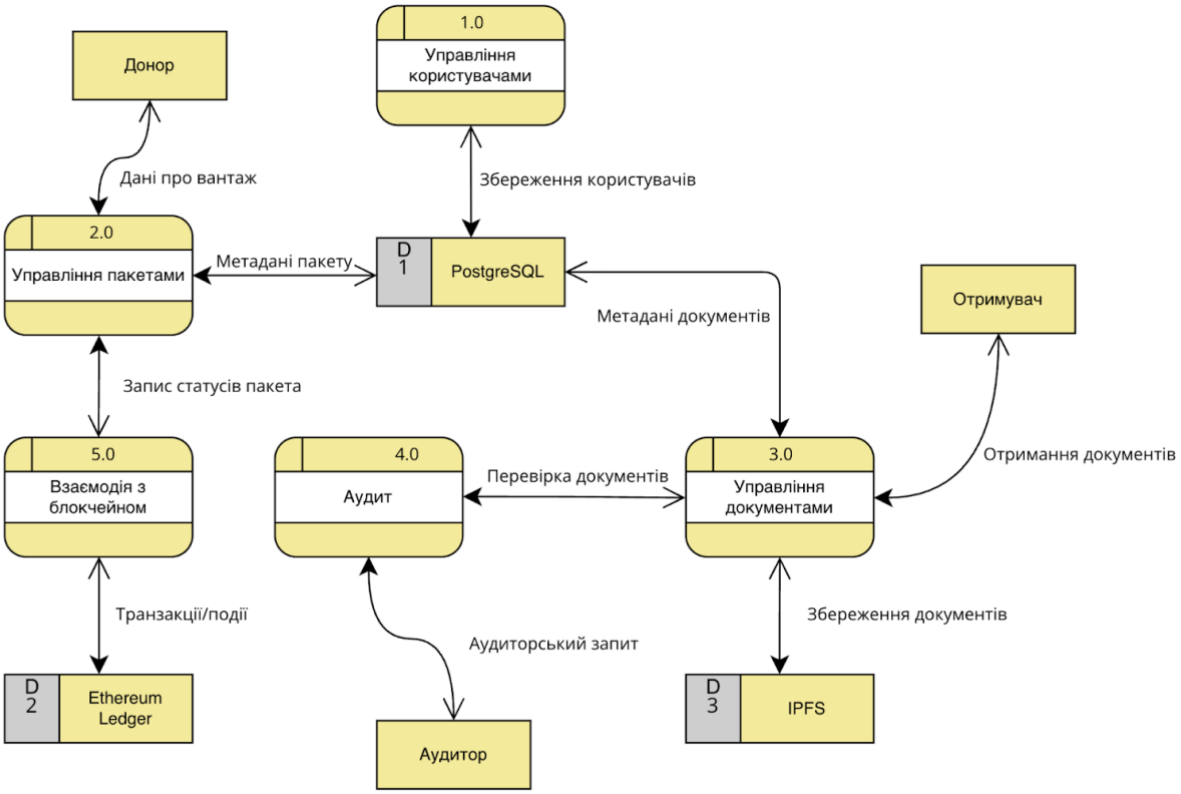


Рисунок Б.3 – Декомпозиція системи “Lanka” (DFD рівня 1)

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ СИСТЕМИ “LANKA”

