МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ НАУК ТА ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРИЗОВАНИХ СИСТЕМ УПРАВЛІННЯ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри
_____ Олександр ЛИТВИНЕНКО
"_____"_____ 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

(ПОЯСНЮВАЛЬНА ЗАПИСКА)
ЗДОБУВАЧА ВИЩОЇ ОСВІТИ
ОСВІТНЬОГО СТУПЕНЯ «МАГІСТР»

Тема: Програмна система електронного документообігу з використанням технології блокчейн

Виконавець: Антон БУГАЙ

Керівник: Наталія АПЕНЬКО

Нормоконтролер: Євгеній ТУПОТА

Київ 2023

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних наук та технологій

Кафедра комп'ютеризованих систем управління

Спеціальність 123 «Комп'ютерна інженерія»

Освітньо-професійна програма «Системне програмування»

Форма навчання денна

ЗАТВЕРДЖУЮ

Завідувач кафедри

Олександр ЛИТВИНЕНКО

«_____» _____ 2023 р.

ЗАВДАННЯ

на виконання кваліфікаційної роботи

Бугая Антона Миколайовича

1. Тема кваліфікаційної роботи Програмна система електронного документообігу з використанням технології блокчейн

затверджена наказом ректора від «28» 08 2023 р. № 1494/ст

2. Термін виконання роботи (проєкту): з 02.10.2023 р. по 31.12.2023 р.

3. Вихідні дані до роботи (проєкту): середовище розробки *PhpStrom*, редактор діаграм *Microsoft Visio*, текстовий процесор *Microsoft Word*.

4. Зміст пояснювальної записки: вступ, аналіз предметної області, аналіз технологій розробки, програмна реалізація системи, огляд системи електронного документообігу, висновки.

5. Перелік обов'язкового графічного (ілюстративного) матеріалу:

1) Клієнт-серверна архітектура

2) Діаграма класів контролерів

3) Діаграма класів моделей

4) Діаграма зв’язків між таблицями

5) Процес реєстрації користувача (схема алгоритму)

6) Процес створення документа (схема алгоритму)

6. Календарний план

№ п/п	Етапи виконання кваліфікаційної роботи	Термін виконання етапів	Примітка
1	Пошук та аналіз джерел для аналізу предметної області за темою кваліфікаційної роботи	02.10-21.10	
2	Розробка плану кваліфікаційної роботи	22.10-02.10	
3	Розробка розділу 1: Аналіз предметної області	03.11-13.11	
4	Розробка розділу 2: Аналіз технологій розробки системи	14.11-24.11	
5	Розробка розділу 3: Програмна реалізація системи	25.11-02.12	
6	Розробка розділу 4: Огляд системи електронного документообігу	03.12-05.12	
7	Оформлення пояснювальної записки та написання висновків	06.12–18.12	
8	Розробка презентації для захисту роботи	19.12–24.12	
9	Підготовка до захисту	25.12–31.12	

7. Дата отримання завдання: «02» 10. 2023 р.

Керівник кваліфікаційної роботи _____ Наталія АПЕНЬКО

Завдання прийняв до виконання _____ Антон БУГАЙ

РЕФЕРАТ

Кваліфікаційна робота на тему «Програмна система електронного документообігу з використанням технології блокчейн». Записка до кваліфікаційної роботи містить: 91 с., 32 рис., 6 табл., 26 літературних джерел, 1 додаток.

Ключові слова: БЛОКЧЕЙН, СИСТЕМА ЕЛЕКТРОННОГО ДОКУМЕНТООБІГУ, СЕД, ДОКУМЕНТ, СМАРТ-КОНТРАКТ.

Об’єктом дослідження є електронний документообіг з використанням технології блокчейн.

Предметом дослідження є програмна система електронного документообігу з використанням технології блокчейн.

Метою кваліфікаційної роботи є розробка програмної системи електронного документообігу, яка використовує блокчейн для підвищення безпеки.

Технічними та програмними засобами що використовувалися при розробці кваліфікаційної роботи є персональний комп’ютер з операційною системою *Windows* 11 та *Linux Ubuntu* 20.04 *lts*.

Для створення програмного модулю використовувався об’єктно орієнтований метод програмування, алгоритми побудовані відповідно міжнародним і державним стандартам. Основними характеристиками та показниками в роботі є ефективність, простота та безпека розробленого програмного забезпечення.

В свою чергу науковою новизною – шифрування документів за допомогою алгоритмів хешування, перевірка їх на дійсність та доступ до зашифрованих даних тільки авторизованим користувачам.

Значущість виконаної роботи полягає у детальному аналізі програмних засобів електронного документообігу, розробці власного зручного для користувача додатку

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

СЕД – система електронного документообігу, комплекс програмних та апаратних засобів, спрямований на автоматизацію процесів створення, обігу, обробки та зберігання електронних документів.

Блокчейн – це розподілена база даних, яка записує транзакції у вигляді блоків, з'єднаних за допомогою криптографії.

IoT – мережа фізичних об'єктів, які мають вбудовані технології, що дозволяють здійснювати взаємодію з зовнішнім середовищем.

SHA (*Secure Hash Algorithm*) – набір криптографічних хеш-функцій, що включає алгоритми *SHA-0* і *SHA-1*, та групи *SHA-2* і *SHA-3*. *SHA-256* є входить в групу *SHA-2* разом із *SHA-512* та іншими.

PoW (*Proof-Of-Work*) – принцип консенсусу в блокчейн-технології, який використовується для вирішення проблеми подвійного витрачання і забезпечення безпеки мережі. У системах, які використовують *PoW*, новий блок додається до ланцюга блоків шляхом вирішення обчислювально складної задачі, яка вимагає значних обчислювальних ресурсів.

PoS (*Proof-Of-Stake*) – принцип консенсусу в блокчейн-технології, який протистоїть *Proof-of-Work* (*PoW*). У системах, що використовують *PoS*, новий блок додається до ланцюга блоків шляхом вибору того, хто стає лідером, на основі кількості криптовалюти, якою вони володіють або заморозили у системі.

DeFi – мережа фінансових додатків, побудована на основі блокчейну.

NFT (*Non-Fungible Token*) – цифрові активи, що використовують технологію блокчейн для унікальної ідентифікації та підтвердження справжності цифрових об'єктів, таких як зображення, відео, музика та інші файли.

Перший та другий першовзор – поняття відноситься до криптографії, вказує на можливість знаходження відображення (входу) до певного значення хеш-функції (виходу).

Нікнейм – ім'я користувача в системі.

ВСТУП

Зважаючи на швидкий розвиток технологій і високі вимоги до ефективності та безпеки документообігу, сучасні організації ставлять перед собою завдання вдосконалення своїх систем управління документообігом. Електронний документообіг стає важливим елементом сучасного бізнесу, а використання технології блокчейн значно підвищує безпеку та надійність цього процесу.

Сучасні умови вимагають нових підходів до забезпечення цілісності, ефективності та безпеки. У цьому контексті використання технології блокчейн спрямоване на забезпечення високого рівня децентралізації, незмінності та конфіденційності, тим самим сприяючи підвищенню безпеки та надійності віртуального обміну інформацією.

Спостереження за застосуванням технології блокчейн у різних галузях показує, що ця технологія стає невід'ємною частиною вдосконалення бізнес-процесів та підвищення ефективності. У сфері електронного документообігу, де важливо забезпечити надійність і незмінність даних, використання блокчейна стало перспективним напрямком розвитку. Мета дослідження полягає в тому, щоб визначити, як цю технологію можна успішно інтегрувати в електронний документообіг для досягнення максимальної ефективності та безпеки.

Зараз у світі активно розвиваються інтелектуальні технології та цифрові інновації. Прискорений перехід на електронні формати в усіх сферах діяльності підкреслює важливість створення безпечних та ефективних механізмів документообігу. Розробка на основі технології блокчейн матиме практичну користь для компаній, організацій та установ, які хочуть підвищити рівень інформаційної безпеки та оптимізувати внутрішні бізнес-процеси.

Однією з ключових переваг блокчейн-технології є використання криптографії для забезпечення безпеки даних. У контексті електронного документообігу, це означає, що інформація має високий рівень захисту завдяки сучасним методам

шифрування та підпису. Кожен блок інформації з'єднаний з попереднім хешем, створюючи ланцюг, який ускладнює будь-яку спробу зміни чи вилучення даних. Це робить систему стійкою до хакерських атак та забезпечує високий рівень імутабельності.

Технологія блокчейн також пропонує децентралізований підхід до управління документами. Відсутність центрального органу контролю виключає можливість однієї точки невдачі та робить систему більш стійкою до відмов. Крім того, впровадження смарт-контрактів дозволяє автоматизувати багато процесів, що призводить до підвищення ефективності, зменшення часових затрат та уникнення помилок у виконанні угод.

Блокчейн також може сприяти підвищенню конфіденційності та автентифікації документів. За допомогою ключів шифрування та ідентифікації, він може контролювати доступ до інформації, забезпечуючи тільки визначеним користувачам право перегляду чи зміни даних. Це робить систему електронного документообігу більш безпечною і відповідною стандартам конфіденційності та захисту особистої інформації.

Актуальність даної теми обумовлена рядом факторів, які свідчать про необхідність вирішення проблем, пов'язаних із управлінням електронними документами в організаціях. Аналізуючи і порівнюючи існуючі розв'язання проблеми, можна визначити ключові аспекти актуальності теми:

- швидкозростаючий обсяг електронної інформації;
- підвищення вимог до безпеки та конфіденційності;
- потреба у раціоналізації бізнес-процесів;
- виклики сучасності у зв'язку з глобалізацією – глобальна природа сучасного бізнесу вимагає швидкого та ефективного обміну документами між різними регіонами та учасниками бізнес-процесів, що висуває нові вимоги до систем електронного документообігу.

Розв'язання цих проблем через розробку та впровадження програмної системи електронного документообігу, базованої на технології блокчейн, є доцільним та актуальним завданням. Висока ефективність такої системи може значно поліпшити

управління документами в організаціях, сприяючи підвищенню продуктивності та забезпеченню безпеки і конфіденційності інформації.

Об’єктом кваліфікаційної роботи є електронний документообіг, зокрема процеси процес управління та обміну документами у великих інформаційних системах. Електронний документообіг є динамічною системою, що включає в себе неперервні процеси створення та обробки документів, а також їх обмін між різними структурними підрозділами організації. Об’єкт має інтерфейс взаємодії з різними учасниками – від співробітників організації до зовнішніх клієнтів. Важливо врахувати потреби різних категорій користувачів при проектуванні інтерфейсу.

Однією з ключових характеристик є необхідність забезпечення безпеки електронних документів, їх конфіденційності та цілісності. Це стає ще більш актуальним у сучасному цифровому середовищі. Безпека системи підвищується за рахунок особливостей технології блокчейн та використання криптографії у системі.

Об’єкт відображає потребу сучасного бізнесу в ефективному, прозорому та безпечному обміні електронними документами, що враховує високий темп робочого процесу та необхідність миттєвого доступу до інформації.

Предметом дослідження є програмна система електронного документообігу, що базується на технології блокчейн. Ця система представляє собою комплексну інформаційну структуру, яка охоплює весь цикл життя електронного документа, від його створення до архівування. При цьому, вона використовує принципи та переваги, які надає технологія блокчейн. Характеристики предмету:

1) життєвий цикл документа:
   - створення – системі передбачається процес створення електронного документа, включаючи в себе визначення його типу, вмісту та учасників процесу;
   - Обробка – враховуючи особливості технології блокчейн, документ може бути оброблений різними учасниками системи, зберігаючи при цьому відомості про кожну транзакцію.
2) система управління документами:

– архівація – система передбачає процес архівації документів, щоб забезпечити їхню довгострокову збереженість та доступність;
– пошук та витяг – управління базою даних документів, що включає в себе ефективний пошук, аналітику та можливість витягу необхідної інформації.

3) використання технології блокчейн:
   – децентралізація – забезпечення безпеки і надійності шляхом використання розподіленої системи збереження даних;
   – смарт-контракти – використання смарт-контрактів для автоматизації деяких процесів, що спрощує управління документами.
4) інтерфейс користувача:
   – інтуїтивний інтерфейс – подання системи у вигляді зручного та інтуїтивно зрозумілого інтерфейсу, який дозволяє користувачам легко взаємодіяти з системою;
   – персоналізація – налаштовані опції та персоналізований доступ для різних категорій користувачів.

Мета кваліфікаційної роботи – розробити та впровадити програмну систему електронного документообігу на основі технології блокчейн, яка буде ефективним інструментом обробки документів. Дослідити взаємодію блокчейну з додатками та можливості його використання для підвищення ступеня захисту електронного документообігу в системі. Система спрямована на полегшення та пришвидшення обробки документації на підприємстві та покращення рівня безпеки документообігу шляхом використання блокчейну для зберігання деяких даних для автентифікації користувачів та перевірки документів на дійсність. Для досягнення цієї мети необхідно вирішити наступні завдання:

1) провести аналіз предметної області;
2) провести аналіз існуючих систем документообігу;
3) ознайомитися з технічними аспектами технології блокчейн, вивчити основні принципи, алгоритми та можливості, які вона надає для забезпечення безпеки та надійності;

4) розробити концептуальну та технічну архітектуру програми, визначити основні компоненти, взаємодію між ними та вибрати оптимальні технології;
5) розробити програму на основі розробленої архітектури, використовуючи технології блокчейн для забезпечення безпеки;
6) визначити вимог до середовища виконання системи;
7) описати роботу користувачів та адміністраторів із розробленою системою електронного документообігу.

Ці завдання спрямовані на створення інноваційного та ефективного рішення для управління електронними документами, що враховує сучасні технології та вимоги користувачів.

Кваліфікаційна робота складається з таких розділів:

1) аналіз предметної області:
   - аналіз систем електронного документообігу;
   - аналіз понять предметної області;
   - дослідження основних аспектів та особливостей технології блокчейн.
2) технології розробки системи:
   - ознайомлення з архітектурою веб-систем;
   - дослідження основних частин архітектури систем;
   - ознайомлення з базами даних.
3) програмна реалізація системи:
   - визначення програмних та апаратних вимог до середовища виконання;
   - огляд структури системи;
   - огляд баз даних.
4) огляд системи електронного документообігу:
   - огляд роботи користувача з системою;
   - адміністрування системи;
   - блокчейн в системі електронного документообігу.

Кожен розділ закінчується висновком, який узагальнює ключові поняття та пропонує напрямки для подальших досліджень.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Аналіз систем електронного документообігу

1.1.1. Огляд предметної області систем електронного документообігу

Система електронного документообігу (СЕД) – це комплекс програмних та апаратних засобів, спрямований на автоматизацію процесів створення, обігу, обробки та зберігання електронних документів в організації чи підприємстві. Для полегшення обробки великої кількості документації такі системи стають все популярнішими серед підприємств різних рівнів та областей діяльності.

Залежно від цілей і завдань розрізняють внутрішній та зовнішній документообіги. Електронний документообіг всередині організації дозволяє здійснювати обмін документами тільки всередині структурних підрозділів підприємства. Зовнішній документообіг – це обмін документами між партнерами (замовниками, регуляторами).

Перевагами використання систем електронного документообігу є прозорість всіх етапів діяльності компанії адже кожен документ адже усі документи, завдання чи процеси реєстрації в системі електронного документообігу супроводжуються обліково-реєстраційною інформацією, що допомагає спростити контроль термінів виконання задач, їх відстеження та, аналіз та планування.

Відкритість кожного процесу в системі передбачає підвищення відповідальності працівників. Завдяки системі повідомлень працівники не зможуть забути про жодне завдання. Розрахунок робочого часу і трудовитрат дозволяє керівництву ефективніше планувати розподіл завдань між співробітниками. Звідси випливає підвищення якості обслуговування клієнтів підприємства.

У загальному вигляді СЕД можна розглядати як організаційно-технічну систему, яка забезпечує процес створення, контролю доступу та розповсюдження

електронних документів у комп'ютерних мережах, і водночас забезпечує можливість керування документообігом в організації. Вагомою перевагою СЕД є здатність точно та якісно виконувати багато завдань документообігу та обробляти великі обсяги документів.

Типи файлів, які зазвичай підтримуються, включають: текстові документи, зображення, електронні таблиці, аудіо- та відеодані та веб-документи. Основною метою СЕД є організація зберігання електронних документів та роботи з ними (включаючи пошук за атрибутами та за вмістом). Зміни документів, терміни виконання та всі їх версії повинні автоматично відстежуватися в системі.

Комплексна система електронного документообігу повинна охоплювати весь цикл управління бізнесом від встановлення завдань створення документів до занесення документів в архів, забезпечуючи зберігання документів у будь-якому місці будь-якого формату, включаючи складні компіляції.

СЕД має об'єднати розрізнені документообіги територіально віддалених компаній в єдину систему та забезпечити гнучке управління документами шляхом чіткого визначення маршрутів руху. Необхідно чітко розмежувати доступ користувачів до різних документів відповідно до закріплених за ними повноважень, функцій і повноважень. Крім того, СЕД має бути адаптований до існуючої організаційної структури компанії, штатного розкладу та систем діловодства, а також інтегрований з існуючими системами компанії.

Основними користувачами систем електронного документообігу є великі держ установи, промислові компанії, банки, авіаперевізники та всі інші організації, діяльність яких супроводжується створенням, обробкою та зберіганням великих обсягів документів.

Відповідно до основних принципів електронного документообігу він повинен функціонувати за такою ознакою:

- одноразове оформлення документів;
- можливість паралельного виконання різних операцій з метою скорочення часу руху документів та підвищення швидкості їх виконання;
- безперервність руху документів;

- єдина інформаційна база документів для їх зберігання та усунення можливості дублювання;
- ефективна система пошуку документів;
- система звітності за статусами і атрибутами документів.

Залежно від сфери застосування виділяються внутрішній електронний документообіг (включаючи процеси всередині компанії) та зовнішній (забезпечення обміну електронними документами між партнерами, компаніями та держ організаціями тощо).

В межах внутрішнього електронного документообігу можна виділити такі:

- управлінський електронний документообіг (включає в себе накази, інструкції та ін.);
- кадровий документообіг (кадрові накази, трудові договори тощо);
- бухгалтерський електронний документообіг (акти, накладні);
- складський документообіг (квитанції, свідоцтва).

Якщо в компанії існують специфічні бізнес-процеси, можуть виникати й інші види обміну електронними документами. Напрямок і масштаби використання електронного документообігу залежать від цілей кожної конкретної компанії.

1.1.2. Основні процеси електронного документообігу

Електронний документообіг включає кілька основних процесів, які охоплюють життєвий цикл електронних документів від їх створення до зберігання.

Створення електронних документів є першим і важливим кроком у процесі електронного документообігу. На цьому етапі визначається формат, структура і зміст документа.

Електронні документи створюються різними способами, для цього використовуються текстові редактори, такі як *Microsoft Word* або *Google Docs*. Існують також спеціальні інструменти для створення електронних таблиць, презентацій та інших типів документів. Онлайн-редактори також дозволяють користувачам спільно редагувати документи в реальному часі, незалежно від їх

місцезнаходження. Деякі організації використовують автоматичні генератори документів, які дозволяють автоматично створювати документи на основі попередньо визначених шаблонів і даних. Це особливо ефективно для стандартних структурованих документів, таких як контракти, звіти або рахунки.

Створені електронні документи зазвичай зберігаються в електронному сховищі, такому як хмарний сервіс або корпоративний сервер. Це дозволяє отримувати доступ до документів з різних пристроїв і місць, що важливо для комфорту команди.

Окрім створення та редагування документів електронний підпис та авторизація електронного документообігу є ключовим для забезпечення безпеки, визначення автентичності та моніторингу участі різних сторін у процесі обробки документу.

Одним із найважливіших аспектів є можливість користувача створювати цифровий підпис для підтвердження своєї участі у створенні чи редагуванні документа. Для цього можуть використовуватися асиметричні криптографічні методи, а також децентралізована система блокчейн для забезпечення вищого рівня безпеки.

В окремих системах є можливість створення віртуального відпису, що дає можливість накладати підпис за допомогою миші чи стилусу.

Процес розсилки документів передбачає використання електронної пошти та систем обміну повідомленнями для ефективного розповсюдження користувачам і групам користувачів. Ключовим аспектом є маршрутизація, де визначаються правила й алгоритми для автоматичного визначення оптимального шляху потоку документів.

Система маршрутизації автоматично призначає осіб, відповідальних за подальшу обробку документів, на основі різних критеріїв, таких як тип документа, вміст та інші параметри. Додатковим аспектом є можливість переглядати, коментувати та коментувати документи, коли вони поширюються.

1.2. Аналіз понять предметної області

1.2.1. Поняття блокчейну

Технологія блокчейн – це система запису та передачі інформації, що дозволяє зберігати дані у вигляді ланцюжка блоків. Кожен блок містить інформацію про певну кількість транзакцій та хеш попереднього блоку. Таким чином, кожен блок забезпечує взаємозв’язок з попереднім блоком, що утворює ланцюжок. По своїй суті блокчейн являє собою розподілену базу даних, яка має вигляд зв’язного списку. Отже тут можна зберігати майже будь-яку інформацію, яка потребує високого ступеня захисту.

Основні переваги блокчейн-технології включають децентралізацію, прозорість, надійність та ефективність. Завдяки децентралізованості, дані зберігаються на різних комп’ютерах, через що мережу майже не можливо зламати. Прозорість дозволяє користувачам перевіряти та слідкувати за всіма транзакціями в режимі реального часу. Надійність забезпечується тим, що кожен блок містить унікальний хеш-код, який унеможливлює його модифікацію без відповідного впливу на всю мережу.

Крім того, блокчейн-технологія дозволяє зменшити витрати на проведення транзакцій, оскільки усі операції можуть бути здійснені без посередницьких послуг. Наприклад, у банківській системі на оплату послуги банку йде певна частина коштів, яку можна значно зменшити застосовуючи блокчейн. Але вона не є безкоштовною технологією. Кожна транзакція коштує певну кількість криптовалюти. Зазвичай ця сума не велика, але виходячи з цього слід ретельно планувати, які саме дані потрібно записувати в блокчейн аби мінімізувати витрати на його використання.