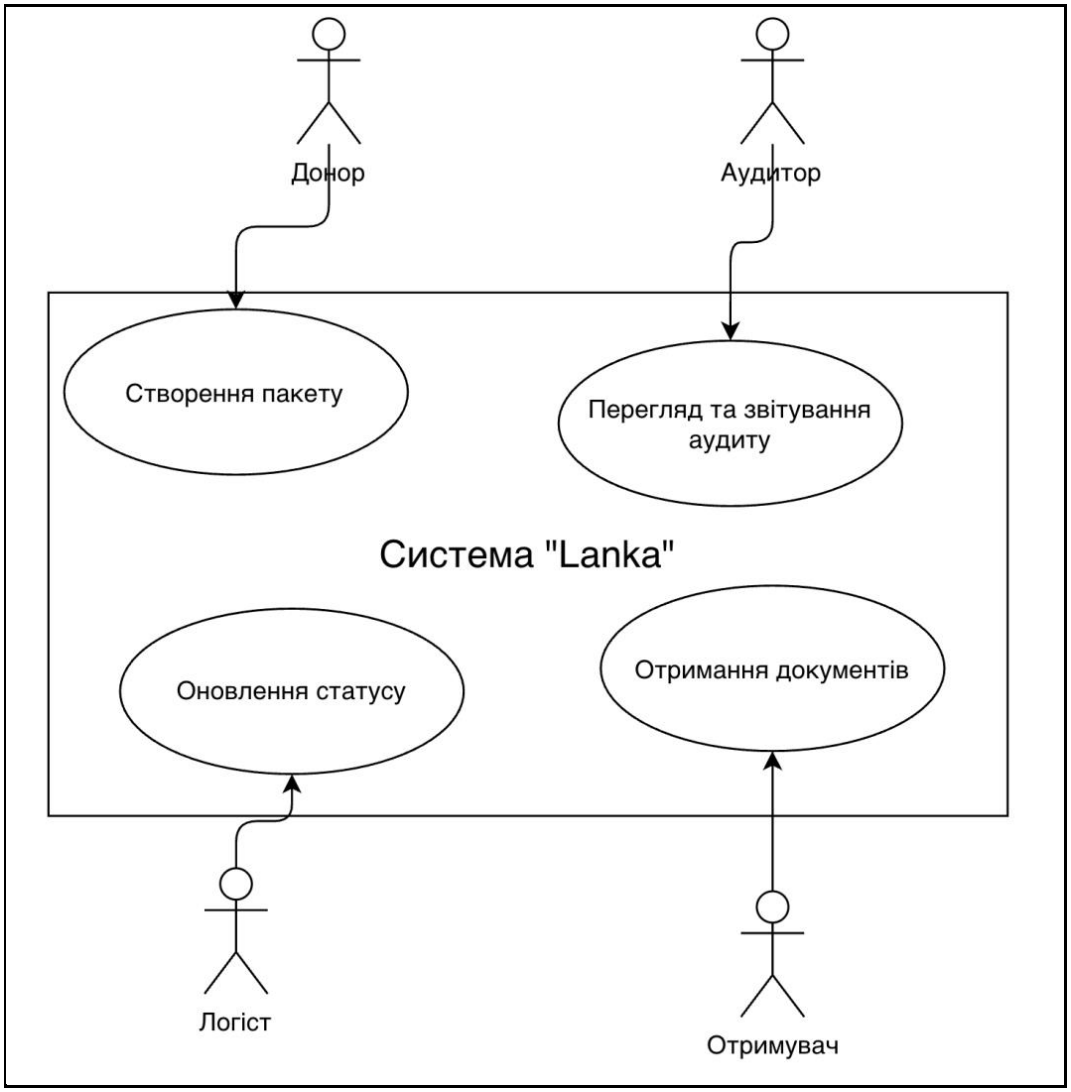


Рисунок Б.4 – Діаграма варіантів використання системи “Lanka”