Одним з основних принципів технології блокчейн є консенсус. Це означає, що кожен користувач у мережі повинен погоджуватися з правильністю транзакцій. Якщо один користувач намагається здійснити фальшиву транзакцію, то інші користувачі відхиляють цю транзакцію і не дозволяють їй пройти. Таким чином, мережа залишається безпечною і надійною.

1.2.2. Безпека блокчейну

Блокчейни захищені за допомогою різних механізмів, серед яких передові криптографічні методи та математичні моделі поведінки і прийняття рішень. Блокчейн-технологія є базовою структурою більшості криптовалютних систем та запобігає дублюванню або знищенню таких цифрових активів. Безпека блокчейну досить складна тема, тому важливо розуміти основні концепції та механізми, що забезпечують надійний захист цих інноваційних систем.

Хоча багато функцій відіграють роль у безпеці блокчейну, є дві найважливіші концепції консенсусу та незмінності. Консенсус відноситься до здатності нод узгоджувати справжній стан мережі та дійсність транзакцій у розподіленій блокчейн-мережі. Як правило, процес досягнення консенсусу залежить від так званих алгоритмів консенсусу.

З іншого боку, незмінність відноситься до здатності блокчейнів запобігати зміні вже підтверджених транзакцій. Хоча ці транзакції часто пов’язані з переказом криптовалют, вони можуть також стосуватися запису інших негрошових форм цифрових даних.

У сукупності консенсус та незмінність забезпечують основу для безпеки даних у блокчейн-мережах. У той час як алгоритми консенсусу забезпечують дотримання правил системи та згоду всіх залучених сторін із поточним станом мережі, незмінність гарантує цілісність даних та записів транзакцій після підтвердження достовірності кожного нового блоку даних.

Блокчейни значною мірою покладаються на криптографію для забезпечення безпеки своїх даних. У цьому контексті фундаментальне значення мають криптографічні хеш-функції. Хешування – це процес, при якому алгоритм (хеш-функція) отримує вхідні дані будь-якого розміру і повертає результат (хеш), що містить передбачуваний та фіксований розмір (або довжину). Немає значення, якої довжини буде вхід, вихід завжди матиме однакову довжину. Але якщо вхід

зміниться, вихід буде зовсім іншим. Однак, якщо вхід не змінюється, результат хешу завжди буде однаковим незалежно від того, скільки разів ви запускаєте хеш-функцію.

Алгоритми хешування розроблені як односторонні функції, тобто їх неможливо скасувати без великої кількості часу та ресурсів. Іншими словами, досить легко створити хеш з вхідних даних, але відносно важко зробити навпаки (знайти вхідні дані з вихідних). Загалом, чим складніше знайти вхідні дані, тим безпечнішим вважається алгоритм хешування.

У блокчейнах ці вихідні значення, відомі як хеші, використовуються як унікальні ідентифікатори для блоків даних. Хеш кожного блоку генерується із прив'язкою до хешу попереднього блоку, і саме це створює ланцюг зв'язаних блоків. Хеш блоку залежить від даних, що містяться в цьому блоці, а це означає, що будь-яка зміна, внесена в дані, потребує зміни хешу блоку.

Тому хеш кожного блоку генерується на основі даних, що містяться в цьому блоці і хешу попереднього блоку. Ці хеш-ідентифікатори відіграють важливу роль у забезпеченні безпеки та незмінності блокчейну.

Хешування також використовується в алгоритмах консенсусу, які в свою чергу використовуються для перевірки транзакцій. Наприклад, у блокчейні *Bitcoin* алгоритм *Proof of Work* (*PoW*) використовує хеш-функцію *SHA*-256. Як випливає з назви, *SHA*-256 приймає введені дані та повертає хеш довжиною 256 біт чи 64 символи.

Різні хеш-функції вироблятимуть хеш різної довжини, але можливий розмір виходу кожного хеш-алгоритму завжди сталий, тобто незалежно від довжини вхідних даних вихід завжди матиме однакову кількість символів. Наприклад, *SHA-256* створить 256-бітний вихід, тоді як *SHA-1* завжди створить 160-бітний вихід. Абревіатура *SHA* означає *Secure Hash Algorithm*. Це набір криптографічних хеш-функцій, що включає алгоритми *SHA-0* і *SHA-1*, а групи *SHA-2* і *SHA-3*. *SHA-256* є входить в групу *SHA-2* разом із *SHA-512* та іншими. Зараз тільки *SHA-2* і *SHA-3* вважаються безпечними.

Звичайна хеш-функція має багато застосувань, зокрема пошук у базі даних, аналіз великих файлів і керування даних. Криптографічні хеш-функції широко використовуються в програмах захисту інформації, таких як автентифікація повідомлень і цифрові відбитки пальців. Що стосується *Bitcoin*, криптографічні хеш-функції є невід'ємною частиною процесу майнінгу, а також відіграють певну роль у створенні нових адрес і ключів.

Справжня сила хешування виявляється при роботі з величезними обсягами інформації. Наприклад, ви можете запустити великий файл або набір даних через хеш-функцію, а потім використати її вихід, щоб швидко перевірити точність і цілісність даних. Це можливо через детерміновану природу хеш-функції: вхід завжди призводить до спрощеного та стисненого результату. Цей метод позбавляє від необхідності зберігати та запам'ятовувати великі обсяги даних.

Хешування особливо корисне в контексті технології блокчейн. Блокчейн *Bitcoin* має ряд операцій, пов'язаних із хешуванням, більшість із яких виконується під час майнінгу. Фактично, майже всі протоколи криптовалют покладаються на хеш-функції для зв'язування та об'єднання груп транзакцій у блоки та для створення криптографічних зв'язків між кожним блоком, створюючи тим самим блокчейн.

Злом криптографічної хеш-функції потребує грубої сили та великої кількості апаратних ресурсів. Щоб «розгорнути» криптографічну хеш-функцію, ви повинні вибирати вхідні дані методом проб і помилок, поки не отримаєте правильний результат. Однак також можливо, що різні вхідні дані дадуть однаковий результат, і в цьому випадку відбудеться «зіткнення».

Технічно криптографічна хеш-функція повинна відповідати трьом властивостям, щоб вважатися надійно захищеною:

1) стійкість до колізії – неможливість знаходження двох різних входів, які виробляють однаковий хеш;
2) стійкість до знаходження першого першовзору – відсутність можливості "розвороту" хеш-функції (знаходження входу через заданий вихід);
3) стійкість до знаходження другого першовзору – відсутність можливості знайти будь-який другий вхід, який би мав такий самий хеш, що і перший.

Колізії виникають, коли різні вхідні дані створюють однаковий хеш. Тоді хеш-функція вважається стійкою до клізій, доки хтось не виявить колізію. Колізії завжди існуватимуть для будь-якої хеш-функції через нескінченну кількість входів і кінцеву кількість виходів. Таким чином, хеш-функція є стійкою до колізії, коли ймовірність її виявлення настільки низька, що потрібні мільйони років обчислень. З цієї причини, хоча не існує хеш-функції без колізій, деякі функції настільки сильні, що їх можна вважати стабільними (наприклад, *SHA-256*). Серед різних алгоритмів *SHA* групи *SHA-0* і *SHA-1* більше не вважаються стійкими до колізій. Наразі стійкими вважаються тільки групи *SHA-2* і *SHA-3*.

Стійкість до знаходження першого першовзору. Ця властивість тісно пов'язана з поняттям односторонньої функції. Хеш-функція вважається стійкою до знаходження першого першовзору, якщо існує дуже низька ймовірність того, що хтось зможе знайти вхідні дані, використовуючи згенерований вихід. Ця властивість відрізняється від попередньої властивості, оскільки зловмисник повинен вгадати вхідні дані на основі певних вихідних даних.

Властивість стійкості до знаходження першого першовзору є цінною для захисту даних, оскільки просте хеш повідомлення може підтвердити його автентичність без розкриття додаткової інформації. Насправді багато постачальників послуг зберігають і використовують хеші, згенеровані з паролів, замість використання їх у відкритому виді.

Стійкість до знаходження другого першовзору. Цей тип стабільності знаходиться між двома попередніми властивостями. Атака знаходження другого першовзору складається з пошуку певного вхідного сигналу, який може створити такий вихід, який був згенерований за допомогою іншого раніше відомого входу.

Іншими словами, атака знаходження другого першовзору включає виявлення колізії, але замість пошуку двох випадкових вхідних даних, які виробляють однаковий хеш, атака спрямована на пошук вхідних даних, здатних відтворити хеш, який вже був згенерований іншими вхідними даними.

Таким чином, будь-яка хеш-функція, стійка до колізій, є стійкою до атаки знаходження другого першовзору, тому остання завжди потребує колізії. Тим не

менш, все ще можливо провести атаку з знаходженням першого першовзору у функції, стійкій до колозії, оскільки вона передбачає пошук входу, за відомого виходу.

1.2.3. Централізована та децентралізована системи

Централізовані системи передбачають управління системою одним органом. Вся влада і повноваження знаходяться у однієї центральної точки управління, яка має повний контроль над рішеннями та ресурсами усієї системи. Взаємодія централізованої системи з користувачем проілюстрована на рис. 1.1.

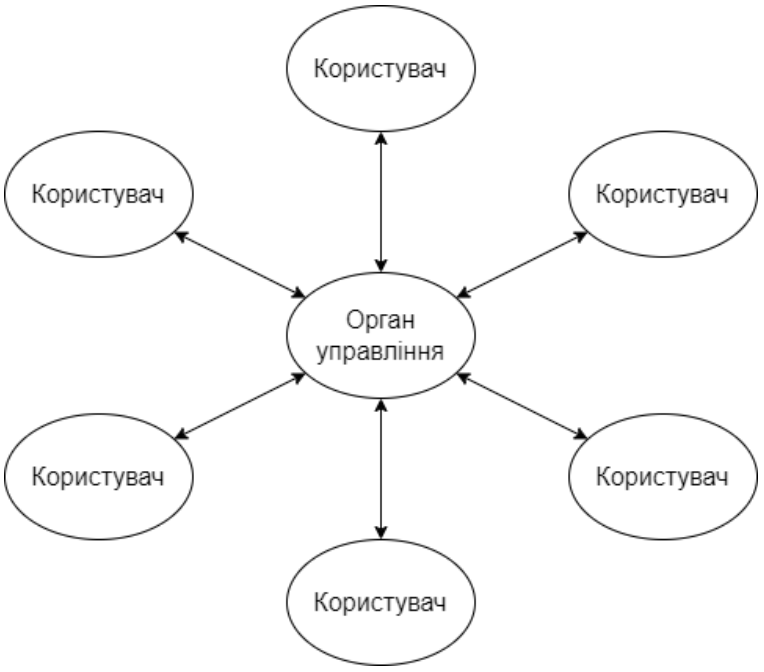

Рис. 1.1. Взаємодія централізованої системи з користувачем

Основними перевагами такої системи є:

- однозначний контроль – одна центральна точка влади приймає ключові рішення поведінки системи;
- ефективність управління – централізовані системи можуть бути ефективними для забезпечення стратегії управління;

- однозначна відповідальність – прийняття та виконання рішень частіше всього лежить в одних руках, це значно полегшує визначення відповідальності.

До недоліків централізованих систем можна віднести:

- залежність від центрального органу є важливим недоліком, адже якщо точка влади зазнає проблем або втрачає здатність приймати рішення то це може призвести до серйозних наслідків у всій системі;
- низька гнучкість – централізовані системи можуть бути менш гнучкими та менш придатними до швидкого реагування на зміни;
- велика віддаленість полягає у тому, що у великих централізованих систем влада і рішення віддалені від кінцевих користувачів або груп;
- централізовані системи зазвичай мають більші витрати на їх обслуговування.

Децентралізовані системи передбачають управління ресурсами розподілене між учасниками цієї системи. Центральної точки системи немає і таких підхід прагне зменшити залежність від управління одним органом та створити більш гнучкі та резистентні до втручань системи. Граф децентралізованої системи зображено на рис. 1.2.

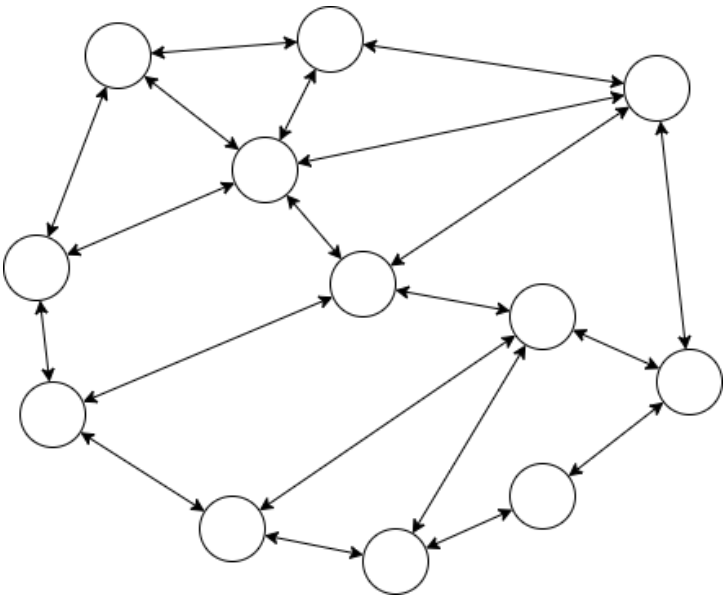

Рис. 1.2. Граф децентралізованої системи

Серед ключових аспектів систем такого способу управління є:

- децентралізована контроль над рішеннями між різними учасниками системи замість зібрання всієї влади в одній точці;
- ресурси, такі як оброблювальна потужність і зберігання даних розподілені між усіма частинами системи;
- безпека та резистентність – системи менш вразливі до виходу з ладу одного з учасників або атак на систем, оскільки немає єдиної точки зберігання даних, а вся інформація розподілена між учасниками;
- децентралізовані системи виявляють ознаки самоорганізації, де учасники можуть реагувати на зміни і приймати рішення локально;
- децентралізовані системи більш гнучкі та краще сприяють інноваціям, адже рішення приймаються локально та швидко адаптуються до змін.

Децентралізовані системи використовуються в різних областях, включаючи фінанси (криптовалюти), децентралізовані мережі (блокчейн), інтернет-ресурси, інтернет речей (*IoT*) тощо.

1.2.4. Огляд існуючих блокчейнів

Ethereum – це децентралізована блокчейн-платформа, яка встановлює однорангову мережу, яка безпечно виконує та перевіряє код програми, що називається смарт-контрактами. Смарт-контракти дозволяють учасникам здійснювати транзакції одну з одною без довіреного центрального органу. Записи транзакцій є незмінними, підлягають перевірці та безпечно розподіляються по мережі, що дає учасникам повну власність і видимість даних транзакцій. Транзакції надсилаються та отримуються обліковими записами *Ethereum*, створеними користувачами. Відправник повинен підписувати транзакції та витрачати *Ether*, криптовалюту *Ethereum*, як вартість обробки транзакцій у мережі.

Ethereum пропонує надзвичайно гнучку платформу для створення децентралізованих програм за допомогою за допомогою зробленої спеціально мови програмування *Solidity* та віртуальної машини (ноди) *Ethereum*. Розробники

децентралізованих додатків, які розгортають смарт-контракти на *Ethereum*, отримують вигоду від багатої екосистеми інструментів для розробників і встановлених передових практик, які прийшли зі зрілістю протоколу. Ця зрілість також поширюється на якість користувацького досвіду для звичайного користувача додатків на *Ethereum*, з такими гаманцями, як *MetaMask*, *Argent*, *Rainbow* та іншими, які пропонують прості інтерфейси для взаємодії з блокчейном *Ethereum* і розгорнутими там смарт-контрактами.

Велика база користувачів *Ethereum* заохочує розробників розгортати свої додатки в мережі, що ще більше зміцнює *Ethereum* як основну платформу для децентралізованих систем, таких як *DeFi* та *NFT*. У майбутньому зворотно сумісний протокол *Ethereum 2.0*, який зараз розробляється, забезпечить більш масштабовану мережу для створення децентралізованих програм, які вимагають більшої пропускної здатності транзакцій. Розглянемо кілька технологій, які працюють на платформі *Ethereum*:

- децентралізовані фінанси *DeFi* – це мережа фінансових додатків, побудована на основі блокчейну. Вона відрізняється від існуючих фінансових мереж тим, що вона відкрита і програмована, працює без центрального органу влади та дозволяє розробникам пропонувати нові моделі для платежів, інвестування, кредитування та торгівлі. Використовуючи розумні контракти та розподілені системи, клієнти можуть легко створювати безпечні децентралізовані фінансові програми. Наприклад, компанії *DeFi* вже пропонують продукти;
- незамінні токени *NFT* розшифровується як "*Non-Fungible Token*", що перекладається як "непереносний токен". *NFT* являє собою цифрові активи, що використовують технологію блокчейн для унікальної ідентифікації та підтвердження справжності цифрових об’єктів, таких як зображення, відео, музика та інші файли. Основна відмінність *NFT* від інших видів криптовалют, таких як біткоїн або ефіріум, полягає в тому, що вони є непереносними та унікальними. Кожен *NFT* має свій унікальний ідентифікатор, що робить його неповторним на відміну від інших токенів.

NFT часто використовуються у культурній та художній сфері, де художники, музиканти та інші творчі особи можуть створювати цифрові твори та продавати їх як унікальні активи. Володіння *NFT* підтверджується записом у блокчейні, що надає їм довіреної справжності та непідробності. Покупці *NFT* отримують цифрову власність та права на володіння конкретним цифровим контентом.

Переваги блокчейну *Ethereum*:

- *Ethereum* дозволяє створювати та виконувати смарт-контракти - програмні коди, які автоматично виконують умови, записані в них, без потреби посередників;
- *Ethereum* надає інфраструктуру для розробки та розгортання децентралізованих програм (*DApps*);
- *Ethereum* активно розвивається, і внесення змін в протокол дозволяє покращувати його функціональність та безпеку;
- безліч проектів та *ICO* (*Initial Coin Offering*) здійснюють свою діяльність на платформі *Ethereum*, що сприяє поширенню та прийняттю цієї технології.

Недоліки блокчейну *Ethereum*:

- *Ethereum* стикається з проблемами масштабованості, що призводить до високих комісій та затримок в мережі в періоди підвищеної активності;
- для звичайних користувачів *Ethereum* може здаватися складним і важким для розуміння, особливо в порівнянні з більш звичними централізованими сервісами;
- для написання смарт-контрактів використовується мова програмування *Solidity*, яку створюють спеціально для такої роботи. Ця сова програмування досить молода відносно інших, більш поширених, в ній присутня нестача класів, зручних для початківців, що є значущою проблемою.

Логотип блокчейну *Ethereum* зображено на рис. 1.3.

Рис. 1.3. Логотип *Ethereum*

Cardano – це блокчейн-платформа, розроблена з метою створення стійкого та масштабованого середовища для виконання смарт-контрактів та розробки децентралізованих додатків (*DApps*). Вона була заснована Чарльзом Хоскінсоном, одним із співзасновників *Ethereum*, та його командою.

Блокчейн *Cardano* написаний мовою програмування *Haskell*, яка має високий рівень відмовостійкості. Застосований алгоритм *Proof-of-Stake Ouroboros* передбачає участь у голосуванні всіх учасників, він схожий на новий алгоритм *Ethereum Casper*, але, за словами творців *Ouroboros*, це єдиний консенсусний алгоритм, безпеку якого можна довести за допомогою математики.

Одним із унікальних аспектів *Cardano* є науковий підхід до розробки. Він базується на дослідженнях та публікаціях у галузі криптографії та комп'ютерних наук. Крім того, *Cardano* поділяє свою архітектуру на два шари, щоб забезпечити ефективнішу обробку транзакцій. Шар розрахунків відповідає за передачу вартості (аналогічно біткоїну), тоді як шар обчислень підтримує виконання смарт-контрактів.

В екосистемі активно розвиваються різні проекти та *DApps*. *Cardano* також активно досліджує можливості співпраці з урядами та корпораціями для створення стійких блокчейн-рішень.

Переваги блокчейну *Cardano*:

- використання наукового підходу у розробці, що сприяє більш фундаментальному та усвідомленому вирішенню проблем блокчейну;
- *Proof-of-Stake* (*PoS*) *Ouroboros* є енергоефективним та більш стійким консенсус-протоколом у порівнянні з традиційним *Proof-of-Work*;

- поділ на Шар розрахунків та Шар обчислень підвищує ефективність мережі та спрощує її масштабування;
- *Cardano* прагне забезпечити високу швидкість транзакцій та масштабованість завдяки використанню складних протоколів;
- активна взаємодія з урядами, освітніми установами та країнами, що розвиваються, що може сприяти більш широкому прийняттю технології блокчейн.

Недоліки:

- *Cardano*, порівняно з старішими блокчейн-платформами, все ще є відносно новою технологією, і її повноцінна ефективність ще має бути доведена в практичному використанні;
- на момент останніх оновлень могло бути менше децентралізованих додатків (*DApps*), що розробляються та запущені на *Cardano* порівняно з конкурентами;
- велика конкуренція серед блокчейн-платформ, через те, що *Cardano* ще не зарекомендував себе так широко як, наприклад, *Ethereum*, ця платформа залишається менш популярною;
- впровадження нових технологій та прийняття їх суспільством завжди пов’язане з низкою труднощів, і *Cardano* не є винятком;
- *Cardano* все ще перебуває в процесі розробки, і його повноцінна реалізація може зайняти багато часу.

Логотип платформи *Cardano* зображений на рис. 1.4.