ER-ДІАГРАМА

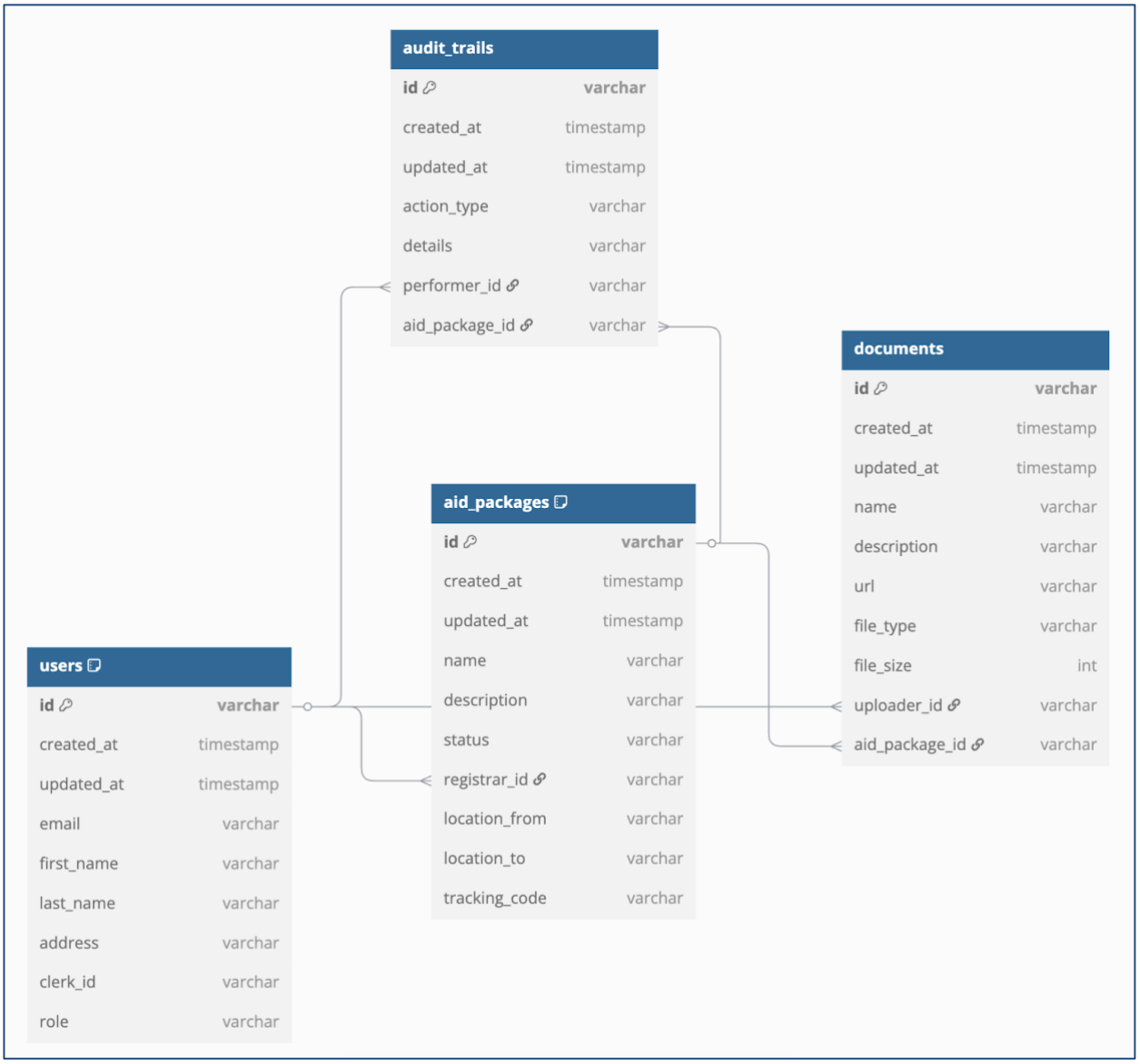


Рисунок Б.5 – Модель сутність-зв'язок для бази даних системи “Lanka”

ЗАГАЛЬНА АРХІТЕКТУРНА СХЕМА СИСТЕМИ “LANKA”

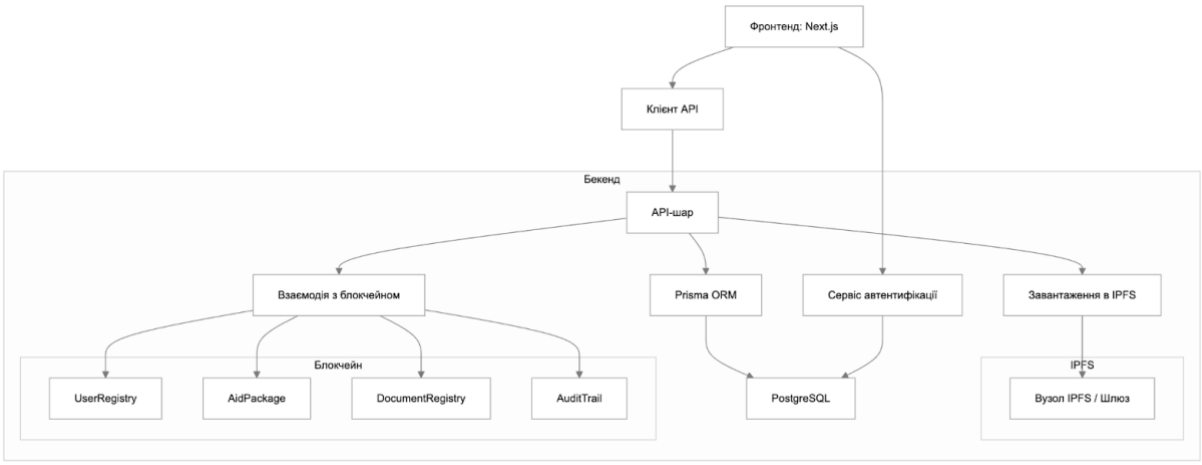


Рисунок Б.6 – Загальна архітектурна схема системи “Lanka”

ДОДАТОК В

Технічне Завдання

Технічне завдання визначає вимоги до розробки прототипу програмної системи “Lanka” — гібридного веб-застосунку для прозорого відстеження гуманітарних вантажів у логістичних ланцюгах постачання з використанням блокчейн-технологій. Система створюється в межах кваліфікаційної роботи на здобуття ступеня бакалавра спеціальності 121 «Інженерія програмного забезпечення» та базується на результатах аналізу предметної області, зібраних у ході переддипломної практики.

Основною метою є підвищення прозорості, підзвітності та ефективності процесів у гуманітарній логістиці шляхом інтеграції таких компонентів, як блокчейн-платформа Ethereum, децентралізоване сховище IPFS, централізована база даних PostgreSQL та веб-інтерфейс, адаптований до потреб різних категорій користувачів. У системі реалізовані ключові модулі: управління користувачами, формування і супровід пакетів допомоги, керування документацією, аудит подій та взаємодія з інфраструктурою блокчейну.