Рис. 1.4. Логотип *Cardano*

Відмінність блокчейну *Cardano* від *Ethereum* полягає концепції та цілі проекту. *Ethereum* створений для підтримки смарт-контрактів та децентралізованих програм (*DApps*). *Cardano* розробляється з упором на забезпечення безпеки, стійкості та масштабованості блокчейн-мережі. Він також підтримує смарт-контракти, але його розробка ґрунтується на дослідженнях та формальних методах. Для цих двох блокчейнів існують свої мови програмування. Для *Ethereum* цією мовою є *Solidity*, а у *Cardano* – функціональна мова програмування *Plutus*.

1.3. Висновки до розділу

В умовах сучасного росту підприємства все більше проваджують електронний документообіг. Перевагами використання систем електронного документообігу є прозорість всіх етапів діяльності компанії адже кожен документ адже усі документи, завдання чи процеси реєстрації в системі електронного документообігу супроводжуються обліково-реєстраційною інформацією, що допомагає спростити контроль термінів виконання задач, їх відстеження та, аналіз та планування.

Криптографічні хеш-функції базуються на складних алгоритмах шифрування, який видає результат завжди однієї довжини, такий результат і називають хешем, а процес шифрування – хешуванням. Хешування завжди однобоке, «розгорнути» створений хеш-функцією шифр майже не можливо за умови, якщо ця сама хеш функція є безпечною. Початкові данні можна лише підібрати, що займає дуже багато часу та обчислювальних ресурсів.

Існує кілька вразливостей хеш-функцій, включаючи виникнення колізій, пошук першого першовору та пошук другого першовзору. Колізія виникає, коли різні вхідні дані створюють однаковий вихід. В такому хеш-функція вважається стійкою до колізій поки хтось не виявить колізію. Через нескінченну кількість входів і обмежену кількість виходів колізії завжди відбуватимуться у всіх хеш-функціях. Стійкість до знаходження першого першовзору. Ця властивість тісно пов’язана з поняттям односторонніх функцій. Хеш-функція вважається стійкою до знаходження першого першовзору, якщо існує дуже низька ймовірність того, що хтось зможе

знайти вхідні дані, використовуючи згенерований вихід. Стійкість до знаходження другого першовзору. Цей вид стійкості знаходиться між двома попередніми властивостями. Атака знаходження другого першовзору полягає у знаходженні конкретного входу, за допомогою якого можна згенерувати такий вихід, що вже був згенерований за допомогою іншого входу, який уже відомий. Іншими словами, атака знаходження другого першовзору включає виявлення колізії, але замість пошуку двох випадкових входів, які генерують один і той же хеш, атака націлена на пошук входу, за допомогою якого можна відтворити хеш, який вже був згенерований за допомогою іншого входу.

Технологія блокчейн є базовою структурою більшості систем криптовалют і запобігає копіюванню чи знищенню таких цифрових активів. Багато функцій відіграють важливу роль у безпеці блокчейну, але двома найважливішими концепціями є консенсус і незмінність. Консенсус означає здатність вузлів у децентралізованій мережі блокчейну погоджувати справжній стан мережі та дійсність транзакцій. Як правило, консенсус досягається за допомогою так званих консенсусних алгоритмів. Незмінність, з іншого боку, відноситься до здатності блокчейна запобігати зміні вже підтверджених транзакцій. Ці транзакції часто передбачають передачу криптовалют, але можуть також включати запис цифрових даних в інших формах.

Хешування також використовуються в алгоритмах консенсусу і для перевірки транзакцій. Наприклад, блокчейн *Bitcoin* використовує хеш-функцію *SHA*-256 у своєму алгоритмі *Proof of Work* (*PoW*). Як випливає з назви, *SHA*-256 отримує вхідні дані та повертає хеш довжиною 256 біт або 64 символи. Хоча різні хеш-функції створюють хеші різної довжини, можливий вихідний розмір кожного алгоритму хешування завжди постійний. Тобто вихід завжди матиме однакову кількість символів, незалежно від довжини вхідних даних. Наприклад, *SHA*-256 створює 256 біт виводу, тоді як *SHA*-1 завжди видає 160 біт.

РОЗДІЛ 2
АНАЛІЗ ТЕХНОЛОГІЙ РОЗРОБКИ СИСТЕМИ

2.1. Клієнт-серверна архітектура

Розроблювана програмна система електронного документообігу являє собою веб-систему, яка складається з двох основних частин – клієнтської та серверної.

Клієнт-серверна архітектура є основним принципом веб-розробки. Веб-додаток поділяється на дві основні частини: клієнтську (*frontend*) та серверну (*backend*). Клієнтська частина відповідає за інтерфейс користувача та взаємодію з користувачем, тоді як серверна частина обробляє запити від клієнта, управляє бізнес-логікою та взаємодіє з базою даних. Схема роботи клієнт-серверної архітектури зображено на рис. 2.1.

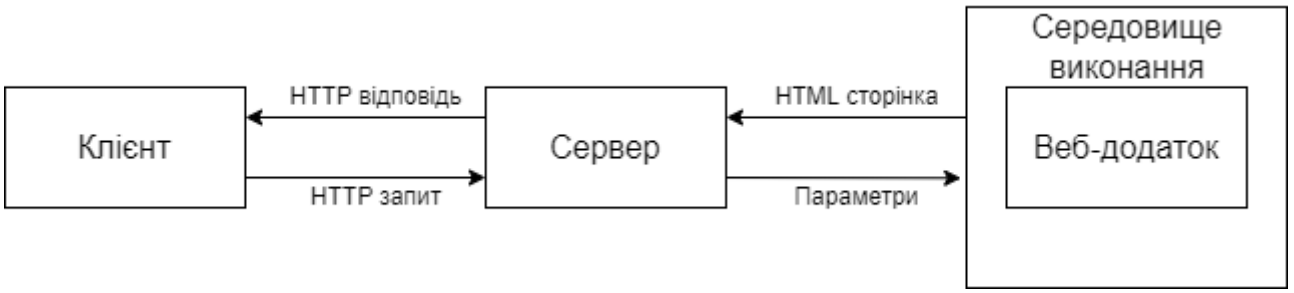

Рис. 2.1. Клієнт-серверна архітектура

На схемі видно, що користувач взаємодіє із додатком через веб-браузер, який посилає запити за допомогою протоколу передачі даних *HTTP* або *HTTPS*, якщо з'єднання має захист у вигляді шифрування інформації. Запит потрапляє на веб-сервер, який передає параметри запиту в середовище виконання веб-додатку. Сам веб-додаток після отримання запиту з параметрами, взаємодіє із базою даних для зчитування, запису, модифікації чи видалення інформації звідти. Після того, як були виконані маніпуляції з інформацією, формується *HTML* сторінка, яка відправляється клієнту через той самий протокол передачі даних, яка потім показується користувачу в браузері.

Деякі функції, які реалізуються на сервері:

- зберігання, доступ, захист та резервне копіювання даних;
- обробка запиту клієнта в залежності від його параметрів;
- відправка відповіді клієнту.

Функції, що реалізується на стороні клієнта:

- представлення інтерфейсу додатку;
- збір даних від користувача;
- формування запитів з параметрами та їх відправка на сервер;
- отримання результатів запиту та відправка додаткових функцій.

Крім того існує кілька концепцій побудови архітектури клієнт-сервер. Слабкий клієнт – сильний сервер. У такій системі вся логіка обробки інформації перенесена на сервер, тобто *backend*, а права у користувача сильно обмежені. Сервер відправляє готову відповідь, яка уже не потребує подальшої обробки, її достатньо тільки вивести на сторінку. Клієнт взаємодіє з користувачем: складає та відправляє запит, приймає результат і виводить інформацію на екран. Сильний клієнт – концепція архітектури, в якій частина логіки обробки інформації знаходить на клієнті. У такому випадку сервер виступає сховищем даних, а вся робота по обробці та подання інформації переноситься на комп’ютер клієнта.

Система (додаток), яка заснована на клієнт-серверній взаємодії, включає три основних компоненти: представлення даних, прикладний компонент, компонент управління ресурсами і їх зберігання. Клієнт-серверну архітектуру поділяють на дворівневу та трирівневу.

Дворівнева клієнт-серверна архітектура складається з двох вузлів – сервер, який відповідає за отримання запитів і відправку відповідей клієнту, використовуючи при цьому лише власні ресурси; клієнт, який представляє користувацький інтерфейс.

Принцип роботи такої системи полягає в тому, що сервер після отримання запиту формує відповідь та відправляє клієнту безпосередньо, без використання сторонніх ресурсів. Дворівнева архітектура зображена на рис. 2.2.

Рис. 2.2. Дворівнева клієнт-серверна архітектура

Трирівнева архітектура складається з трьох компонентів: представлення даних – призначений для користувача інтерфейс; прикладний компонент – сервер додатків; керування ресурсами – сервер бази даних, який надає інформацію.

Принцип роботи полягає в тому, що декілька серверів обробляють запит клієнта. Розподіл операцій знижує навантаження на сервер. Дані зазвичай обробляються на сервері бази даних. Трирівнева клієнт-серверна архітектура зображена на рис. 2.3.

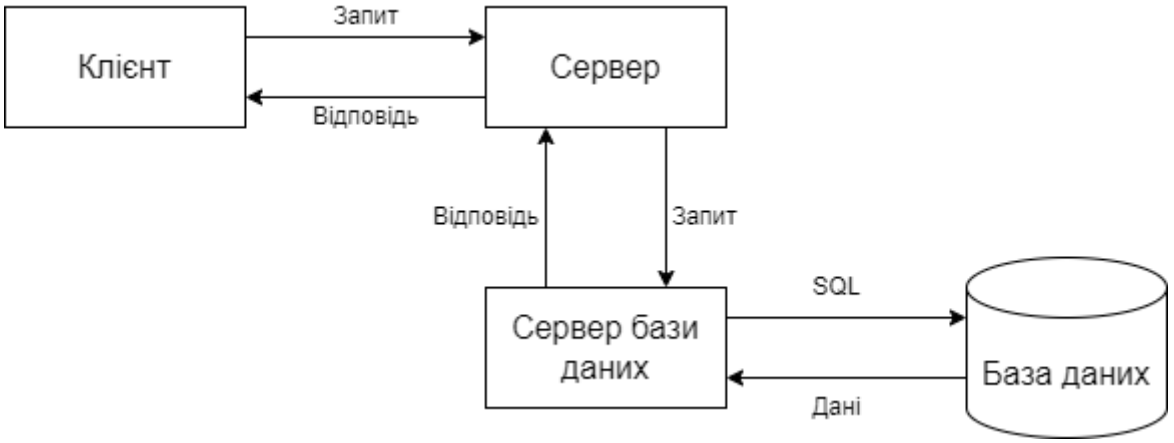

Рис. 2.3. Трирівнева клієнт-серверна архітектура

Трирівневу архітектуру можна розширити до багаторівневої шляхом додавання додаткових серверів. Багаторівнева архітектура дозволяє підвищити ефективність інформаційної системи, а також оптимізувати розподіл її апаратних і програмних ресурсів. Така система зображена на рис. 2.4.

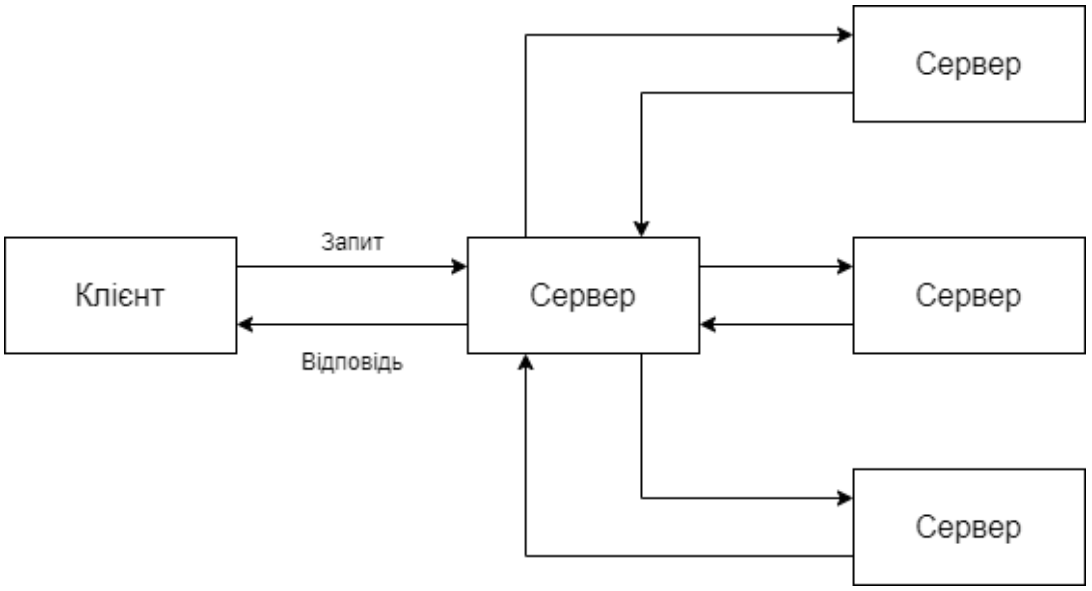

Рис. 2.4. Багаторівнева клієнт-серверна архітектура

Взаємодія клієнт-сервер дозволяє розділяти функціонал і обчислювальне навантаження між клієнтськими і серверними додатками. Знання архітектури додатка дозволяє тестувальнику більш якісно провести функціональне, крос-браузерне тестування, тестування юзабіліті і швидкодії.

2.2. Клієнтська та серверна частини системи

2.2.1. Клієнтська частина системи

Клієнт (або *frontend*) взаємодіє з користувачем. Ця частина додатку відповідає за інтерфейс програми, який відображається в веб-браузері. Для розробки клієнтської частини системи зазвичай використовують мову розмітки *HTML*, таблиці каскадних стилів *CSS* та мову програмування *JavaScript*. *JavaScript* – це широко використовувана мова програмування, яка застосовується у веб-розробці для створення інтерактивних елементів на стороні клієнта (браузер користувача). Крім того, ця мова програмування використовується для надсилання запитів до сервера без перезавантаження сторінки.

Фреймворки та бібліотеки *JavaScript* є інструментами, створеними для полегшення розробки, підвищення продуктивності та стандартизації коду.

Розглянемо найпопулярніші серед фреймворків та бібліотек мови програмування *JavaScript*.

Angular – відкрита та вільна платформа для розробки веб-додатків, написана мовою програмування *TypeScript* (мова програмування, яка являє собою підмножину *JavaScript*), що розробляється командою з *Google*, а також спільнотою розробників з різних компаній. Це потужний та розширюваний фреймворк, який широко використовується для створення складних веб-додатків. Він надає структурований підхід до розробки та безліч інструментів для спрощення завдань розробників. Деякі ключові аспекти фреймворку *Angular*:

- *Angular* додаток будується з компонентів які являють собою незалежні блоки інтерфейсу, які можуть включати шаблони, стилі і логіку;
- *Angular* використовує шаблони визначення візуального представлення компонентів, таким чином дозволяє вбудовувати вирази та директиви безпосередньо у *HTML*;
- *Angular* надає потужну функцію двостороннього зв’язування даних, що означає, що зміни моделі можуть автоматично оновлювати представлення і навпаки. Це знижує необхідність вручну оновлювати *DOM*;
- структурується з використанням модулів, де модуль - це контейнер для компонентів, сервісів та інших функцій, який забезпечує організацію коду та керування залежностями;
- використовує послуги для організації загальної функціональності, такої як обробка *HTTP*-запитів, автентифікація, загальні дані та інші завдання. Сервіси можуть бути впроваджені у компоненти, що дозволяє легко керувати залежностями;
- пропонує директиви для розширення можливостей *HTML*. Наприклад, директива *ngFor* використовується для відображення списків, а *ngIf* – для умови відображення елементів;
- надає інструменти для обробки *HTTP*-запитів, що робить простою взаємодію із зовнішніми даними та *API*;

- *Angular* забезпечує потужний механізм маршрутизації для створення односторінкових додатків (*SPA*). Роутинг дозволяє керувати навігацією між різними компонентами та представленнями;
- *Angular* має вбудовану підтримку для модульного тестування. Код може бути тестований як за допомогою інструментів *Angular CLI*, так і з використанням сторонніх бібліотек для тестування;
- він реалізує командний рядок (*CLI*), який спрощує створення, розгортання та управління проектами.

React – *JavaScript*-бібліотека з відкритим вихідним кодом для розробки інтерфейсів користувача. *React* розробляється та підтримується *Facebook*, *Instagram* та спільнотою окремих розробників та корпорацій. Може використовуватися для розробки односторінкових та мобільних додатків. Його мета – надати високу швидкість розробки, простоту та масштабованість. Як бібліотека для розробки інтерфейсів користувача *React* часто використовується з іншими бібліотеками, такими як *MobX*, *Redux* і *GraphQL*. Основні аспекти бібліотеки *React*:

- основний будівельний блок у *React* – це компонент, компоненти можуть бути маленькими та використовуватися багато разів, що сприяє кращій підтримці коду;
- *React* у своїй структура використовує *JSX*, синтаксис розширення *JavaScript*, що дозволяє писати код, схожий на *XML* або *HTML*, що робить код *React* більш читаним та зручним;
- *React* використовує віртуальний *DOM* для ефективного оновлення лише тих частин сторінки, які змінилися, що покращує продуктивність та оптимізує рендеринг компонентів;
- дані *React* зазвичай передаються зверху вниз (від батька до дочірнього компонента) через властивості (*props*), це забезпечує прозорий потік даних і запобігає неявним змінам;
- компоненти *React* мають стани, які можуть змінюватися протягом часу, а оновлення станів призводить до перерисовки компонента;

- *React* надає методи життєвого циклу, які дозволяють компонентам виконувати певні дії на різних етапах свого існування, наприклад, при створенні, оновленні та знищенні;
- *React* заохочує створення маленьких і використовуваних компонентів, які можна комбінувати в більші і складніші інтерфейси;
- введені в *React* 16.8, хуки надають спосіб використання станів та інших можливостей *React* у функціональних компонентах, раніше доступних лише класовим компонентам;
- *React Router* – це бібліотека для додавання навігації до програми *React*, яка дозволяє створювати односторінкові програми з різними представленнями та маршрутизацією;
- *Redux* – це бібліотека керування станом, яка може використовуватися з *React* для ефективного керування даними у додатку.

Vue.js – це прогресивний *JavaScript*-фреймворк для створення інтерфейсів користувача. Він є відкритим і простим у освоєнні інструментом, який дозволяє розробникам створювати масштабовані та гнучкі веб-додатки. Є гарним вибором для розробки інтерфейсів, особливо для тих, хто віддає перевагу простоті та легкості в освоєнні. Він також добре інтегрується з іншими бібліотеками та інструментами, що робить його гнучким та розширюваним фреймворком. Ось основні концепції та особливості *Vue.js*:

- *Vue.js* підтримує двостороннє зв’язування даних, що дозволяє автоматично оновлювати дані компонента при їх зміні та навпаки;
- додаток *Vue.js* будується з компонентів, які можуть бути маленькими та використовуваними багаторазово і кожен з них є незалежною частиною інтерфейсу;
- *Vue CLI* – офіційний інструмент командного рядка для розробки програм на *Vue.js*, який дозволяє створювати, розгортати та керувати проектами *Vue*;
- *Vue.js* надає директиви, які можна використовувати у шаблонах для додавання різного функціоналу;

- *Vue.js* дозволяє створювати обчислювані властивості, які залежать від інших даних у додатку і можна стежити за даними для реакції на їх зміну;
- компоненти *Vue.js* мають життєвий цикл, який включає різні хуки, такі як *beforeCreate*, *created*, *beforeMount*, *mounted* тощо. Ці хуки дозволяють виконувати код на різних етапах життя компонента;
- *Vue.js* забезпечує реактивність даних, що означає, що зміни даних автоматично оновлюють представлення, і навпаки. Це робить код зрозумілішим і забезпечує легкість відстеження змін;
- *Vue Router* – бібліотека для додавання навігації до *Vue.js*. Дозволяє створювати маршрути, керувати переходами між сторінками та браузерною історією;
- *Vuex* – бібліотека керування станом, яка може використовуватися з *Vue.js* для ефективного керування даними в додатку, особливо при роботі з складнішими станами та взаємодією компонентів.

2.2.2. Серверна частина системи

Backend – це та частина веб-додатків або програмної системи, яка відповідає за обробку даних, бізнес-логіку та взаємодію з базами даних. Він розміщується на серверній стороні та забезпечує основні функції, які не видимі для очей кінцевого користувача, але необхідний для нормального функціонування веб-додатків. Функції та задачі серверної частини:

- приймає *HTTP*-запити від клієнтів (браузерів, мобільних додатків і т. д.); обробляє запити і витягує необхідну інформацію;
- виконує бізнес-логіку, забезпечуючи основні функції додатків; приймає рішення про те, як обробляти дані і взаємодіяти з іншими компонентами системи;
- управляє взаємодією з базою даних, виконує запити та оновлює дані; забезпечує надійність і цілість зберігання даних;

- створює аутентифікацію та авторизацію користувачів; управляє сеансами користувачів і забезпечує безпеку додатків;
- забезпечує завантаження, зберігання і передачу файлів і мультимедійних ресурсів; управляє доступом до різних ресурсів;
- реалізує заходи щодо забезпечення безпеки додатків, включаючи захист від атак і шифрування даних; відстежує та реєструє активності для виявлення потенційних загроз;
- відправляє запити до зовнішніх сервісів або *API*; забезпечує узгодженість та ефективність роботи із зовнішніми ресурсами;
- реалізує механізми кешування для прискорення обробки даних; оптимізує запити до бази даних та інших внутрішніх ресурсів;
- реєструє події та помилки для наступного аналізу; надає метрики та звіти для моніторингу продуктивності та доступності;
- надає механізми для горизонтального масштабування системи; управляє навантаженням і розподілом завдань між різними серверами.

Вибір мови програмування для розробки серверної частини веб-додатку є ключовим моментом, що визначає ефективність і успішність проекту. Для цього існує багато мов програмування і кожна має свої переваги та недоліки. Розглянемо деякі з них.