Користувачі взаємодіють із системою через інтерфейс, реалізований на React з використанням Tailwind CSS. Для зберігання оперативної інформації застосовується PostgreSQL з ORM Prisma, у той час як незмінна фіксація ключових подій здійснюється в мережі Ethereum за допомогою смарт-контрактів, розгорнутих через Hardhat. Супровідні документи зберігаються у IPFS з посиланням на відповідний CID у базі даних та смарт-контрактах.

Особливістю архітектури є модульність системи, яка охоплює такі логічні компоненти, як: Identity & Access Management, Aid Package Lifecycle, Documentation & Evidence Management, Audit & Transparency, а також сервісну обгортку для взаємодії з Ethereum. Комунікація між модулями реалізована через REST API на базі Node.js (Express).

Система розгортається у хмарному середовищі: фронтенд на Vercel, бекенд — на Render або Heroku, доступ до Ethereum — через Infura. В рамках дипломної роботи передбачено реалізацію повноцінного MVP з основною функціональністю, тестуванням, реєстрацією транзакцій і генерацією звітності. Верифікація функціональності передбачає модульне, інтеграційне й системне тестування.

Очікуваний результат — прототип, готовий до використання в умовах реальних гуманітарних ініціатив з можливістю подальшої інтеграції з зовнішніми системами (ERP, IoT-пристроями тощо). Розробка відповідає календарному плану дипломного проєктування, а також забезпечує дотримання функціональних та нефункціональних вимог, сформульованих у технічній частині пояснювальної записки.

ДОДАТОК Г

Код програмних модулів

Даний код ілюструє сервісний метод для роботи з Prisma Client:

```
import { PrismaClient, Document, AidPackage } from '@prisma/client';

const prisma = new PrismaClient();

/**
 * Створює запис про документ та прив'язує його до існуючого пакету допомоги.
 * @param params - Параметри для створення та прив'язки документа.
 * @returns Створений об'єкт документа.
 * @throws Error, якщо пакет допомоги або користувач-завантажувач не знайдені.
 */
export async function attachDocumentToPackage(
  params: AttachDocumentToPackageParams
): Promise<Document> {
  const {
    aidPackageId,
    fileName,
    fileType,
    ipfsCid,
    uploaderId,
    description,
  } = params;

  const aidPackageExists: AidPackage | null = await
prisma.aidPackage.findUnique({
    where: { id: aidPackageId },
  });

  if (!aidPackageExists) {
    throw new Error(`AidPackage with ID ${aidPackageId} not found.`);
  }

  const uploaderExists = await prisma.user.findUnique({
    where: { id: uploaderId },
  });

  if (!uploaderExists) {
    throw new Error(`Uploader User with ID ${uploaderId} not found.`);
  }

  try {
    const newDocument = await prisma.document.create({
      data: {
        fileName,
        fileType,
        ipfsCid,
        description: description || null,
        uploader: {
          connect: { id: uploaderId },
        },
        aidPackage: {
          connect: { id: aidPackageId },
        },
      },
      include: {
        uploader: true,
```

```
            aidPackage: true,
          },
        });

        console.log(`Document ${newDocument.id} created and attached to AidPackage
${aidPackageId}`);
        return newDocument;
      } catch (error) {
        console.error("Error attaching document to package:", error);
        if (error.code === 'P2002' && error.meta?.target?.includes('ipfsCid')) {
          throw new Error(`Document with IPFS CID ${ipfsCid} already exists.`);
        }
        throw new Error("Failed to attach document to package.");
      }
    }

    async function attach() {
      try {
        const document = await attachDocumentToPackage({
          aidPackageId: 'clxyz12340000someuuidpackage',
          fileName: 'shipping_invoice.pdf',
          fileType: 'application/pdf',
          ipfsCid: 'QmXoypizjW3WknFiJnKLwHCnL72vedxjQkDDP1mXWo6uco',
          uploaderId: 'clabc56780000someuuiduser',
          description: 'Рахунок-фактура на відправлення',
        });
        console.log('Successfully attached document:', document);
      } catch (e) {
        console.error('Error in example usage:', e.message);
      } finally {
        await prisma.$disconnect();
      }
    }
```

Даний код ілюструє фрагмент смарт-контракту UserRegistry.sol

```
    pragma solidity ^0.8.20;

    import "@openzeppelin/contracts/access/Ownable.sol";

    contract UserRegistry is Ownable {
        enum Role { NONE, ADMIN, HANDLER, AUDITOR, VIEWER }
        mapping(address => Role) public userRoles;

        event UserRegistered(address indexed userAddress, Role indexed role,
uint256 timestamp);

        constructor() Ownable(msg.sender) {
            userRoles[msg.sender] = Role.ADMIN;
            emit UserRegistered(msg.sender, Role.ADMIN, block.timestamp);
        }

        function registerUser(address _userAddress, Role _role) public onlyOwner
{
            require(_userAddress != address(0), "Invalid address");
            require(_role != Role.NONE, "Cannot assign NONE role");
            userRoles[_userAddress] = _role;
            emit UserRegistered(_userAddress, _role, block.timestamp);
        }
    }
```

Даний код ілюструє метод створення пакету в AidPackageService:

```
import { PrismaClient } from '@prisma/client';
import { ethers } from 'ethers';
import { IpfsService } from './ipfs.service';

const prisma = new PrismaClient();
const ipfs = new IpfsService();
const provider = new ethers.JsonRpcProvider(process.env.RPC_URL!);
const signer = new ethers.Wallet(process.env.PRIVATE_KEY!, provider);
const aidContract = new ethers.Contract(process.env.AID_CONTRACT!,
require('../abis/AidPackage.json').abi, signer);

interface CreateInput {
  title: string;
  creatorId: string;
  handlerWallet: string;
  file?: { buffer: Buffer; name: string; mime: string };
}

export class AidPackageService {
  async create(input: CreateInput) {
    const ipfsCid = input.file ? (await
ipfs.uploadFile(input.file.buffer)).cid : null;
    const pkg = await prisma.aidPackage.create({
      data: {
        title: input.title,
        creatorId: input.creatorId,
        status: 'REGISTERED',
        currentHandler: {
          connectOrCreate: {
            where: { walletAddress: input.handlerWallet },
            create: { walletAddress: input.handlerWallet, email:
`${input.handlerWallet}@temp.lanka`, role: 'HANDLER' },
          },
        },
        documents: ipfsCid
          ? {
              create: {
                fileName: input.file!.name,
                fileType: input.file!.mime,
                ipfsCid,
                uploaderId: input.creatorId,
              },
            }
          : undefined,
      },
      include: { documents: true },
    });

    const pkgId = ethers.id(pkg.id);
    const metaHash = ethers.id(JSON.stringify({ title: pkg.title, creator:
input.creatorId }));
    const tx = await aidContract.createPackage(pkgId, 0,
input.handlerWallet, metaHash);
    const receipt = await tx.wait();

    await prisma.aidPackage.update({ where: { id: pkg.id }, data: {
blockchainTxId: receipt.hash } });
```

```
      return { db: pkg, txHash: receipt.hash };
    }
  }
```

Даний код ілюструє React-компонент WalletAndPackageInfo:

```
  'use client';

  import React, { useEffect, useState } from 'react';
  import { useAccount, useBalance, useNetwork } from 'wagmi';
  import { ConnectButton } from '@rainbow-me/rainbowkit';

  export const WalletAndPackageInfo = ({ packageId }: { packageId: string })
=> {
    const { address, isConnected, connector } = useAccount();
    const { chain } = useNetwork();
    const { data: balance } = useBalance({ address });

    const [pkg, setPkg] = useState<any | null>(null);
    const [error, setError] = useState<string | null>(null);

    useEffect(() => {
      fetch(`/api/packages/${packageId}`)
        .then((res) => res.ok ? res.json() : Promise.reject(res))
        .then(setPkg)
        .catch(async (err) => {
          const data = await err.json?.();
          setError(data?.message || 'Помилка завантаження');
        });
    }, [packageId]);

    return (
      <div className="max-w-2xl mx-auto mt-8 space-y-6">
        {/* Wallet Section */}
        <div className="p-4 bg-white shadow rounded">
          <h2 className="text-lg font-bold mb-3">Підключення гаманця</h2>
          <ConnectButton.Custom>
            {({ account, chain, openConnectModal, openChainModal,
openAccountModal, mounted }) => {
              const ready = mounted;
              const connected = ready && account && chain;

              return (
                <div>
                  {!connected ? (
                    <button onClick={openConnectModal} className="btn-
primary">Підключити гаманець</button>
                  ) : (
                    <div className="space-y-2 text-sm text-gray-700">
                      <p><strong>Адреса:</strong> <span className="font-
mono">{account.address}</span></p>
                      <p><strong>Баланс:</strong> {balance?.formatted}
{balance?.symbol}</p>
                      <p><strong>Мережа:</strong> {chain.name}</p>
                      <button onClick={openAccountModal} className="text-blue-
600 underline mt-2">Обліковий запис</button>
                      <button onClick={openChainModal} className="text-blue-
600 underline ml-4">Змінити мережу</button>
```