Python – високорівнева, інтерпретована мова програмування з акцентом на легкість читання коду. *Flask* і *Django* є популярними фреймворками для веб-розробки на *Python*.

Переваги:

- простий і читаний синтаксис;
- загальна бібліотека сторонніх модулів;
- потужні веб-фреймворки (*Django*, *Flask*), які сильно прискорюють розробку.

Недоліки:

- інтерпретованість знижує продуктивність у порівнянні з компільованими мовами програмування;

- *GIL* (*Global Interpreter Lock*) може стати проблемою для паралельного виконання.

Java – об’єктно-орієнтована мова програмування, відома своєю платформною незалежністю. Фреймворки, такі як *Spring*, надають потужні інструменти для створення веб-додатків.

Переваги:

- висока продуктивність і басштабованість;
- повна платформна незалежність завдяки віртуальній машині *Java* (*JVM*);
- багаточисельні інструменти та фреймворки для веб-розробки.

Недоліки:

- потребує більше коду в порівнянні з більш сучасними мовами;
- велике споживання пам’яті.

Ruby – інтерпретована мова програмування. *Ruby on Rails* – фреймворк для веб-розробки на *Ruby*. *Ruby on Rails* сфокусований на угодах перед конфігураціями, що прискорює розробку.

Переваги:

- простий та зрозумілий синтаксис;
- концепція "згод перед конфігураціями" покращує продуктивність;
- активне мережа користувачів і безліч готових рішень.

Недоліки:

- менша продуктивність у порівнянні з деякими мовами, що компілюються;
- менша популярність у порівнянні з іншими мовами.

Go (або *Golang*) – мова програмування, створена *Google*, яка поєднує високу продуктивність і простоту розробки. *Go* часто використовується для створення розподілених систем та веб-серверів.

Переваги:

- висока продуктивність та ефективне використання ресурсів;
- компілюється у виконуваний код, що покращує продуктивність;
- вбудована підтримка паралелізму.

Недоліки:

− відсутність деяких можливостей, доступних в інших мовах;

− менша кількість бібліотек порівняно з мовами, що стали більш усталеними.

PHP (*Hypertext Preprocessor*) – це інтерпретована мова програмування, що широко використовується для веб-розробки. Він призначений для створення динамічних веб-сторінок і може бути вбудований в *HTML*-код.

Переваги:

− простий у освоєнні, що робить його популярним серед новачків;

− велика спільнота та величезна база знань;

− широке використання в промисловості, включаючи великі веб-проекти.

Недоліки:

− деякі розробники вважають *PHP* менш елегантним, ніж сучасні мови;

− низька строгість типів може викликати деякі проблеми у великих проектах.

PHP широко застосовується у веб-розробці для створення динамічних веб-сайтів та програм. Він також використовується як серверна мова для безлічі популярних *CMS* (*Content Management Systems*), таких як *WordPress* та *Drupal*.

Для прискорення та полегшення розробки серверної частини на цих мовах програмування розробники використовують різні фреймворки. Серверні фреймворки являють собою набір інструментів і бібліотек, призначених для спрощення створення *backend*-частини веб-додатків. Вони забезпечують структуру та базовий функціонал, дозволяючи розробникам фокусуватись на реалізації бізнес-логіки, а не на рутинних аспектах розробки серверної частини. Розглянемо деякі із фреймворків для розробки серверної частини системи.

Express (*Node.js*) – фреймворк *web*-додатків для *Node.js*, реалізований як вільне та відкрите програмне забезпечення під ліцензією *MIT*. Він спроектований для створення веб-додатків та *API*. Де-факто є стандартним каркасом для *Node.js*. *Express* є мінімалістичним та гнучким серверним фреймворком для *Node.js*. Він надає набір основних функцій для розробки веб-застосунків та *API*.

Переваги:

− легко освоїти;

− велика спільнота та безліч плагінів;

- підтримка *middleware* для розширення функціоналу.

Недоліки:

- базовий набір функцій може потребувати підключення додаткових бібліотек;
- на великих проектах можуть виникнути труднощі з масштабуванням, оскільки *Express* є мінімальним фреймворком.

Django (*Python*) – вільний фреймворк для веб-застосунків на мові програмування *Python*, що використовує шаблон проектування *MVC*. Проект підтримує організація *Django Software Foundation*. Сайт на *Django* будується з одного або декількох додатків, які рекомендується робити відчужуваними. *Django* – високорівневий фреймворк, орієнтований на швидку та зручну розробку веб-додатків.

Переваги:

- вбудовані інструменти для аутентифікації, адміністрування та роботи з базами даних;
- випереджальна увага до безпеки;
- широкі можливості роботи з базами даних.

Недоліки:

- *Django* може здатися надлишковим для простих або маленьких проектів через великий обсяг вбудованих функцій.

Spring (*Java*) – універсальний фреймворк із відкритим кодом для *Java*-платформи. Існує також форк для платформи *.NET Framework*, названий *Spring.NET*. *Spring Boot* – фреймворк для *Java*, орієнтований спрощення процесу створення додатків, заснованих на *Spring*.

Переваги:

- потужна система керування залежностями;
- інтеграція з іншими проектами у екосистемі *Spring*;
- широкі можливості створення *REST API*.

Недоліки:

- порівняно з деякими фреймворками, може здатися складним для новачків;

- процес конфігурування *Spring Boot* може бути складним і вимагає розуміння безлічі параметрів та налаштувань.

Ruby on Rails (*Ruby*) – фреймворк, написаний мовою програмування *Ruby*, реалізує архітектурний шаблон *Model-View-Controller* для веб-застосунків, а також забезпечує їх інтеграцію з веб-сервером і сервером баз даних. Є відкритим програмним забезпеченням та поширюється під ліцензією *MIT*. Побудований за принципами угод перед змінами, що прискорює розробку.

Переваги:

- продуктивність та швидкість розробки;
- вбудовані інструменти для тестування та адміністрування;
- підтримка парадигми Конвенція за угодою.

Недоліки:

- незважаючи на високу продуктивність на початкових етапах, масштабування *Ruby on Rails* додатків може вимагати додаткових зусиль.

Flask (*Python*) – фреймворк для створення веб-застосунків мовою програмування *Python*, що використовує набір інструментів *Werkzeug*, а також шаблонізатор *Jinja2*. Належить до категорії так званих мікрофреймворків – мінімалістичних каркасів веб-додатків, які свідомо надають лише базові можливості.

Переваги:

- простота та гнучкість;
- відмінно підходить для створення прототипів та невеликих додатків;
- розширюваність за рахунок використання плагінів.

Недоліки:

- не надає вбудованих інструментів для роботи з базами даних або автентифікації.

Laravel (*PHP*) – веб-фреймворк із відкритим кодом, призначений для розробки з використанням архітектурної моделі *MVC*. *Laravel* випущено під ліцензією *MIT*. Вихідний код проекту розміщується на *GitHub*.

Переваги:

- елегантний синтаксис і читаний код;

- вбудовані інструменти для роботи з базами даних, аутентифікації та кешування;
- потужна *ORM* (*Eloquent*) для зручної роботи із базою даних.

Недоліки:

- *Laravel*-додатки можуть споживати більше пам’яті через використання безлічі функціональних можливостей, що може бути проблемою для систем з обмеженими ресурсами.

2.3. Бази даних

Вибір типу бази даних є стратегічним рішенням, що визначає успішність зберігання та обробки даних в інформаційній системі. Для ефективного функціонування програми необхідно ретельно розглянути два основні підходи: реляційні бази даних (*SQL*) та нереляційні бази даних (*NoSQL*). Кожен із цих типів надає свої унікальні характеристики, переваги та недоліки, що робить вибір важливим етапом у проектуванні інформаційних систем. Під час прийняття рішення про вибір між *SQL* і *NoSQL* важливо врахувати особливості проекту, вимоги до даних, рівень масштабованості та бажану гнучкість у роботі з інформацією.

2.3.1. Реляційні бази даних

Реляційна база даних (РБД) – це тип бази даних, яка використовує реляційну модель для організації даних. Реляційна модель описує дані у вигляді таблиць, де дані представлені у вигляді рядків і стовпців. Кожна таблиця представляє собою відношення (*relation*), а кожний рядок у таблиці – кортеж (*tuple*), або запис. Стовпці визначають атрибути даних.

Розроблена Е. Ф. Коддом з *IBM* модель реляційної бази даних дозволяє зв’язувати будь-яку таблицю з іншою таблицею за допомогою загального атрибута. Замість використання ієрархічних структур для організації даних Кодд запропонував

перейти до використання моделі даних, у якій дані зберігаються, отримують доступ і зв’язуються в таблицях без реорганізації таблиць, які їх містять.

Реляційні бази даних можна уявити як набір файлів електронних таблиць, які допомагають компаніям упорядковувати, керувати та зв’язувати дані. У моделі реляційної бази даних кожна «електронна таблиця» – це таблиця, яка зберігає інформацію, представлену у вигляді стовпців (атрибутів) і рядків (записів або кортежів).

Атрибути (стовпці) визначають тип даних, і кожен запис (або рядок) містить значення цього конкретного типу даних. Усі таблиці в реляційній базі даних мають атрибут, відомий як первинний ключ, який є унікальним ідентифікатором рядка, і кожен рядок можна використовувати для створення зв’язку між різними таблицями за допомогою зовнішнього ключа – посилання на первинний ключ іншої таблиці. Приклад реляційної ази даних зображений на рис. 2.5.

Рис. 2.5. Реляційна база даних

Серед основних характеристик реляційних баз даних є мова запитів *SQL* (через неї реляційні бази даних у народі називають *SQL* базами даних).Для маніпуляції даними використовуються різного виду команди, серед яких *SELECT*, для вибору даних із таблиці; *INSERT*, для додавання інформації в таблицю; *UPDATE* використовують для редагування даних; *DELETE* – видалення рядку даних за спеціальною умовою. Запити можуть включати в себе умови та з’єднання для фільтрації і поєднання даних. Крім того мова *SQL* може брати на себе частину логіки оброки даних, яка може знаходитися на сервері, для цього існує безліч функцій для

роботи з текстом, математичні функції та навіть умови *IF*(), які можуть повертати різні дані в залежності від того, який результат вибірки ви отримали.

Зовнішній ключ являє собою стовпчик або набір стовбців у таблиці, який посилається на первинний ключ іншої таблиці. Це створює зв'язок між двома таблицями, дозволяючи одній таблиці посилатися на дані в іншій. Первинний ключ (*Primary key*) – це унікальний ідентифікатор кожного запису в таблиці. Кожна таблиця може мати тільки один первинний ключ. Первинний ключ в таблиці стає зовнішнім ключем в іншій таблиці, встановлюючи зв'язок між ними. Припустимо, у нас є дві таблиці: «Клієнти» (табл. 1.1) та «Замовлення» (табл. 1.2).

Таблиця 1.1

Таблиця «Клієнти»

Customer_id	***Customer_name***	***Country***
1	Компація А	США
2	Компація А	Канада
3	Компація А	Британія

Таблиця 1.2

Таблиця «Замовлення»

Order_id	***Order_date***	***Customer_id***	***Total_amount***
101	2023-01-15	1	500
102	2023-02-10	2	750
103	2023-04-12	3	1200

У таблиці «Замовлення» стовпець *Customer_id* є зовнішнім ключем, який посилається на первинний ключ таблиці «Клієнти». Це означає, що кожен запис у таблиці «Закази» пов'язаний із конкретним клієнтом з таблиці «Клієнти».

ACID – це абревіатура, що представляє чотири основні принципи, які забезпечують надійність та цілісність даних у транзакційних системах.

Атомарність (*Atomicity*) гарантує, що транзакцію буде виконано цілком або не буде виконано взагалі. Якщо одна частина транзакції не може бути завершена, вся транзакція відкочується до попереднього стану, і система залишається в незміненому стані.

Приклад: якщо транзакція включає переказ грошей з рахунку А на рахунок В, і переказ на рахунок В завершується успішно, але переказ з рахунку А невдалий (наприклад, через збій), то вся транзакція відкочується, і гроші не переказуються.

Узгодженість (*Consistency*) означає, що транзакція переводить базу даних із одного узгодженого стану до іншого. Внаслідок виконання транзакції база даних залишається в цілісному стані з урахуванням усіх правил та обмежень.

Приклад: якщо ми маємо правило, що сума грошей на всіх рахунках не повинна бути негативною, то після завершення транзакції це правило має залишитися дотриманим.

Ізольованість (*Isolation*) означає, що виконання однієї транзакції не впливає виконання інших транзакцій, що відбуваються паралельно. Це означає, що результати транзакції не стають видимими для інших транзакцій до завершення.

Приклад: Якщо дві транзакції виконуються паралельно, кожна з них повинна бачити базу даних у стані, якби вона була виконана послідовно.

Довговічність (*Durability*) гарантує, що результати успішно завершеної транзакції залишаться незмінними навіть у разі збоїв системи. Після підтвердження завершення транзакції, її результати зберігаються в базі даних і залишаються стійкими до будь-яких подальших збоїв.

Приклад: Якщо транзакція успішно завершена, дані, внесені нею в базу даних, повинні бути збережені навіть при вимкненні електроживлення або інших збоїв.

Принципи *ACID* забезпечують надійність транзакцій у реляційних базах даних, що дуже важливо для підтримки цілісності даних у додатках, де потрібна точність і надійність операцій із базою даних.

Переваги реляційних баз даних:

- такі бази даних ідеальні для зберігання структурованих даних, де існує чітка схема;

- *ACID* принципи забезпечують надійність і цілісність даних;
- *SQL* надає засоби для запитів та аналітики.

Недоліки реляційних баз даних:

- зміна схеми може бути складною і вимагає зупинки системи;
- вертикальне масштабування може стати дорогим зі збільшенням обсягу даних;
- для обробки великих обсягів даних можуть знадобитися складні запити.

2.3.2. Нереляційні бази даних

Нереляційна база даних – це база даних, яка не використовує табличну схему рядків і стовпців, яка є в більшості традиційних систем баз даних. Натомість нереляційні бази даних використовують модель зберігання, оптимізовану для конкретних вимог типу даних. Наприклад, дані можуть зберігатися як прості пари ключ/значення, як документи *JSON* або як граф, що складається з ребер і вершин.

Спільним для всіх цих сховищ даних є те, що вони не використовують реляційну модель. Крім того, вони, як правило, більш конкретні щодо типу даних, які вони підтримують, і того, як дані можна запитувати. Наприклад, сховища даних часових рядів оптимізовані для запитів до послідовностей даних на основі часу. Однак сховища даних графіків оптимізовано для дослідження зважених зв’язків між об’єктами. Жоден із форматів не підійде для керування транзакційними даними.

Термін *NoSQL* стосується сховищ даних, які не використовують *SQL* для запитів. Натомість сховища даних використовують інші мови програмування та конструкції для запиту даних. На практиці «*NoSQL*» означає «нереляційну базу даних», хоча багато з цих баз даних підтримують *SQL*-сумісні запити. Однак базова стратегія виконання запиту зазвичай сильно відрізняється від того, як традиційна *RDBMS* виконувала б той самий *SQL*-запит.

Серед основних характеристик нереляційних баз даний можна виділити гнучкість системи. У *NoSQL* базах даних схема даних може бути динамічно змінена, що забезпечує більшу гнучкість у роботі з різними типами даних без строгих вимог

до структури. Крім цього ще можна зазначити, що *NoSQL* бази даних легко масштабуються горизонтально, дозволяючи додавати нові вузли для обробки обсягу даних, що збільшується.

Існує кілька різних типів *NoSQL* баз даних, включаючи:

- зберігають дані у форматі документів (*JSON*, *BSON*);
- зберігають дані у вигляді пар ключ-значення;
- організують дані у вигляді стовпців замість рядків;
- призначені для зберігання та обробки даних у вигляді графів;
- набір значень, упорядкованих за часом.

Переваги *NoSQL* баз даних:

- *NoSQL* бази даних добре підходять для роботи з великими обсягами неструктурованих даних;
- відрізняються високою продуктивністю та масштабованістю, особливо при горизонтальному масштабуванні;
- легко масштабуються горизонтально за рахунок додавання нових вузлів у кластер.

Недоліки *NoSQL* баз даних:

- відсутність загальних стандартів та мови запитів може ускладнити розробку та обслуговування систем;
- адміністрування *NoSQL* баз даних може вимагати спеціалізованих навичок та інструментів;
- деякі типи *NoSQL* баз даних (особливо бази ключ-значення) можуть мати обмежені можливості забезпечення *ACID*-властивостей в порівнянні з реляційними базами даних.

Бази даних *NoSQL* пропонують різноманітні інструменти для маніпулювання даними, залежно від потреб вашої конкретної програми. Вибір між реляційними базами даних і базами даних *NoSQL* залежить від вашого типу даних, вимог до продуктивності та масштабованості, а також уподобань розробки.

2.4. Висновки до розділу

Розроблювана програмна система електронного документообігу являє собою веб-систему, яка складається з двох основних частин – клієнтської та серверної. Клієнт-серверна архітектура є основним принципом веб-розробки. Веб-додаток поділяється на дві основні частини: клієнтську (*frontend*) та серверну (*backend*). Клієнтська частина відповідає за інтерфейс користувача та взаємодію з користувачем, тоді як серверна частина обробляє запити від клієнта, управляє бізнес-логікою та взаємодіє з базою даних.

Існує кілька концепцій побудови архітектури клієнт-сервер. Слабкий клієнт – сильний сервер. У такій системі вся логіка обробки інформації перенесена на сервер, тобто *backend*, а права у користувача сильно обмежені. Сервер відправляє готову відповідь, яка уже не потребує подальшої обробки, її достатньо тільки вивести на сторінку. Клієнт взаємодіє з користувачем: складає та відправляє запит, приймає результат і виводить інформацію на екран. Сильний клієнт – концепція архітектури, в якій частина логіки обробки інформації знаходить на клієнті. У такому випадку сервер виступає сховищем даних, а вся робота по обробці та подання інформації переноситься на комп’ютер клієнта.

Клієнтська частина взаємодіє з користувачем та відповідає за інтерфейс програми, який відображається в веб-браузері. Для розробки клієнтської частини системи зазвичай використовують мову розмітки *HTML*, таблиці каскадних стилів CSS та мову програмування *JavaScript*. *JavaScript* - це широко використовувана мова програмування, яка застосовується у веб-розробці для створення інтерактивних елементів на стороні клієнта (браузер користувача). Крім того, ця мова програмування використовується для надсилання запитів до сервера без перезавантаження сторінки.

Серверна частина, або *backend* – це та частина веб-додатків або програмної системи, яка відповідає за обробку даних, бізнес-логіку та взаємодію з базами даних. Він розміщується на серверній стороні та забезпечує основні функції, які не видимі

для очей кінцевого користувача, але необхідний для нормального функціонування веб-додатків.

Серверна частина обробляє запити та взаємодіє з базою даних. Вибір бази даних є важливим рішенням, що визначає успішність зберігання та обробки даних в інформаційній системі. Всього існує кілька типів баз даних: реляційні бази даних (*SQL*) та нереляційні бази даних (*NoSQL*). Кожен із цих типів надає свої унікальні характеристики, переваги та недоліки. Важливо врахувати особливості проекту, вимоги до даних, рівень масштабованості та бажану гнучкість у роботі з інформацією.

Реляційна база даних – це набір інформації, яка розміщується у таблицях, рядки яких відповідають одному запису в таблицю, а стовпчики – одному пункту запис, між таблицями створюються зв'зки, через це такі бази даних і називаються реляційними. Зв'язки – це логічне зв'язування різних таблиць за допомогою спеціальних ключів та унікальних даних для кожного запису. Такі бази даних використовують *SQL* (*Structured Query Language*) для визначення та маніпуляції даними.

Нереляційна база даних – це база даних, яка не використовує схему таблиць, рядків і стовпців, яка є в більшості традиційних систем баз даних. Натомість нереляційні бази даних використовують моделі зберігання, оптимізовані для конкретних потреб типу даних. Наприклад, дані можна зберігати як прості пари ключ-значення, документи *JSON* або графи, що складаються з ребер і вершин.

Спільним для цих сховищ даних є те, що вони не використовують реляційну модель. Крім того, прийнято детально описувати типи підтримуваних даних і спосіб запиту даних. Наприклад, сховища даних часових рядів оптимізовано для запитів послідовностей даних на основі часу. Однак сховища даних графів оптимізовано для дослідження зважених зв'язків між об'єктами. Жоден формат не підходить для керування транзакційними даними.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

3.1. Апаратні та програмні вимоги

Для забезпечення ефективної роботи системи електронного документообігу з використанням технології блокчейн важливо врахувати апаратні вимоги до середовища, які описані в таблиці 3.1.

Таблиця 3.1

Апаратні вимоги

Компонент	**Вимоги**
Процесор	*Intel Xeon E5* або аналогічний
Оперативна пам’ять	Мінімум 32 ГБ *ECC RAM*
Системний накопичувач	*HDD* або *SSD* мінімум 500 Гб
Криптографічні прискорювачі	Підтримка апаратної криптографії
Системи захисту інформації	*Firewall* та *IDS*/*IPS* та апаратне шифрування
Системи резервного копіювання	Копіювання на зовнішній носій
Можливість масштабування	Легкозамінні та масштабовані компоненти

Ефективна реалізація системи електронного документообігу на основі технології блокчейн вимагає певних програмних компонентів та середовища. Ключові вимоги описані в таблиці 3.2.