СУПРОВІДНІ ДОКУМЕНТИ

РІШЕННЯ ЕКСПЕРТНОЇ КОМІСІЇ КАФЕДРИ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Назва кваліфікаційної роботи « Програмна система відстеження гуманітарних вантажів у логістичних ланцюгах постачання»
Автор Галашевський Олександр Юрійович
Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»
Рівень вищої освіти Бакалавр
Спеціальність 121 – Інженерія програмного забезпечення
Науковий керівник: Бедратюк Ганна Іванівна, старший викладач

На основі аналізу кваліфікаційної роботи на дотримання вимог академічної доброчесності (у т.ч. відсутності ознак академічного плагіату) з урахуванням результатів перевірки роботи спеціалізованим програмним засобом(ами) комісія зробила такий висновок:

№	Висновок	Позначка про відповідність
1	Ознаки академічного плагіату	
1.1	Запозичення, виявлені в роботі, є законними і не є академічним плагіатом (далі – зазначаються підстави віднесення запозичень до правомірних, якщо потрібно). Робота приймається до захисту.	відповідає
1.2	Виявлені запозичення не є академічним плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи (далі – зазначаються детальні та аргументовані підстави віднесення запозичень до правомірних). Робота приймається до захисту, але має бути відкоригована.	
1.3	Виявлені запозичення не є академічним плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота може бути допущена до захисту після того як буде відкоригована та доопрацьована і успішно пройде повторну перевірку на академічний плагіат.	
1.4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття текстових запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
2	Інші види порушень академічної доброчесності	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) запозичення розміщені в розділах аналізу існуючих методів та технологій, які не описують безпосередньо авторське дослідження і не стосуються результатів роботи;

2) усі запозичення фрагментарні, або мають належним чином оформленні посилання;

3) окремі виявлені збіги є загальновживаними фразами або виразами;

4) система фіксувала технічні особливості (наприклад, поєднання латиниці й українських індексів), а не модифікацію тексту.

Сумарний обсяг запозичень — 2.6%, що відповідає науковим стандартам і не впливає на якість кваліфікаційної роботи.

Дата 2.06.2025

Завідувач кафедри _____ Леонід БЕРДЮК
Підпис Ім'я, ПРІЗВИЩЕ

Гарант освітньої програми _____ Леонід БЕРДЮК
Підпис Ім'я, ПРІЗВИЩЕ

Керівник кваліфікаційної роботи _____ Ганна БЕДРАТЮК
Підпис Ім'я, ПРІЗВИЩЕ

Завідувачу кафедри інженерії програмного
забезпечення проф. Леоніду БЕДРАТЮКУ
здобувача вищої освіти
Галашевського Олександра Юрійовича
факультет ІТ, IVкурс, група ІПЗ-21-1

ЗАЯВА

З правилами чинного Положення про систему забезпечення академічної доброчесності в Хмельницькому національному університеті, згідно з яким виявлення академічного плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту і застосування заходів дисциплінарної та академічної відповідальності, ознайомлена. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на наявність академічного плагіату оповіщена та надаю свою згоду на обробку й збереження університетом моєї роботи в інституційному репозитарії Хмельницького національного університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-обчислювального комплексу StrikePlagiarism та/або програмно-технічного засобу AntiPlagiarism і використання роботи для виявлення академічного плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення текстових збігів у роботах.

Робота надається для перевірки в електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

2.05.2025
дата

підпис

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ДЕКЛАРАЦІЯ УЧАСНИКА ОСВІТНЬОГО ПРОЦЕСУ

щодо дотримання академічної доброчесності

Цією декларацією я, Галашевський Олександр Юрійович,
студент IV курсу спеціальності 121 – Інженерія програмного забезпечення,
група ІПЗ-21-1
здобувач вищої освіти (шифр та назва спец-ті, курс, академічна група)

підтверджую, що ознайомився з Положенням про систему забезпечення академічної доброчесності у Хмельницькому національному університеті та Кодексом академічної доброчесності і **зобов'язуюсь** дотримуватися їх вимог під час освітнього процесу, проведення наукової діяльності, виконання організаційно-адміністративних функцій тощо.

Усвідомлюю, що у разі порушення мною принципів академічної доброчесності нестиму відповідальність перед академічною спільнотою ХНУ згідно з нормами, визначеними Положенням про систему забезпечення академічної доброчесності у Хмельницькому національному університеті, законодавства України.

2 0.1 2025 р.

Підпис

Завідувачу кафедри
інженерії програмного забезпечення
проф. Бедратюку Л. П.
студента групи ІПЗ-21-1
Галашевський О. Ю.
Прізвище, ініціали

ЗАЯВА

Прошу закріпити за мною тему кваліфікаційної роботи освітнього ступеня «бакалавр» за спеціальністю 121 «Інженерія програмного забезпечення»: Програмна система відстеження гуманітарних вантажів у логістичних ланцюгах постачання

(керівник роботи – Бедратюк Ганна Іванівна)
Прізвище, ім'я, по батькові

2.01.2025
Дата

Підпис студента

Протокол аналізу звіту подібності науковим керівником

Заявляю, що я ознайомився (-лась) з Повним звітом подібності, який був згенерований Системою виявлення і запобігання плагіату щодо роботи:

Автор: Галашевський Олександр Співавтор:

Назва: БКР_Програмна система відстеження гуманітарних вантажів у логістичних ланцюгах постачання Галашевськии Науковий керівник:

Підрозділ: Кафедра інженерії програмного забезпечення

Коефіцієнт подібності 1:2.6%
Коефіцієнт подібності 2:0%

Мікропробіли: 0
Заміна букв: 0
Інтервали: 0
Білі знаки: 0

Дата створення звіту: 2025-06-01 14:21:13.0

Після аналізу Звіту подібності констатую наступне:

☐ Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається.

☐ Запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Таким чином робота повертається на доопрацювання.

☐ Виявлено запозичення і плагіат або навмисні текстові спотворення (маніпуляції), як передбачувані спроби укриття плагіату, які роблять роботу невідповідною вимогам законодавства (Ст. 32. ЗУ Про вищу освіту, пункт 3.1, Ст. 42. ЗУ Про освіту) та вимог НАЗЯВО (Критерій 5), а також кодексу етики і процедурам. Таким чином робота не приймається.

Обгрунтування:

Дата 01.06.2025

експерт 

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ
освітнього ступеня «Бакалавр»

Дипломник Галашевський Олександр Юрійович
Тема Програмна система відстеження гуманітарних вантажів у логістичних ланцюгах постачання
Спеціальність 121 – Інженерія програмного забезпечення

Обсяг кваліфікаційної роботи:

Кількість листів креслень _____; кількість сторінок записки _____
1.Короткий зміст пояснювальної записки та прийнятих рішень У вступі кваліфікаційної роботи обґрунтовано актуальність проблеми ефективної логістики гуманітарної допомоги в умовах сучасних криз, зокрема в контексті повномасштабного вторгнення в Україні. Вказано на існуючі проблеми традиційних ланцюгів постачання, такі як недостатня прозорість, корупційні ризики, неефективне управління ресурсами та складність координації між учасниками. Метою роботи є розробка програмної системи “Lanka” на основі блокчейн-технологій, яка забезпечить прозоре відстеження гуманітарних вантажів, підвищить ефективність операцій і зміцнить довіру між донорами та отримувачами. Для досягнення мети поставлено завдання: аналіз проблем гуманітарної логістики, дослідження сучасних технологій із акцентом на блокчейн, розробка гібридної архітектури з використанням Ethereum та IPFS, створення бази даних і смарт- контрактів, розробка користувацького інтерфейсу, реалізація ключових модулів та проведення тестування. Об’єктом дослідження є процес управління гуманітарними вантажами, предметом – застосування блокчейн-технологій для підвищення прозорості та надійності. Наукова новизна полягає у створенні гібридної архітектури та унікального набору смарт-контрактів, практична значущість – у створенні функціонального прототипу системи, придатного для впровадження гуманітарними організаціями.

2. Висновок про відповідність роботи поставленому завданню Завдання, сформульовані у вступі, включають глибокий аналіз проблем гуманітарної логістики, дослідження сучасних технологій із акцентом на блокчейн, формулювання вимог до системи, розробку гібридної архітектури, структури бази даних і смарт-контрактів, створення користувацького інтерфейсу, реалізацію ключових модулів та проведення тестування. У висновках (наданих у вигляді аналізу розділів) відображено, що проведено аналіз проблем і технологій, сформульовано цілі та завдання, розроблено гібридну архітектуру з використанням PostgreSQL, Ethereum та IPFS, реалізовано базу даних із застосуванням Prisma ORM, створено модульну структуру системи, а також проведено тестування бази даних і бекенду. Проте відсутній детальний опис реалізації користувацького інтерфейсу, ігрових механік, тестування геймплею та візуалізації результатів тестування. Таким чином, отримані результати в основному відповідають поставленим завданням, однак частково не охоплюють усі аспекти, зокрема повноцінну реалізацію UI та ігрових компонентів.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки та передових методів роботи Розділ 1 систематизує та досліджує