Таблиця 3.2

Програмні вимоги

Компонент	**Вимоги**
Операційна система	*Ubuntu Server* 20.04 *LTS* (або аналогічна)
Блокчейн-Нода *Ethereum*	Програмний пакет *Geth* (*Go Ethereum*)
Смарт-контракти	Компілятор *Solidity*
База даних	СУБД *MySQL* або *PostgreSQL*
Ідентифікація та аутентифікація	*OAuth* або *OpenID Connect*
Інструменти безпеки	*SSL/TLS* для шифрування. *Firewall* та *Fail2Ban*
Системи резервного копіювання	*Automysqlbackup* (для *MySQL*) або pg_*dump* (для *PostgreSQL*)
Web3 Бібліотеки	*Web3.php*
Інструменти моніторингу та аудиту	*Prometheus*, *Grafana*

3.2. Структура програмної системи

Для розробки програмної системи електронного документообігу з використанням технології блокчейн було використано такі технології:

- веб-браузер *Google Chrome*;
- інтегроване середовище розробки (*IDE*) *JetBrains PhpStorm*;
- мова гіпертекстової розмітки *HTML*;
- бібліотека шаблонів для *HTML Bootstrap5*;
- таблиці каскадних стилів *CSS*;
- серверна мова програмування *PHP*;
- шаблон *MVC* (*Model-View-Controller*);
- шаблон *ORM*;
- шаблон *Active Record*;
- система управління базами даних *MySQL*;

- бібліотека мови *JavaScript jQuery*;
- блокчейн платформа *Ethereum*.

За архітектуру серверної частини відповідає шаблон проектування *MVC* (*Model-View-Controller*). Основний принцип цього шаблону полягає у розділі логіки між сутностями. За обробку, зчитування, додавання та видалення інформації відповідає модель (*Model*). Тут представлена логіка отримання та запису в базу. Для полегшеної роботи з базами даних в моделях використовують *Active Record* та *ORM* (*Object Relational Mapping*).

За бізнес логіку системи у цьому шаблоні проектування відповідає контролер (*Controller*). Контролер оперує моделями та інформацією з бази даних, яку він отримує за допомогою моделей та передає уже оброблені дані у представлення (*View*). Представлення відповідає за вивід інформації користувачу. Отримуючи дані від контролера, представлення відтягує потрібні шаблони сторінок, передає туди всю отриману інформацію і після цього вже готова сторінка виводиться користувачу у браузер. Представлення також може відповідати за збір інформації від користувача та передачі її контролеру для подальшої обробки. Для цього зазвичай використовуються форми та різні тригери на сторінці браузера.

3.2.1. Клієнтська частина

Під час розробки клієнтської частини системи для полегшення а прискорення створення структури та зовнішнього вигляду інтерфейсу було використано бібліотеку шаблонів *Bootstrap* версії 5. Це популярна бібліотека, яка в собі містить шаблони різних блоків, які постійно використовуються у веб-розробці. Наприклад форми, навігаційні панелі, кнопки, групи кнопок, модальні вікна, випадаючі списки та багато іншого.

Крім того основною особливістю *Bootstrap* є саме створення структури сторінки, яка автоматично підлаштовується під ширину екрану користувача. Така структура складається із кореневого блоку з класом *container*, якщо потрібно, щоб максимальна ширина контенту була 1140 пікселів, або *container-fluid*, якщо потрібно

розширити контент на усю ширину сторінки або блоку. Після цього у кореневому блоці формується структура із строк у вигляді блоків із класом *row*. Таких блоків може бути безліч в одному кореневому блоці. У кожній такій стрічці є по 12 колонок (саме через це структуру *Bootstrap* називають сіткою, бо сторінка ділиться на стрічки та колонки утворюючи сітку) які можна використовувати у будь-якому порядку, поєднувати в групи та навіть у кожній колонці можна створити кореневий блок *container* із такою самою структурою, утворивши таким чином ієрархічну структуру. В блоці *row* створюються блоки з класами *col* та вказується кількість колонок, які треба виділити під цей блок, наприклад *col*-3. В такому випадку блок буде займати 3 колонки. Ми можемо використати ще 9 колонок, при цьому можна задіяти одразу всі вільні створивши блок *col*-9 чи розділити на якусь кількість блоків, наприклад *col*-3, *col*-4 та *col*-2. Структурна сітка *Bootstrap* зображена на рис. 3.1.

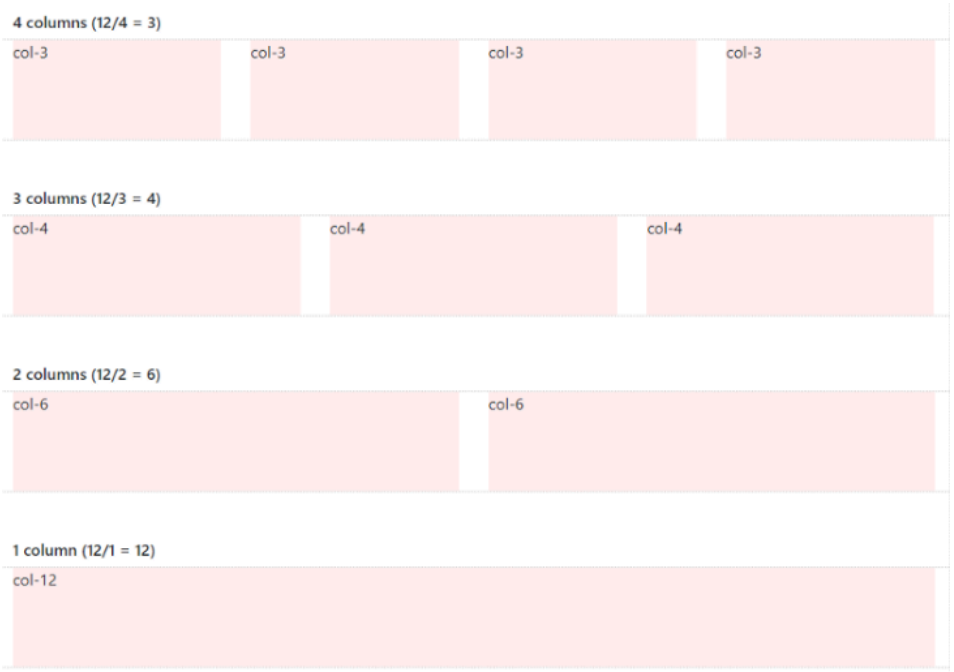

Рис. 3.1. Структурна сітка *Bootstrap*

У *Bootstrap* адаптація макета під різну ширину вікна реалізовано за допомогою медіа-запитів. Кожен медіа-запит у *Bootstrap* будується на підставі мінімальної ширини *viewport* браузера. Ключове значення ширини *viewport* в медіа-запиті називається *breakpoint* (контрольною точкою) (рис. 3.2).

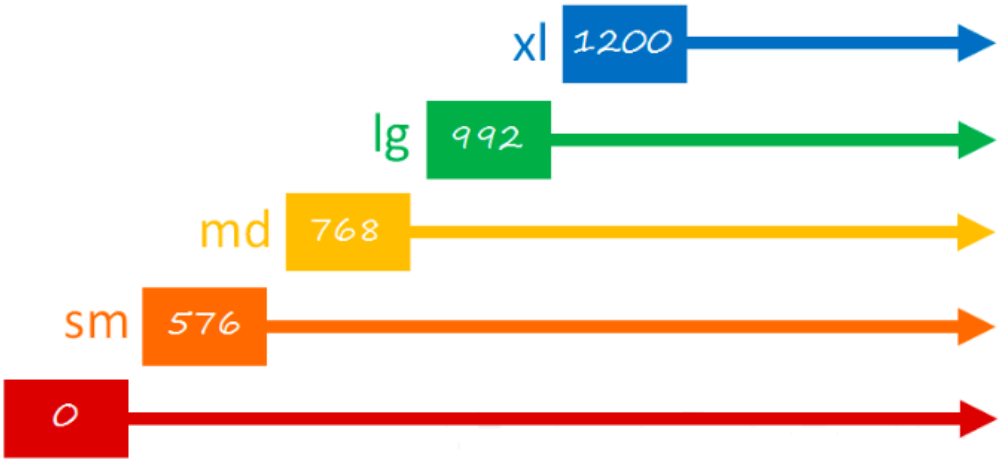

Рис. 3.2. Контрольні точки Bootstrap сітки

Це означає, що до *576px* макет сайту може відображатися одним чином, від *576px* до *768px* – іншим і т.д. Таким чином, можна зробити макет, який на кожній із цих ділянок може виглядати по-різному.

Контрольні точки мають позначення. Перший *breakpoint* не має позначення (*xs*), другий називається – *sm*, третій – *md*, 4-ий – *lg* і 5-ий – *xl*. Ці позначення в імені класу будуть вказувати на те, з якої ширини *viewport* стилі, визначені в ньому, будуть застосовуватися до елемента. З використанням контрольних точок блоки колонок будуть мати такі класи: *col-sm-2*, *col-md-3*, *col-lg-6* і так далі. Відповідність ширини вікна до контрольної точки показано в таблиці 3.3.

Таблиця 3.3

Відповідність ширини вікна до контрольної точки

Позначення	**Розмір вікна**	**Назва класу**
xs	< 576 пікселів	*col-*
sm	≥ 576 пікселів	*col-sm-*
md	≥ 768 пікселів	*col-md-*
lg	≥ 992 пікселів	*col-lg-*
xl	≥ 1200 пікселів	*col-xl-*

За інтерактивність клієнтської частини системи відповідає мова програмування *JavaScript*. *jQuery* – це швидка та легка *JavaScript* бібліотека, яка

спрощує взаємодію з *HTML*-документами, обробку подій, анімації та взаємодію з *AJAX* для асинхронного обміну даними з веб-сервером. Вона була розроблена з метою спростити складні та часті операції, які розробники виконують з допомогою *JavaScript*. Основні функції та можливості *jQuery* включають:

- *jQuery* дозволяє зручно вибирати *HTML*-елементи за допомогою *CSS*-подібних селекторів. Наприклад, ви можете вибрати всі елементи <*p*> на сторінці або всі елементи з конкретним класом. Приклад вибірки елемента за допомогою *jQuery* та :

  $("p");
- *jQuery* надає зручний інтерфейс для обробки подій, таких як клік, зміна розміру, введення миші тощо. Приклад обробки події кліку по кнопці:

  $("button").click(function(){

  alert("Кнопка була натиснута!");

  });
- за допомогою *jQuery* можна легко створювати анімації для *HTML*-елементів, такі як згортання, розгортання, зміна прозорості тощо. Приклад створення анімації для блоку:

  $("div").animate({left: '250px'});
- *jQuery* надає простий та зручний інтерфейс для виконання асинхронних *HTTP*-запитів, що дозволяє обмінюватися даними між клієнтом та сервером без перезавантаження сторінки. Приклад роботи з *AJAX* запитами:

  $.ajax({

  url: "server.php",

  success: function(result){

  $("#div1").html(result);

  }

  });

Загалом, використання *jQuery* дозволяє швидше та зручніше взаємодіяти з *DOM*, обробляти події та виконувати анімації, що полегшує процес розробки веб-додатків також ця бібліотека є фундаментом для багатьох фреймворків

JavaScript. Однак з часом з'явилися нові стандарти *JavaScript*, які також забезпечують схожий функціонал, і в деяких випадках може бути вигідним використовувати їх напряму, обираючи сучасні підходи до розробки веб-додатків.

3.2.2. Серверна частина

Серверна частина програмної системи електронного документообігу з використанням технології блокчейн працює на серверній мові програмування *PHP* 8.0. Програма приймає *HTTP* запити від клієнта, кожен запит потрапляє у відповідний контролер, який потім оперує отриманими параметрами запиту та моделями для збереження чи зчитування інформації із баз даних.

Діаграма класів контролерів зображена на рис. 3.3.

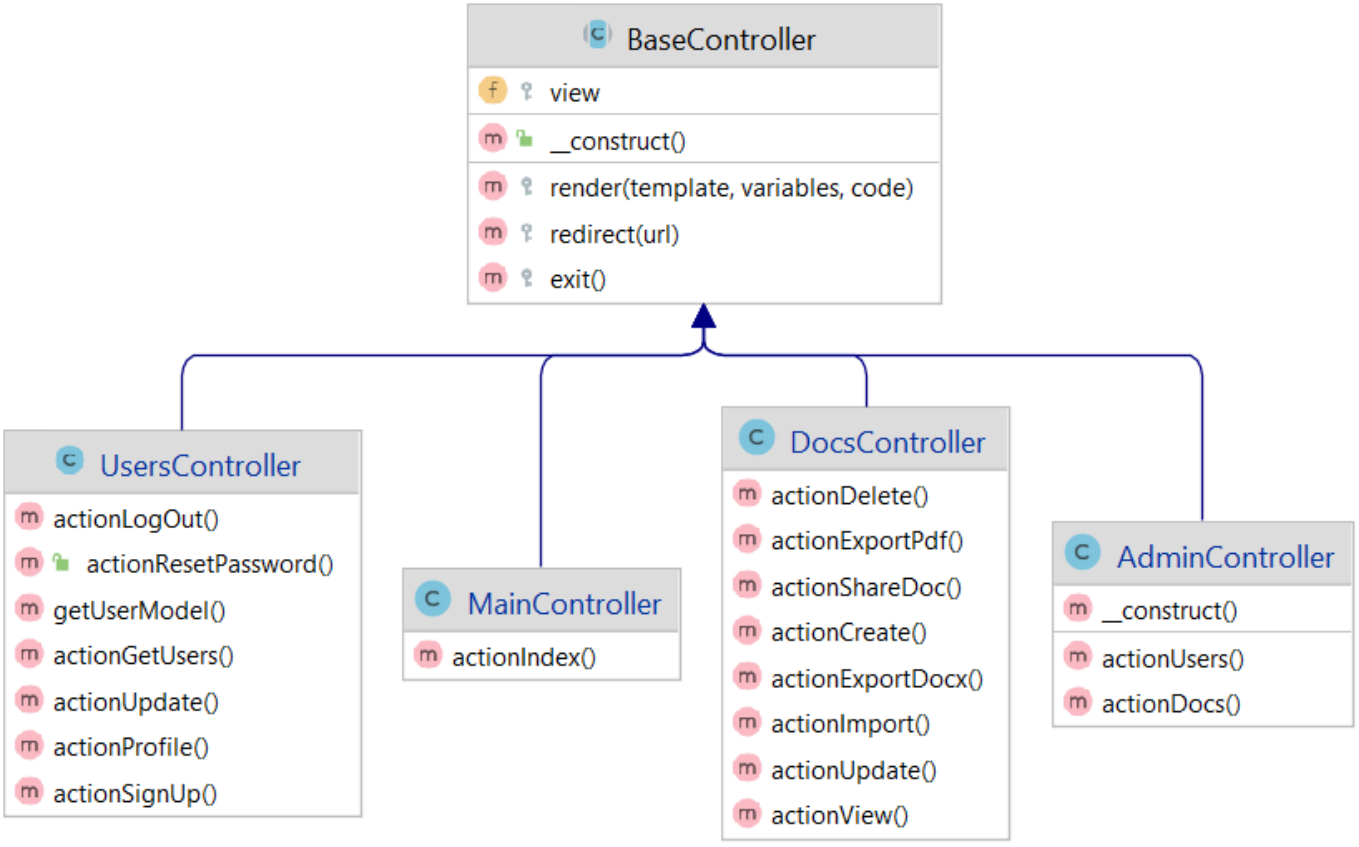

Рис. 3.3. Діаграма класів контролерів

Кожен контролер наслідує базовий клас, у якого є набір базових методів які використовуються у класах-нащадках. За обробку даних відповідають спеціальні

методи контролерів, які називають екшн та назва таких методів починається із *action*.

Маршрутизація у системі реалізована таким чином, що у *url* запиті вказується назва контролеру та назва екшену, після чого можуть іти *GET*-параметри запиту. Приклад url:

/users/profile?id=1

У цьому рядку частина *users* це назва контролеру, тобто обирається клас UsersControler.php (назва класу-контролеру завжди закінчується словом *Controler*), та назва методу, та *profile* – назва методу, який буде обробляти запит (методи, які беруть участь у маршрутизації завжди починаються на *action*). Окремий контролер відповідає за обробку інформації окремої сутності, наприклад за обробку запитів документів відповідає *DocsControler*.

За роботу з базами даних відповідають класи-моделі які реалізують логіку шаблону *Active Record*. Цей шаблон проектування забезпечує зручні методи для роботи з базами даних. Діаграма класів моделей зображена на рис. 3.4.

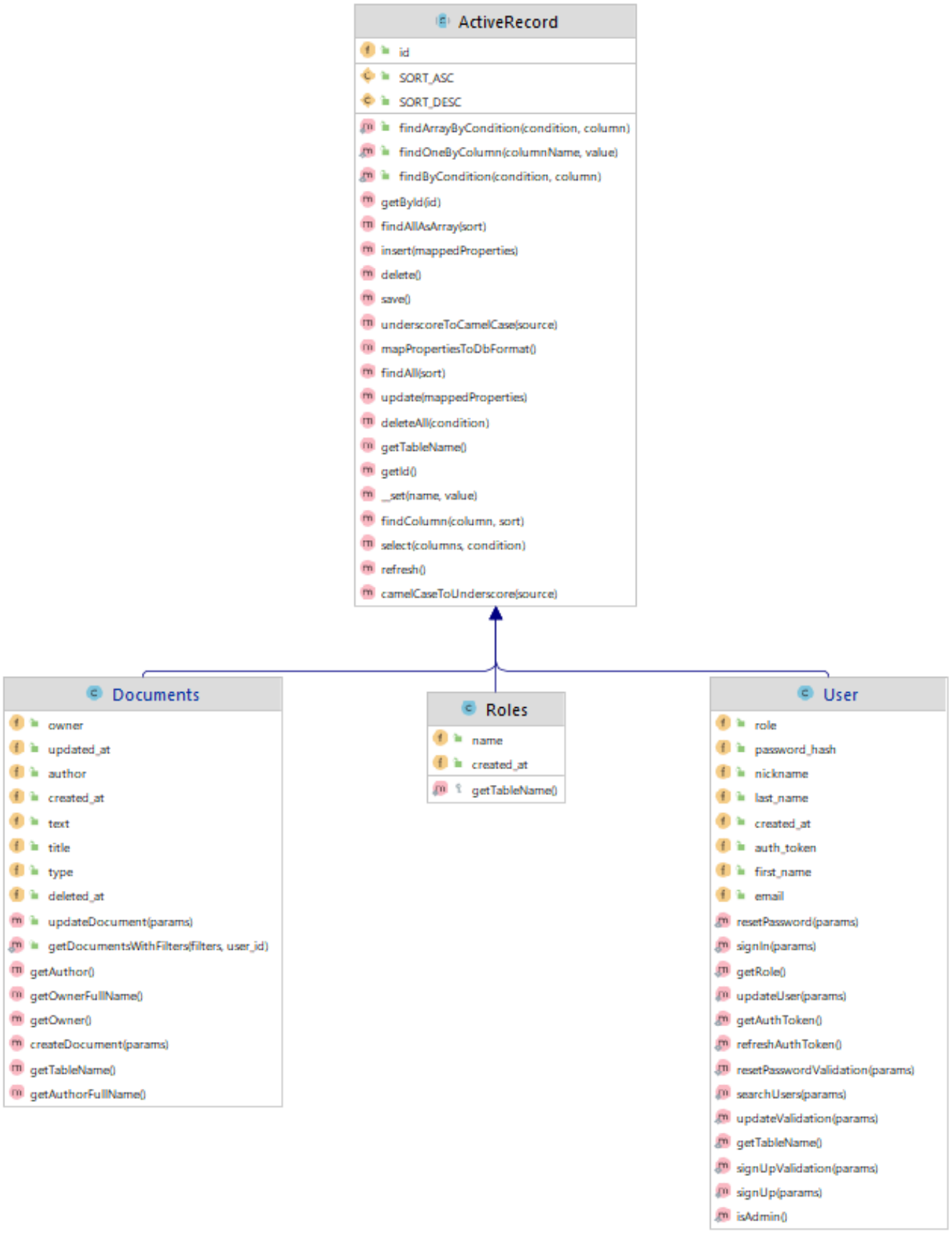

Рис. 3.4. Діаграма класів моделей

Кожен клас тут наслідує базовий клас *ActiveRecord* який в собі і реалізує всю логіку шаблону *ActiveRecord*. Крім цього, класи-моделі також реалізують логіку шаблону проектування *ORM* (*Object Relational Mapping*) або об'єктно-реляційного відображення. Це підхід до роботи з базами даних, який дозволяє взаємодіяти з ними за допомогою об'єктно-орієнтованого програмування. Замість того, щоб вручну писати *SQL*-запити для роботи з базою даних, можна використовувати об'єктно-орієнтовані класи для представлення даних та їх взаємодії з базою даних. Взаємодія з базами даних при такому підході розробки полягає в тому, що одна модель відповідає за одну таблицю в базі, об'єкт моделі – запис в таблиці, а кожна змінна класу відповідає за одну колонку в записі.

3.2.3. Опис баз даних

Бази даних розроблюваної системи електронного документообігу з використанням технології блокчейн побудовані на СУБД *MySQL*. Система управління базами даних є програмним забезпеченням, яке відповідає за ефективне та надійне зберігання, організацію, взаємодію та управління базою даних. Основні функції СУБД:

- створення та визначення схеми бази даних, включаючи таблиці, поля, індекси, обмеження та інші об'єкти;
- зберігання фізичної структури даних на диску та їх організація для швидкого доступу та ефективного використання ресурсів;
- надання мови запитів, яку використовують розробники та користувачі для взаємодії з базою даних, також *SQL* (*Structured Query Language*) є однією з найпоширеніших мов запитів для реляційних баз даних;
- дозвіл на вставку нових даних, оновлення існуючих записів та видалення зайвих даних в базі даних;
- забезпечення цілісності даних шляхом встановлення обмежень, які контролюють правила вставки, оновлення та видалення даних;

- реалізація механізмів для забезпечення відмовостійкості та відновлення даних у випадку непередбачуваних ситуацій, таких як відмова жорсткого диска чи системи;
- забезпечення захисту даних та обмеження доступу до бази даних, визначення прав доступу для користувачів та ролей;
- використання оптимізаторів запитань для покращення ефективності виконання *SQL*-запитів та оптимізації використання ресурсів.

До найпопулярніших реляційних баз даних відносяться *MySQL, PostrgeSQL, Microsoft SQL Server, Oracle Database SQLite* та *MariaDB*.

При розробці системи електронного документообігу для полегшення роботи з СУБД було використано плагін для *IDE JetBrains PhpStorm*. Він має вигляд секції, у якій відображаються всі бази даних з їх схемами (рис. 3.5).

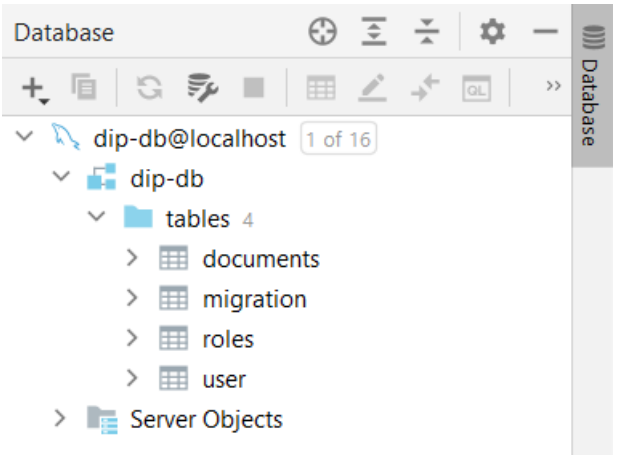

Рис. 3.5. Плагін для роботи з азами даних для *IDE PhpStorm*

Під час розробки було створено кілька таблиць, деякі х них мають зв’язки. Всього існує кілка видів зв’язків між таблицями в реляційних базах даних:

- один до одного (1:1) – один запис в одній таблиці відповідає одному запису в іншій таблиці;
- один до багатьох (1:Б) – один запис в одній таблиці може відповідати багатьом записам в іншій;
- багато до багатьох (Б:Б) – складний зв’язок між двома таблицями, для реалізації якого потрібно створювати додаткову таблицю, у якій будуть

міститися записи зв’язків між двома таблицями. Багато записів у одній таблиці можуть відповідати багатьом записам у іншій.

Опис зв’язків бази даних системи електронного документообігу показано в таблиці 3.4.

Таблиця 3.4

Опис зв’язків між таблицями

Таблиця 1	**Зв’язок**	**Таблиця 2**
roles	1:Б	*users*
user	1:Б	*documents*

Опис таблиць бази даних:

- *Documents* – таблиця, в якій знаходяться записи створених у системі документів;
- *Migrations* – облік виконаних транзакцій в системі;
- *Roles* – список ролей, які можуть бути присвоєні користувачам. Від ролі користувача залежить те, якими функціями він може користуватись та яка інформація йому доступна;
- *User* – таблиця користувачів. Паролі користувачів записуються у хешованому вигляді для підвищення безпеки злому облікових записів, якщо дані бази даних були викрадені.

Діаграма зв’язків між таблицями зображена на рис. 3.6.

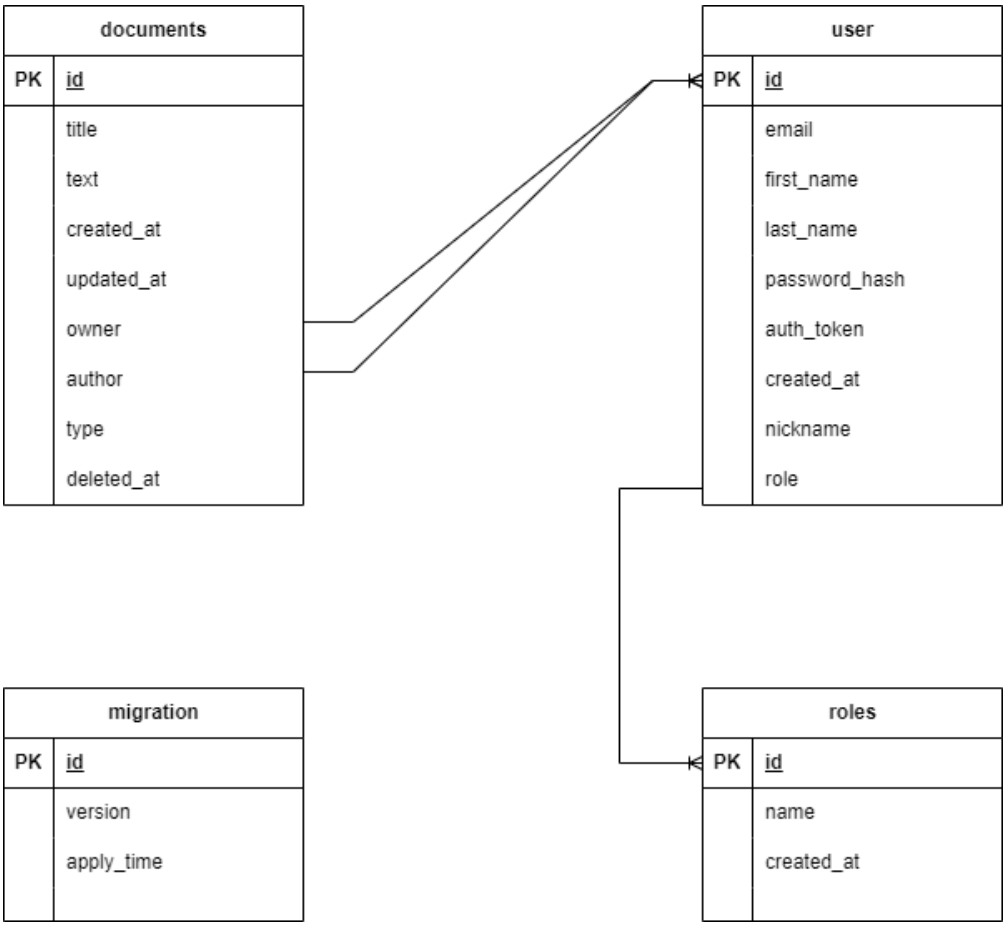

Рис. 3.6. Діаграма зв’язків між таблицями

3.3. Блокчейн у системі електронного документообігу

3.3.1. Взаємодія блокчейну з мовою програмування *PHP*

Програма, написана на мові програмування *PHP* може взаємодіяти з блокчейном *Ethereum* та смарт-контрактами до нього за допомогою бібліотеки *Web3.php*. Це бібліотека для мови програмування *PHP*, призначена для взаємодії з *Ethereum* блокчейном. Вона дозволяє створювати *PHP*-додатки, які можуть виконувати різні операції на *Ethereum*, такі як створення транзакцій, виклик смарт-контрактів та читання стану блокчейну. Основні можливості *web3.php* включають:

- бібліотека дозволяє встановлювати з’єднання з *Ethereum*-нодою. Можна обрати місце розташування ноди або використовувати власну, якщо вона

налаштована. Це важливо для взаємодії з мережею *Ethereum*. Приклад підключення до *Ethereum*-ноди та отримання акрсії:

$web3 = new Web3('http://localhost:8545');

$netVersion = $web3->net->version();

- *Web3.php* дозволяє створювати та підписувати *Ethereum*-транзакції. Можна вказувати отримувача, значення, газ та інші параметри. Після підписання транзакції вона може бути відправлена на мережу *Ethereum*;
- за допомогою *web3.php* можна викликати функції смарт-контрактів, передаючи відповідні дані та параметри. Також можна читати дані з смарт-контрактів без необхідності відправки транзакції;
- бібліотека дозволяє створювати пари ключів (приватний та публічний) для здійснення підписів та ідентифікації користувачів. Це важливо для безпечної взаємодії з *Ethereum*-мережею;
- *Web3.php* дозволяє підписуватися на події від смарт-контрактів, отримуючи сповіщення про зміни в блокчейні, пов’язані з вашими контрактами;
- керування *Ethereum*-адресами та балансами;
- бібліотека дозволяє генерувати *Ethereum*-адреси та отримувати інформацію про баланси акаунтів.

3.3.2. Розгортання *Ethereum*

Розгортання ноди *Ethereum* проводиться на операційній системі *Linux*, в нашому випадку *Linux Ubuntu* 20.04. Для того, щоб запустити вузол блокчейну *Ethereum* у себе на комп’ютері треба встановити пакет *geth* (*Go Ethereum*), для уього у термінали спочатку вводиться команда для оновлення репозиторію пакетів:

sudo apt update

Після цього наступними командами встановлюється пакет *geth* та виводиться версія модуля:

sudo apt install geth

geth version

Для запуску ноди без додаткових налаштувань можна ввести команду:

geth

Щоб повноцінно запустити блокчейн у себе на комп'ютері треба спершу створити директорію для даних, це робиться такими термінальними командами:

mkdir ~/ethereum-node

cd ~/ethereum-node

Слід за цим треба ініціалізувати *Ethereum*-ноду, тут вказується шлях до файлу генезису, який створюється при завантаженні пакету *geth*:

geth --datadir ~/ethereum-node init /path/to/genesis.json

Після цих усіх кроків можна запустити блокчейн-вузол, наступна команда створює *Ethereum*-ноду з відкритим *RPC*-з'єднанням:

geth --datadir ~/ethereum-node --rpc --rpcaddr "127.0.0.1" --rpcport "8545" --rpcapi "eth,net,web3" –nodiscover

Опції команди:

1) --*datadir*: Шлях до директорії з даними ноди;
2) --*rpc*: Вмикає *RPC*-сервер для зовнішніх з'єднань;
3) --*rpcaddr*: *IP*-адреса, яку слід слухати (в нашому випадку 127.0.0.1);
4) --*rpcport*: Порт, який слід слухати (в нашому випадку 8545);
5) --*rpcapi*: Список *API*, доступних через *RPC* (в нашому випадку "*eth,net,web3*");
6) --*nodiscover*: Відключає автоматичне виявлення вузлів.

Ethereum-нода почне синхронізацію з мережею *Ethereum*. Цей процес може зайняти тривалий час. Після закінчення синхронізації з вузлом можна взаємодіяти за допомогою бібліотеки *Web3.php*, яка була описана раніше.

3.3.3. Смарт-контракти для *Ethereum*

Створення смарт-контрактів у мережі *Ethereum* – це процес написання та розгортання програмного коду, який визначає правила і логіку взаємодії між учасниками мережі. Смарт-контракти є автономними програмами, які працюють на

блокчейні та автоматично виконують визначені умови при надходженні певних подій або транзакцій. Основний мова для написання смарт-контрактів у мережі *Ethereum* – це *Solidity*.

Для написання смарт-контракту перш за все потрібно встановити середовище розробки, яке включає в себе *Ethereum*-клієнта (наприклад, *Geth* або *Besu*), редактор коду, у нашому випадку *Remix* (рис. 3.7), та інші необхідні інструменти.

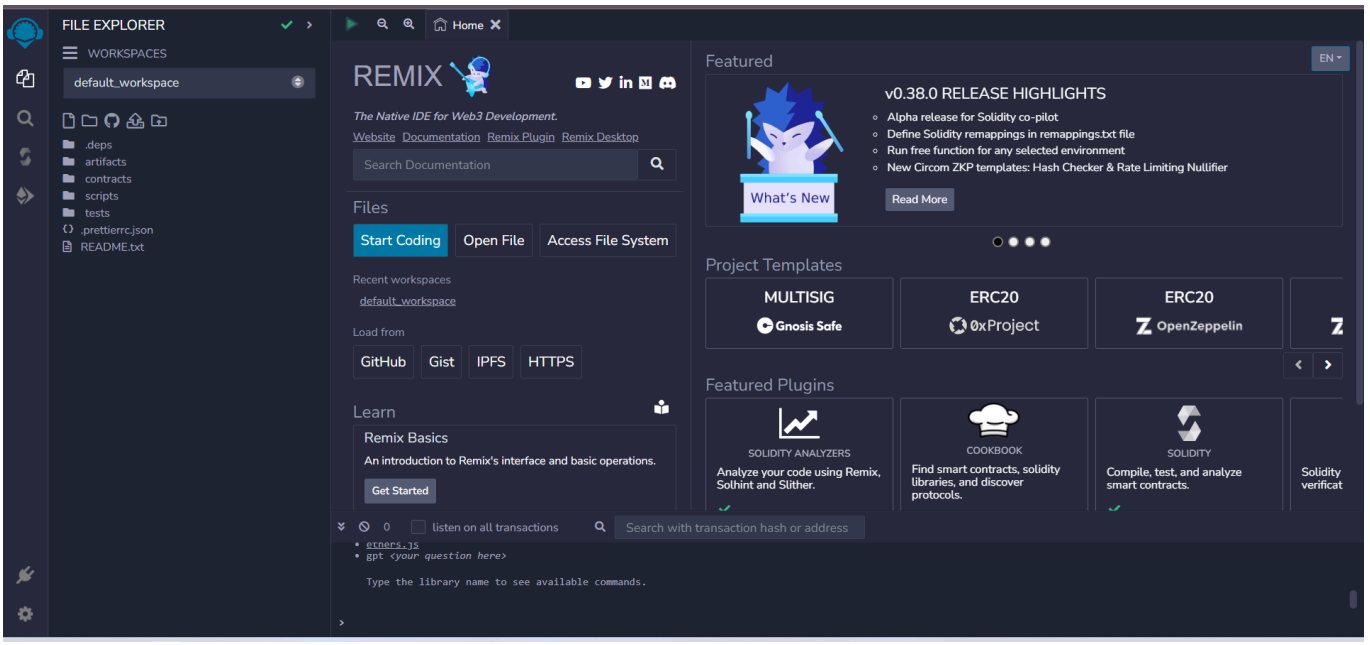

Рис. 3.7. *IDE* для написання смарт-контрактів на мові *Solidity*

Після налаштування середовища слід визначитися, що саме повинен робити смарт-контракт. Наприклад, зберігати та зчитувати інформацію в блокчейні, порівняння даних, які передаються у смарт-контракт з даними, що знаходяться у блокчейні, ведення обліку тощо. Потрібно визначитися з його функціями, змінними та логікою виконання. Нижче представлений код смарт-контракту, який має можливість зберігати рядок, повертати його значення та порівнювати з іншим рядком:

```
pragma solidity ^0.8.0;

contract StringStorage {
    string private storedString;
```

```
    // Функція для зберігання рядка
    function storeString(string memory _newString) public {
        storedString = _newString;
    }

    // Функція для отримання збереженого рядка
    function getString() public view returns (string memory) {
        return storedString;
    }

    // Функція для порівняння рядка з збереженим рядком
    function compareString(string memory _compareString) public view returns
(bool) {
        return (keccak256(abi.encodePacked(storedString))==
keccak256(abi.encodePacked(_compareString)));
    }
}
```

Основні елементи цього смарт-контракту:

- *storedString* – змінна, що зберігає рядок в блокчейні.
- *storeString* – функція для зберігання нового рядка у змінній *storedString*.
- *getString* – функція для отримання поточного збереженого рядка.
- *compareString* – функція для порівняння переданого рядка з збереженим рядком.

Для того, щоб смарт-контракт виконував якісь функції, його треба скомпілювати та розгорнути в блокчейні. *IDE Remix* на борту має вбудований компілятор. Компіляція та розгортання тут проходить автоматично.

Для тестування смарт-контрактів, існує кілька тестових мереж блокчейну *Ethereum*, серед яких можна виділити *Ropsten*, *Kovan*, *Goerli*, *Rinkeby* та *Ethereum Hardhat Network*.

Тестування смарт-контрактів під час розроби системи було проведено на тестовій мережі *Ropsten*. Тестування на тестовій мережі відбувається так само, як і робота на звичайній мережі, за винятком того, що за виконання транзакцій не буде зніматися комісія.

3.4. Висновки до розділу

У цьому розділі було розглянуто розроблювану систему електронного документообігу з використанням технології блокчейн. Система складається з двох основних частин – клієнтської та серверної які взаємодіють між собою за допомогою протоколу *HTTP*. Визначено вимоги до апаратної системи серверу та вимоги до програмних засобів для коректної та ефективної роботи системи.

Клієнтська частина програми побудована за допомогою бібліотеки шаблонів *Bootstrap*, які в собі містить готові варіанти різних блоків, та сітку для створення структури сторінки. За інтерактивність та покращену взаємодію з користувачем відповідає мова програмування *JavaScript* та бібліотека *jQuery*.

Серверна частина написана мовою програмування *PHP*, яка в свою чергу використовує кілька шаблонів проектування для побудови простої та ефективної архітектури серед яких *MVC*, Active Record та *ORM*. За обробку запитів від клієнта відповідають контролери – класи мови *PHP*, які за шаблоном *MVC* містять у собі основну бізнес-логіку системи, оперують даними, які отримують за допомогою моделей. Модель – клас, який містить у собі всю логіку роботи з базами даних. Моделі реалізують шаблон проектування *Active Record*, для зручного та швидкого створення запитів до бази даних. Крім того, моделі реалізують *ORM*, основна логіка якого полягає в тому, що одна модель відповідає одній таблиці в базі даних, об’єкт такого класу відповідає за один запис у відповідній таблиці, а властивості класу – це кожен пункт відповідного запису в таблиці.

База даних базується на системі управління базами даних *MySQL*, розглянуто таблиці та зв’язки між ними і виявлено, що у базі даних для системи електронного

документообігу використовується тільки один вид зв’язу, а саме один до багатьох (1:Б). Розглянуто та описано таблиці, які використовуються системі.

Програма, написана на мові програмування *PHP* може взаємодіяти з блокчейном Ethereum та смарт-контрактами до нього за допомогою бібліотеки *Web3.php*. Це бібліотека для мови програмування *PHP*, призначена для взаємодії з Ethereum блокчейном. Вона дозволяє створювати *PHP*-додатки, які можуть виконувати різні операції на *Ethereum*, такі як створення транзакцій, виклик смарт-контрактів та читання стану блокчейну.

Перед початком роботи з блокчейном, потрібно розгорнути вузол у себе на комп’ютері, найчастіше розгортання ноди *Ethereum* проводиться на операційній системі *Linux*, в нашому випадку *Linux Ubuntu* 20.04 використанням програмного пакету *geth* (*Go Ethereum*).

Створення смарт-контрактів у мережі *Ethereum* – це процес написання та розгортання програмного коду, який визначає правила і логіку взаємодії між учасниками мережі. Смарт-контракти є автономними програмами, які працюють на блокчейні та автоматично виконують визначені умови при надходженні певних подій або транзакцій. Основний мова для написання смарт-контрактів у мережі *Ethereum* – це *Solidity*. Для написання смарт-контракту перш за все потрібно встановити середовище розробки, яке включає в себе *Ethereum*-клієнта (наприклад, *Geth* або *Besu*), редактор коду, у нашому випадку *Remix*, та інші необхідні інструменти. Після налаштування середовища слід визначитися, що саме повинен робити смарт-контракт. Наприклад, зберігати та зчитувати інформацію в блокчейні, порівняння даних, які передаються у смарт-контракт з даними, що знаходяться у блокчейні, ведення обліку тощо.

РОЗДІЛ 4
ОГЛЯД СИСТЕМИ ЕЛЕКТРОННОГО ДОКУМЕНТООБІГУ

4.1. Робота користувача з додатком

4.1.1. Реєстрація та авторизація користувача у системі

Перед початком роботи з системою, користувачу слід отримати дані для авторизації в системі. Реєстрація користувачів проводиться менеджером, який має доступ до панелі адміністрування, тобто має права доступу адміністратора. Після успішної реєстрації генерується унікальний токен авторизації, який буде використовуватись для автентифікації цього користувача, також він заноситься у блокчейн, для перевірки дійсності токену, так як інформація в блокчейні незмінна, в той час як дані в даних можуть бути змінені, якщо зловмисник отримає до неї доступ. інтерфейс сторінки реєстрації зображено на рис. 4.1.

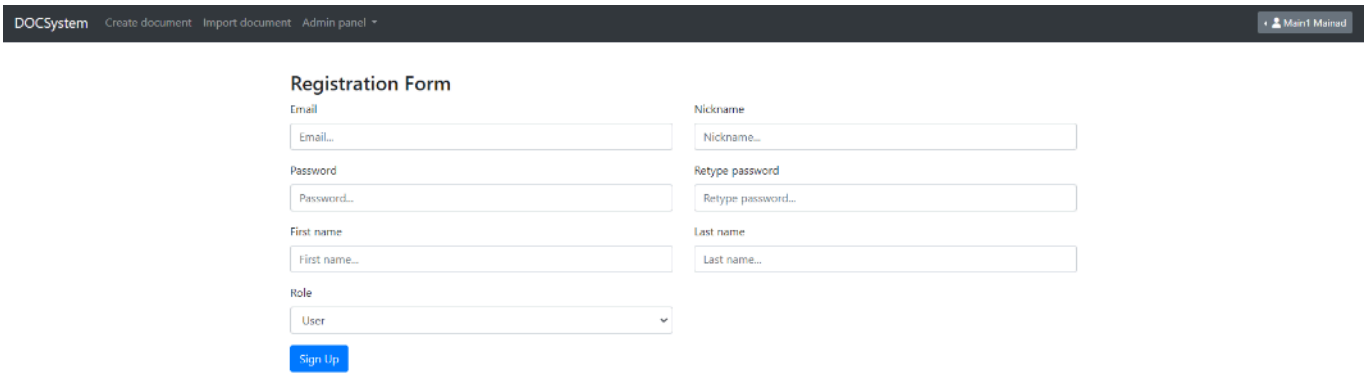

Рис. 4.1. Форма реєстрації користувача

Неавторизований користувач не має доступу до системи. Щоб авторизуватись в системі користувач вводить логін або електронну пошту, які були раніше

зареєстровані менеджером, у спеціальну форму, після чого отримує свій токен авторизації, за допомогою якого він і має доступ до всіх інших функцій системи, до яких він має доступ.

Співробітник авторизується через спеціальну форму та отримує доступ до списку документів, профілю свого облікового запису, можливість створювати, редагувати та відсилати документи іншим користувачам. Без авторизації користувач не зможе виконати жодних дій у системі. Форма авторизації зображена на рис. 4.2.