проблеми гуманітарної логістики, охоплюючи широкий спектр питань, включаючи прозорість ланцюгів постачання, корупційні ризики, неефективність управління ресурсами та координацію учасників. Автор інтегрує різні аспекти предметної області, демонструючи методологічно обґрунтований і репрезентативний аналіз. Водночас, відсутність конкретних прикладів і статистичних даних дещо знижує глибину дослідження, а формулювання вимог до системи потребує уточнення для уникнення неоднозначності.
Розділ 2 проєктує гібридну архітектуру, що поєднує централізовану реляційну базу даних PostgreSQL із блокчейном Ethereum та децентралізованим сховищем IPFS. Описано модульну структуру системи з чітким визначенням функціональних блоків, що свідчить про архітектурно виважений, модульний та технічно досконалий підхід. Проте відсутність розгляду альтернативних архітектурних шаблонів, детальних схем рівневого розділення логіки, а також відсутність опису цільових платформ і вибору рушія обмежують повноту проєктування.
Розділ 3 реалізує базу даних із застосуванням PostgreSQL та Prisma ORM, що відповідає обраній архітектурі, та налаштування середовища розробки з Docker і Vercel Postgres. Описано структуру моделей даних і конфігурації, що демонструє функціонально повний та програмно реалізований підхід до бекенду.

4. Позитивні сторони роботи Робота демонструє глибоке розуміння проблем гуманітарної логістики та сучасних технологій, зокрема блокчейн, що свідчить про високий рівень аналітичного мислення. Гібридна архітектура системи “Lanka” є інноваційною, поєднуючи переваги централізованих баз даних і децентралізованих технологій, що забезпечує надійність, прозорість і ефективність. Використання сучасних технологій, таких як Ethereum, IPFS, PostgreSQL, Prisma ORM, Docker і Vercel, свідчить про технологічно передовий та інженерно обґрунтований підхід. Модульна структура системи та чітке визначення функціональних блоків демонструють продумане проектування. Проведено всебічне тестування бази даних і бекенду, що підтверджує працездатність ключових компонентів.

5. Негативні сторони роботи Робота має недостатньо деталізований аналіз проблем гуманітарної логістики через відсутність конкретних прикладів і статистичних даних, що знижує переконливість аргументації. Формулювання вимог до системи іноді нечітке, що може призвести до неоднозначностей у реалізації. У проєктній частині не розглянуто альтернативні архітектурні шаблони та не наведено візуальних схем, що ускладнює розуміння структури системи.

6. Оцінка графічного оформлення та пояснювальної записки Графічне оформлення виконано відповідно до теми кваліфікаційної роботи та подано у вигляді діаграм і рисунків. Пояснювальна записка оформлена згідно вимог чинних стандартів.

7. Відгук про кваліфікаційну роботу в цілому Тема роботи є надзвичайно актуальною та має високу практичну значущість у контексті гуманітарних криз, особливо в Україні. Зміст роботи відповідає темі та поставленим завданням, демонструючи системний підхід до розробки інноваційної програмної системи з використанням сучасних технологій блокчейн і децентралізованого зберігання даних. Наукова новизна полягає у створенні гібридної архітектури та унікального набору смарт-контрактів, що є вагомим внеском у сферу гуманітарної логістики. Проте реалізація програмного продукту є частково неповною через відсутність опису користувацького інтерфейсу, ігрових механік та комплексного тестування, що обмежує оцінку зручності інтерфейсу та повноти функціоналу. Застосовані технології є сучасними та релевантними, проте

відсутність деталізації середовища розробки та системних вимог ускладнює оцінку готовності продукту до впровадження.

8. Інші зауваження _____

9. Оцінка кваліфікаційної роботи Кваліфікаційна робота виконана з дотриманням основних методичних вимог, містить ґрунтовний аналіз предметної області та сучасних технологій, а також розробку інноваційної гібридної архітектури системи «Lanka». Програмна реалізація охоплює ключові компоненти бекенду, що відповідає поставленим завданням. Однак відсутність повного опису користувацького інтерфейсу, ігрових механік, комплексного тестування та візуалізації результатів знижує рівень завершеності роботи. Здобувач демонструє достатній рівень знань і вміння застосовувати сучасні технології, проте потребує доопрацювання окремих аспектів реалізації та документації. Враховуючи вищезазначене, робота заслуговує оцінки «добре».

РЕЦЕНЗЕНТ Павлова Ольга Олександрівна, д.ф. доцент, зав. каф. КІІС

"02" 06 2025 р. _____
(підпис)

Anti-Plagiarism (UA) v-15.281 Educational

The maximum coincidence with one document 1.0%

Dictionaries check: en_US, ru_RU, ua_UA. **Errors in the documents: 17%**

	Document		Sum coincidence on the DB	
ID: 242707 Title: БКР_Програмна система відстеження гуманітарних вантажів у логістичних ланцюгах постачання Added in a DB: 2025-06-02 Authors: Галашевський Олександр Heads: Г.І. Бедратюк Consultants: Opponents:	Symbols	Lexemes	Symbols	Lexemes
	99820	1367	2815 (3%)	39 (3%)

Plagiarism sources

ID	Description	Plagiarism presence in the document	
		Symbols	Lexemes