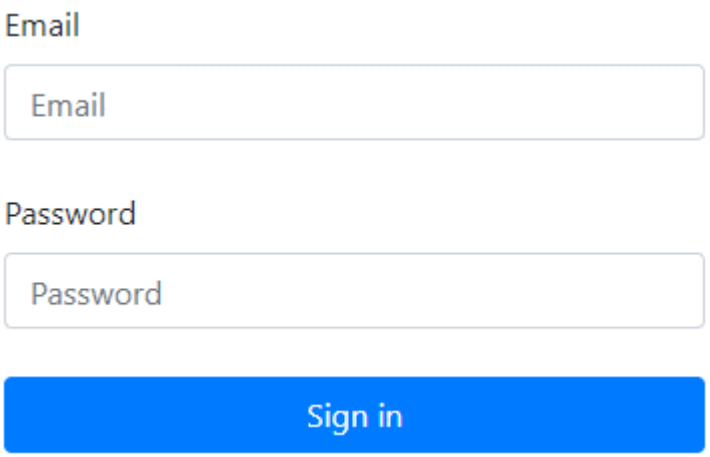

Рис. 4.2. Форма авторизації

Звичайний користувач не має можливості реєструвати нових людей, адже форма реєстрації знаходиться в панелі адміністрування, до якої мають доступ тільки адміністратори. Схема алгоритму реєстрації користувача зображено на рис. 4.3.

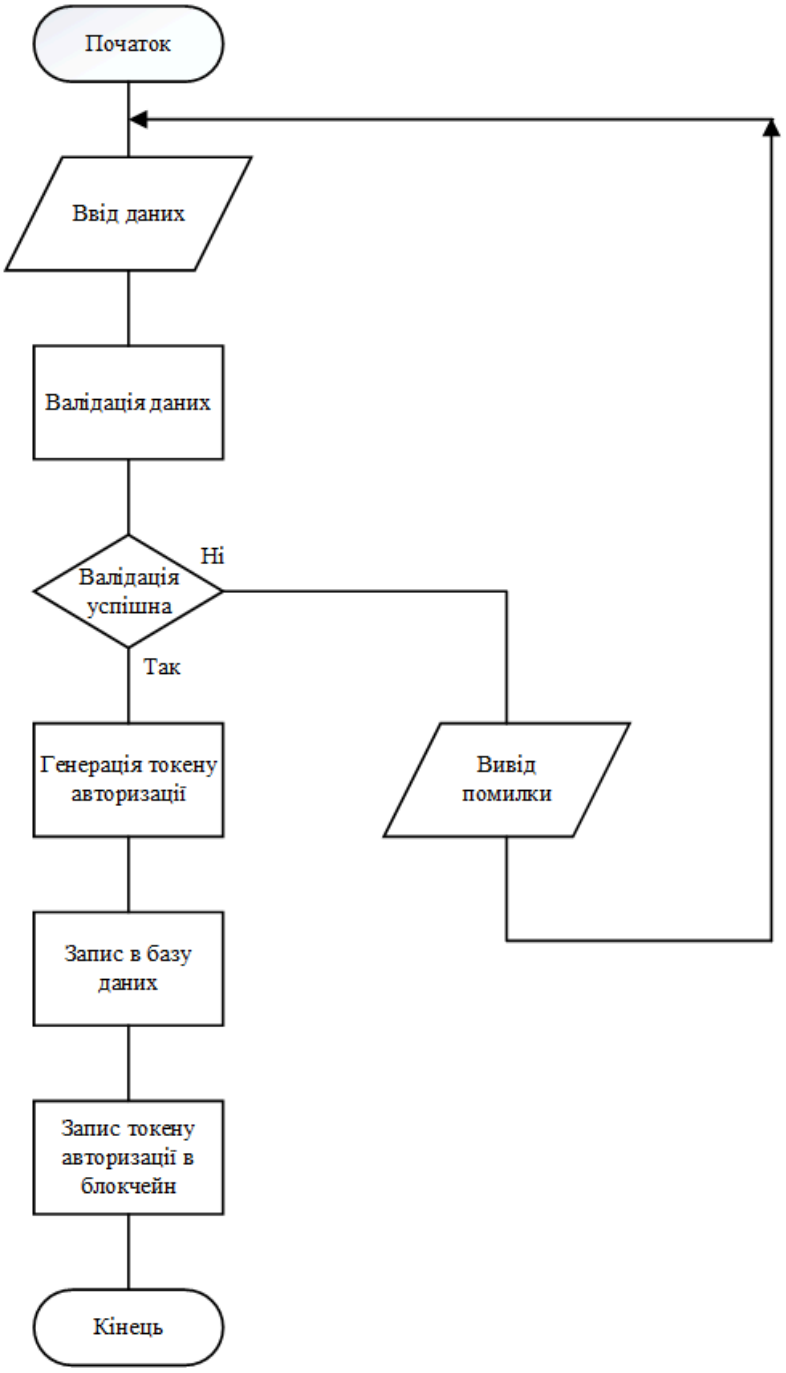

Рис. 4.3. Схема алгоритму реєстрації користувача

4.1.2. Інтерфейс користувача та список документів

Список доступних користувачу документів розміщений на головній сторінці системи (рис. 4.4) у вигляді таблиці з даними. Дані представлені у таблиці: ідентифікатор документа, назва, автор документа, дата створення, жата зміни та колонка з діями для кожного документу.

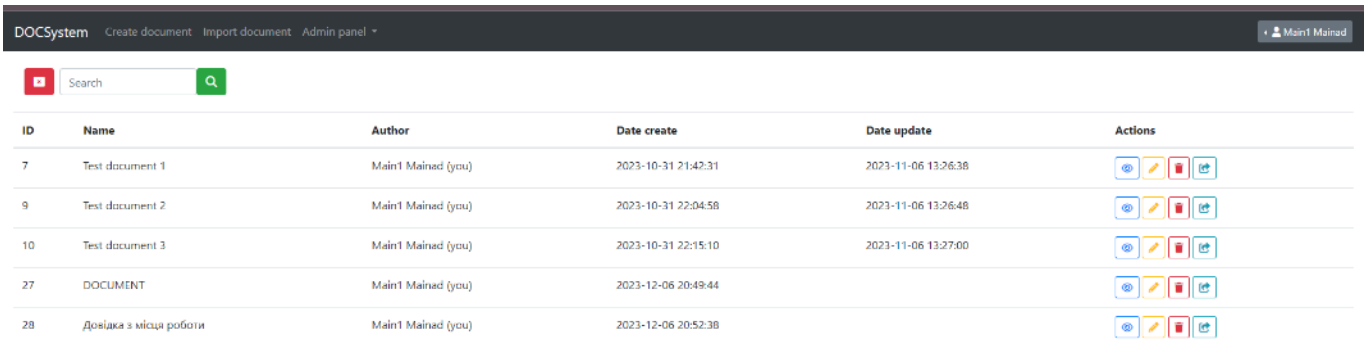

Рис. 4.4. Головний екран системи

Дії, які можна виконувати з документами:

- переглянути документ повністю – можна переглянути документ, дізнатися розгорнуту інформацію;
- редагувати документ – змінити назву документа;
- видалити документ – під час видалення документа він не видаляється повністю, а переноситься в архів, де він буде доступний до перегляду тільки адміністратору;
- відіслати документ іншому користувачу – надіслати документ користувачу, якого можна знайти за його ім’ям, електронною поштою чи ідентифікатором.

У верхній частині екрану знаходиться головне меню, в якому є пункти переходу на голову сторінку, на форму створення документа. В лівій частині знаходиться меню користувача, тут можна перейти в профіль облікового запису, відкрити налаштування профілю та вийти з системи. Меню користувача зображено на рис. 4.5.

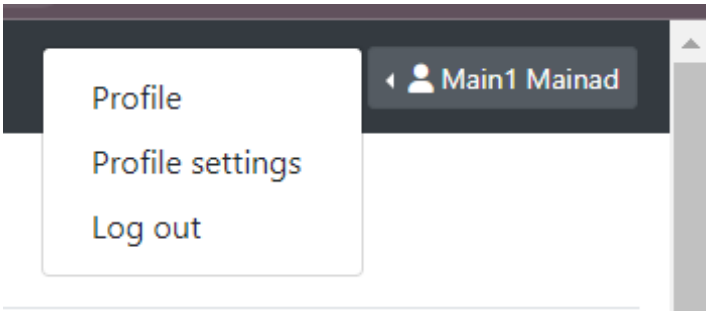

Рис. 4.5. Меню користувача

Серед налаштувань профілю є можливість змінити такі дані як ім’я користувача, його прізвище, нікнейм у системі та роль користувача, якщо профіль переглядає адміністратор. Окремою кнопкою можна перейти до зміни паролю для входу в систему. Налаштування профілю показано на рис. 4.6.

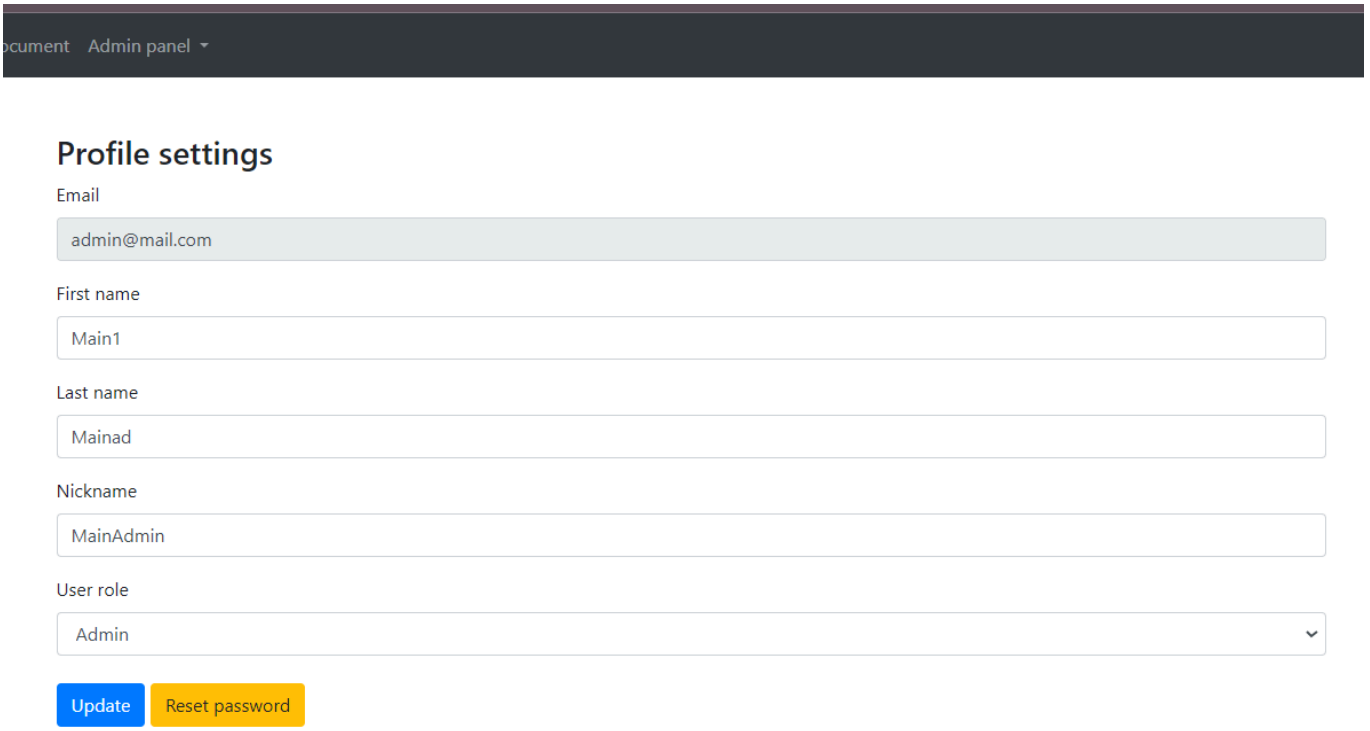

Рис. 4.6. Налаштування профілю користувача

У самому профілі користувача розміщується вся доступна інформація про нього, така як ім’я та прізвище, нікнейм, електронна пошта, його роль та дату реєстрації (рис. 4.7).

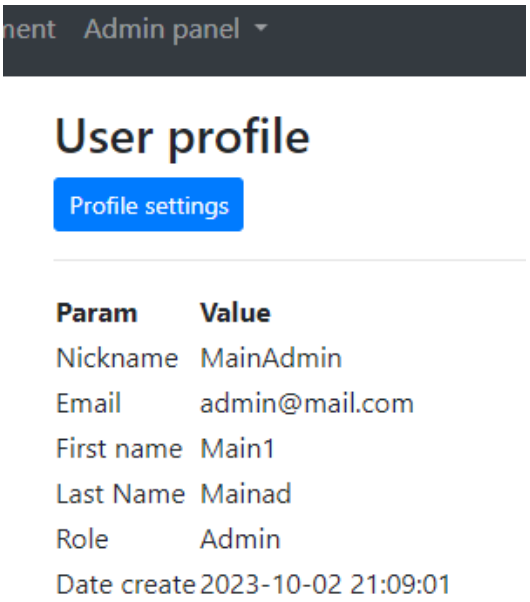

Рис. 4.7. Профіль користувача

4.1.3. Створення та перегляд документів

У програмній системі електронного документообігу користувач, обирає тип документа та вводить назву та створює сам документ, такі як назва, дата, адресат та текст. За необхідності, може бути накладений електронний підпис. Після завершення створення документа, його можна відправити відповідному отримувачу, або ж зберегти в системі для подальшого використання. Система надсилає сповіщення користувачам про нові документи та веде журнал подій для відстеження змін. Документи можуть бути автоматично або за вибором користувача архівовані для збереження історії, а також ведеться управління версіями для зручності роботи з оновленнями. Сторінка творення документа зображена на рис. 4.8.

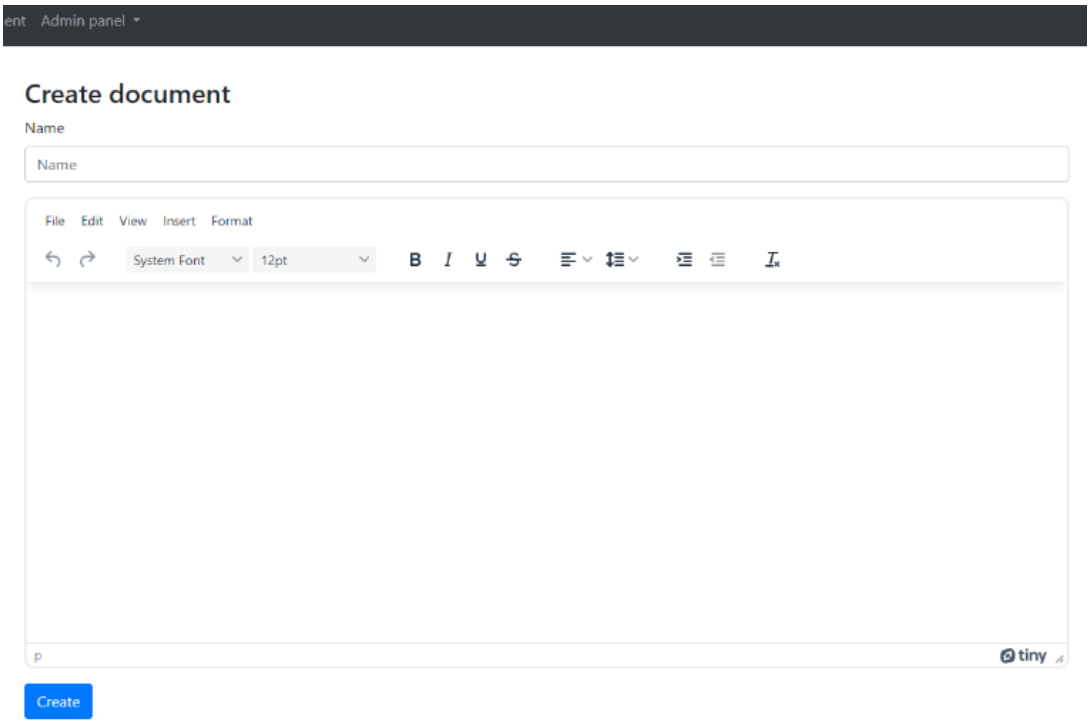

Рис. 4.8. Сторінка створення документа

Створення документа проходить у кілька етапів. Спочатку користувач вводить назву документа та його текст, натискає кнопку створити. Наступним кроком йде валідація, перевірка на правильність вводу полів, якщо перевірка не успішна, користувачу виводиться відповідне повідомлення. Після чого, текст форматується так як текстовий редактор *TinyMCE*, за допомогою якого і створюється текстовий документ, повертає поле з *HTML*-тегами, цей текст треба перевірити на ін’єкції коду, які може ввести зловмисник для отримання контролю над системою чи для крадіжки даних. Після успішного форматування документ записується в базу даних, паралельно цьому створюється його хеш та зберігається у блокчейні за допомогою смарт контракту для подальшої перевірки на дійсність. Схема алгоритму створення документа зображено на рис. 4.9.

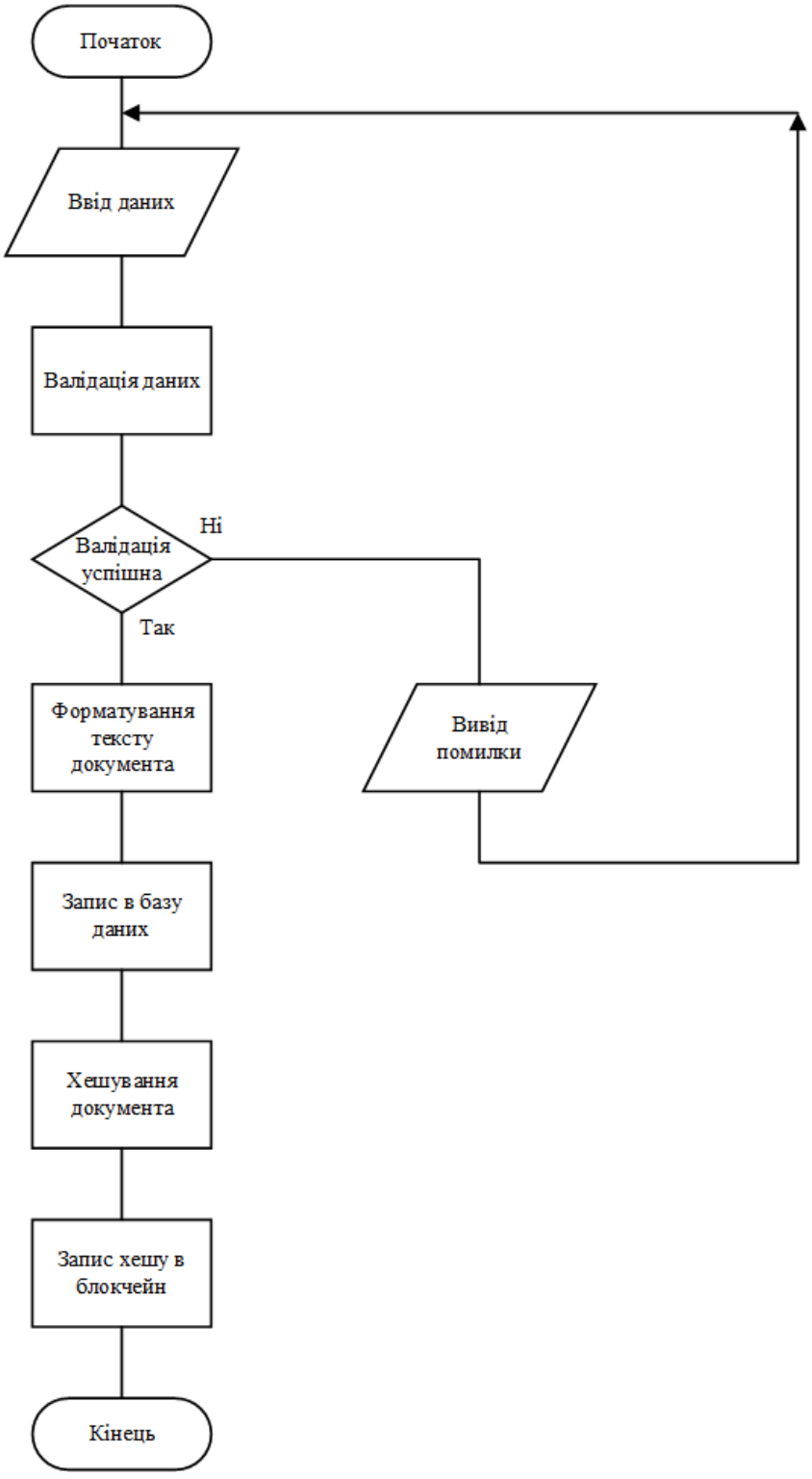

Рис. 4.9. Схема алгоритму створення документа

Після створення, захешована копія документа знаходиться в блокчейні, через що можна перевіряти його на дійсність, для цього у смарт-контракті є окрема функція, яка хешує текст документа, який в неї передається із системи та порівнює із записаними даними у блокчейн. В деяких випадках зберігати хешовані дані буде

кращим виходом, адже функція хешування видає завжди сталої довжини вихід незалежно від довжини входу через це дані можуть займати значно менше простору але за умови, що їх не треба буде використовувати десь, окрім як порівняння з іншими даними, адже хешування одностороннє.

Окрім створення документа в самій системі, є можливість імпортувати документ з комп'ютера. Для цього в головному меню є відповідний пункт. Вікно імпорту документа зображено на рис. 4.10.

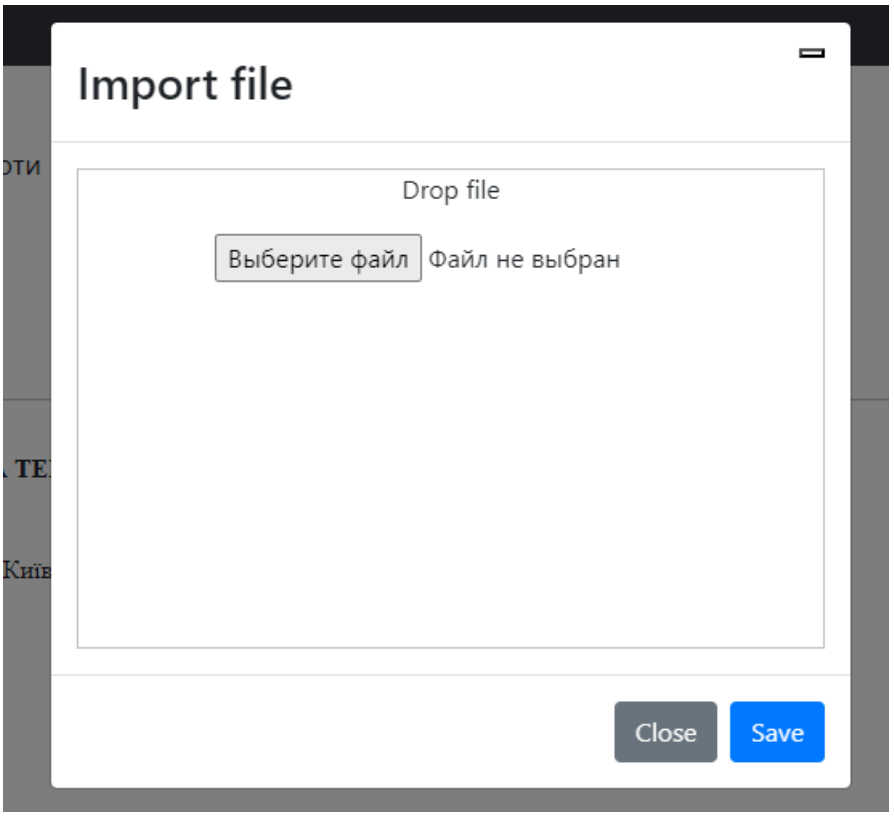

Рис. 4.10. Вікно імпорту документа в систему

Система приймає файли з розширенням *PDF* та *DOCX*. Алгоритм створення таким способом ідентичний до того, якби його створювали за допомогою інструментів системи за винятком того, що текст імпортованого файлу стає основною частиною, а назва файлу – назвою зареєстрованого документа.

4.1.4. Редагування, передача та видалення документів

Редагування документа можна зробити двома пособами, змінити його назву та змінити вміст документа. Перейменування це найпростіший спосіб зміни документа,

адже після цього не потрібно змінювати та записувати дані в блокчейн. Для перейменування на головній сторінці, де список документів, є спеціальна кнопка жовтого кольору та з відображенням олівця, при натисканні на яку з'являється модальне вікно в якому знаходиться відповідна форма для зміни імені (рис. 4.11).

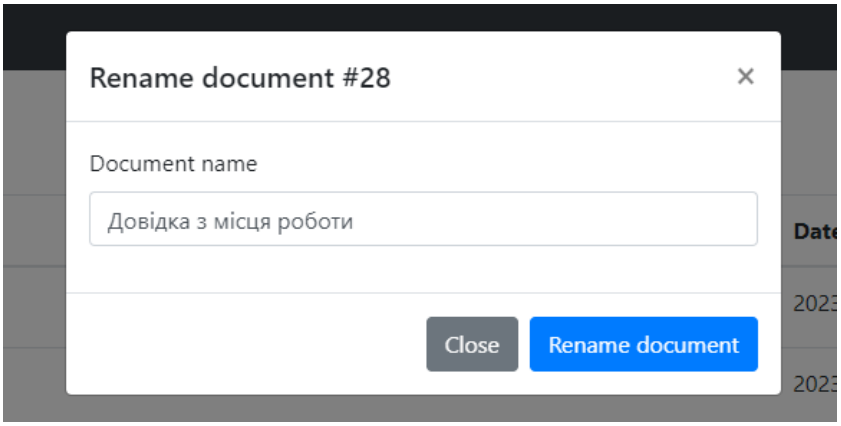

Рис. 4.11. Форма перейменування документа

Для видалення також є спеціальна кнопка з підтвердженням дії, для того щоб користувач ненароком не видалив важливий документ. Документ не зникає з системи, йому виставляється час видалення та переноситься в спеціальний архів, з якого він буде недоступний для перегляду звичайним користувачам, але користувачі з правами доступу адміністратора зможуть взаємодіяти з ним. В архів документів можна перейти тільки з адміністративної панелі системи, через що він не доступне звичайним користувачам.

Відправка документа іншим користувачам працює за принципом копіювання зі зміною власника документа. Для надсилання документа в на головній сторінці є окрема кнопка зеленого кольору, при натисканні на яку викликається відповідне модальне вікно. Знайти користувача системи можна за ідентифікатором його облікового запису, за електронною поштою, за нікнеймом чи іменем користувача. Під час надсилання документа створюється його копія, в базу даних записується копія, власником якої є користувач, якому була відправка. Модальне вікно відправки документа користувачу зображено на рис. 4.12.

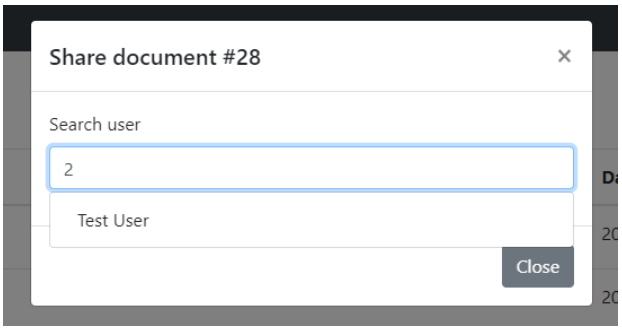

Рис. 4.12. Вікно відправки документа

4.1.5. Перегляд документа

Переглянути документ можна натиснувши на відповідну кнопку з відображенням ока в списку документів, після чого відкриється сторінка із основною інформацією про документ та його вмістом (рис. 4.13).

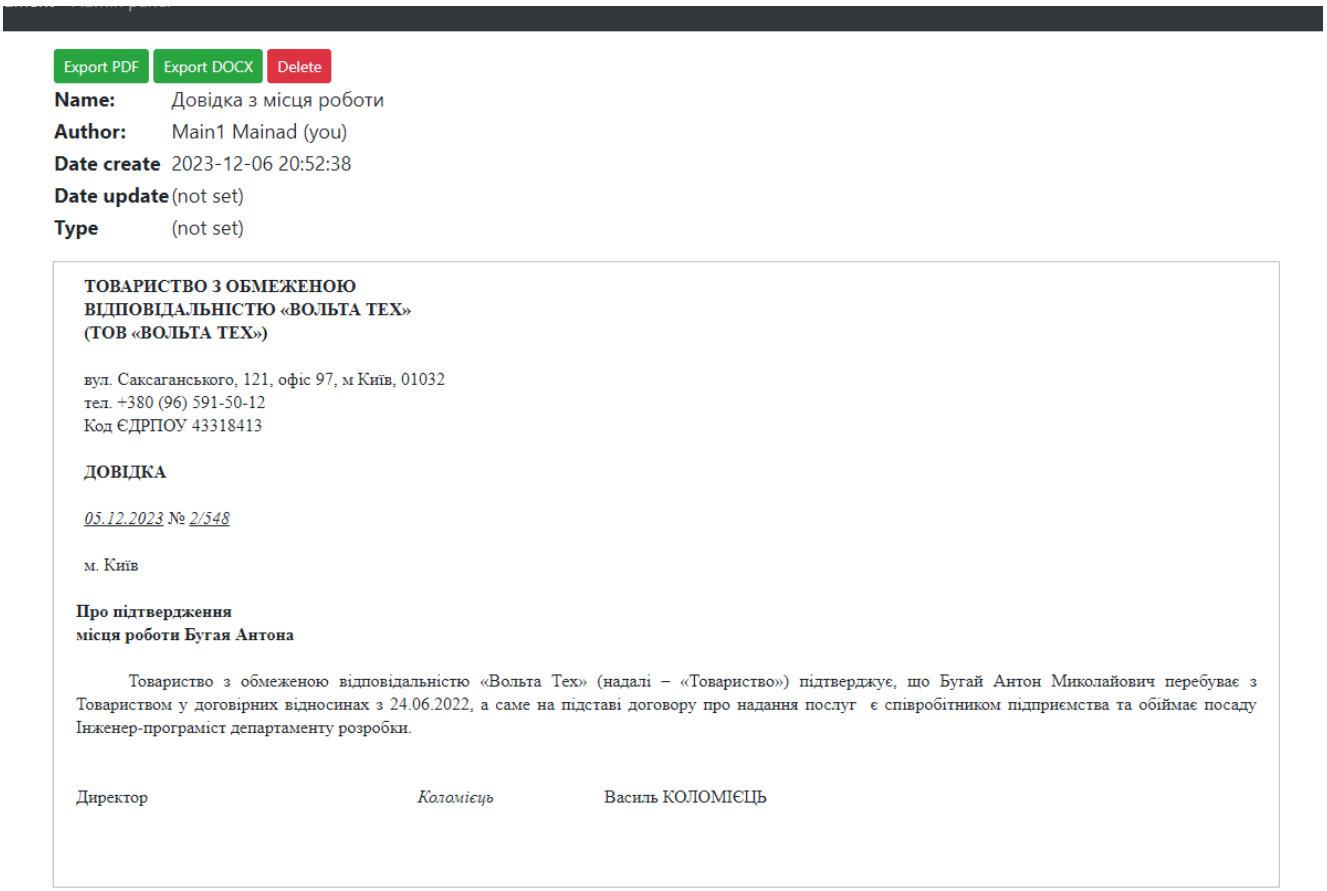

Рис. 4.13. Сторінка огляду документа

З цієї сторінки можна виконати такі дії:

– видалити документ;

- експортувати в файл з розширенням *PDF* (створюється файл з відповідним змістом та завантажується на комп’ютер користувача);
- експортувати в файл з розширенням *DOCX*.

Серед інших параметрів, які тут можна побачити є назва документа, його автор, дата створення, дата останнього оновлення та тип.

4.2. Адміністрування системи

4.2.1. Права доступу

В розробленій системі електронного документообігу є кілька видів ролей користувачів, за якими відрізняються їх права до різних функцій системи.

Користувач (*user*):

- перегляд, створення, перейменування та видалення своїх документів;
- перегляд та редагування свого облікового запису;
- відправка документа іншому користувачеві;
- завантаження документа на комп’ютер.

Адміністратор (*admin*):

- перегляд, створення, перейменування та видалення всіх документів;
- перегляд, редагування та видалення всіх облікових записів;
- відправка документа іншому користувачеві;
- завантаження документа на комп’ютер;
- доступ до адміністративної панелі;
- доступ до архіву документів;
- управління налаштуваннями системи;
- моніторинг діяльності.

4.2.2. Адміністративна панель системи

Адміністративна панель (або адмін-панель) у системі електронного документообігу з використанням технології блокчейн є централізованим інтерфейсом управління, призначеним для адміністраторів системи. Вона забезпечує потужні інструменти для ефективного контролю та моніторингу функціонування системи.

Щоб перейти до адміністративної панелі системи треба натиснути на відповідний пункт у головному меню системи, після цього з'явиться випадаючий список з розділами адміністративної панелі. Розділи адміністративної панелі:

1) користувачі (*users list*) – список усіх користувачів системи (рис. 4.14). Звідси можна переглянути профіль кожного, відредагувати, та переглянути документи, які належать цьому користувачу;

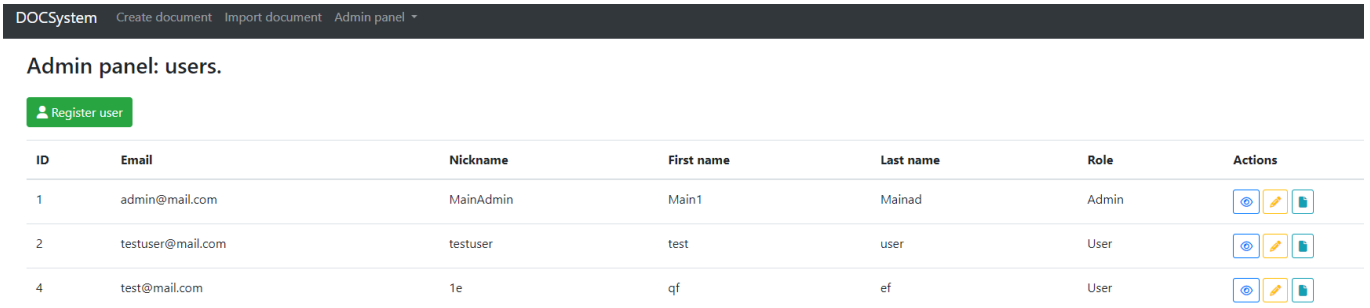

Рис. 4.14. Список користувачів системи

2) сторінка реєстрації користувачів (*register user*) (рис. 4.15);

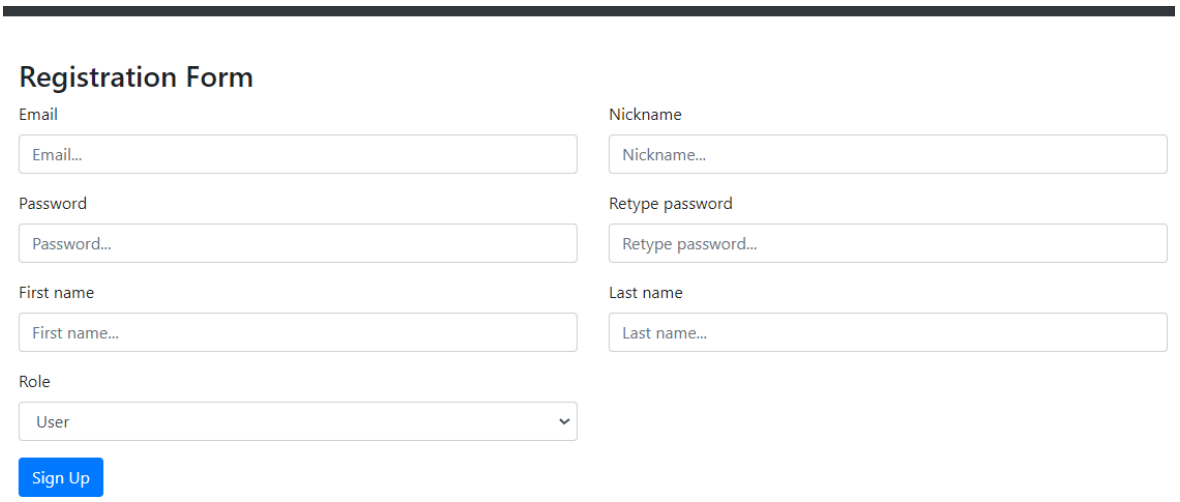

Рис. 4.15. Сторінка реєстрації користувача

3) архів документів (*documents*). Тут знаходиться список всіх документів у системі, навіть ті, які вже видалені. Звідси можна взаємодіяти з документами та переглядати їх (рис. 4.16).

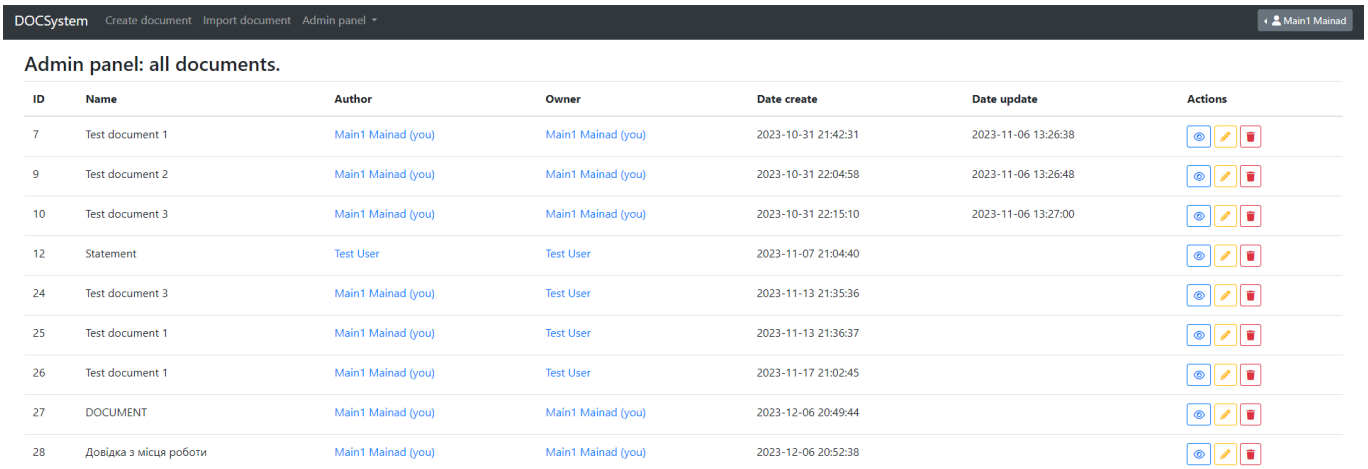

Рис. 4.16. Список всіх документів системи

Адміністративна панель відіграє ключову роль у забезпеченні ефективності та узгодженості роботи системи електронного документообігу з використанням технології блокчейн. Вона надає адміністраторам інструменти для керування, забезпечуючи її стабільне та надійне функціонування.

4.3. Звітність системи електронного документообігу

Звітність в системі електронного документообігу грає важливу роль, забезпечуючи прозорість, ефективність та можливість прийняття обґрунтованих управлінських рішень. Ось кілька ключових аспектів, пов’язаних із звітністю та аналітикою використання системи електронного документообігу:

- звітність надає інформацію про поточний статус кожного документа в системі, починаючи від створення та підписання до завершення процесу;
- аналітика дозволяє відстежувати час, необхідний завершення певних етапів документообігу, що допомагає ідентифікувати вузькі місця та оптимізувати процеси;

- система веде журнал дій користувачів, надаючи адміністраторам повний огляд того, хто, коли і що зробив у системі;
- звітність про зміни в документах допомагає підтримувати безперервність та забезпечувати безпеку даних;
- звіти можуть включати інформацію про те, як часто система використовується, хто активніший за всіх учасників, і які функціональності популярні;
- адміністратори можуть аналізувати, хто має доступ до яких документів, забезпечуючи відповідність політиці безпеки.

Загалом, система електронного документообігу, забезпечена функціональністю звітності та аналітики, стає потужним інструментом для оптимізації бізнес-процесів, покращення безпеки та підвищення ефективності організації.

4.4. Висновки до розділу

Розроблена система враховує потреби користувачів і адміністраторів, надаючи їм інтуїтивно зрозумілий і зручний інтерфейс, який значно спрощує процес створення, редагування, передачі та моніторингу документів.

Адміністраторам система надає інструменти управління та моніторингу, що дозволяє оперативно реагувати на зміни в обігу документів та вдосконалювати систему на основі детальної аналітики і звітності. Гнучкість системи є ключовими факторами, забезпечуючи її адаптованість до змін у потребах користувачів та розвитку технологій.

Також розроблювана система показала результат підвищення безпеки обробки та зберігання документації в системі за рахунок збереження зашифрований документів у блокчейні, що дало змогу порівнювати їх із документами, які знаходяться в системі, тим самим перевіряти їх непідробність. Крім того зберігання токенів авторизації у блокчейні ускладнює злам облікових записів та отримання доступу до функцій системи.

Звітність документообігу має важливу роль у системі електронного документообігу. За допомогою цього функціоналу забезпечується прозорість системи, ефективність та можливість прийняття обґрунтованих управлінських рішень. Звітність надає інформацію про поточний статус кожного документа в системі, починаючи від створення та підписання до завершення процесу; аналітика дозволяє відстежувати час, необхідний завершення певних етапів документообігу, що допомагає ідентифікувати вузькі місця та оптимізувати процеси; звітність про зміни в документах допомагає підтримувати безперервність та забезпечувати безпеку даних; звіти можуть включати інформацію про те, як часто система використовується, хто активніший за всіх учасників, і які функції найпопулярніші; адміністратори можуть аналізувати, хто має доступ до яких документів, забезпечуючи відповідність політиці безпеки.

У розробленій системі користувачі розділяються за своїми ролями. За допомогою ролей реалізовано розділення користувачів та адміністраторів. Користувачі та адміністратори відрізняються доступом до різних функцій системи, адже адміністратор має доступ до адміністративної панелі, в якій він може переглянути архів документів, список користувачів та інформацію про кожного, має доступ до функціоналу реєстрації нових користувачів, а також до звітності системи.

У підсумку, аналіз показав, що розроблена система електронного документообігу відповідає запитам користувачів та адміністраторів, однак для її підтримки та покращення необхідне постійне вдосконалення системи безпеки та взаємодії з користувачами. Система електронного документообігу високу стабільність та відмовостійкість, що гарантує стабільність та безперервність роботи системи. Ключові показники ефективності включають зниження часу на пошук, створення та відправку документа, зменшення помилок у резерваціях та підвищення продуктивності користувачів завдяки інтуїтивно зрозумілому інтерфейсу і простоті використання. За допомогою аналітичних інструментів адміністратори можуть відслідковувати зміни документообігу та моніторити користувачів системи.

ВИСНОВКИ

У світі невпинних технологічних трансформацій та стрімкого розвитку інформаційного суспільства, питання ефективного управління документами стає не тільки актуальним, але й вимагає відповіді від сучасних технологічних рішень. У даній магістерській дипломній роботі ми зосередилися на розробці програмної системи електронного документообігу, яка має не тільки відповісти на ці виклики, але й визначити новий стандарт для безпеки та ефективності у цій сфері.

В ході дослідження і розробки системи ми звернули увагу на технологію блокчейн як ключовий інструмент для вирішення проблем, пов’язаних із захистом та довірою до електронних документів. Технологія блокчейн виявилася не лише потужним засобом криптографічного захисту даних, але й сприяла автоматизації процесів та забезпечила невід’ємний внесок у підвищення ефективності управління документами.

Було проведено аналіз сучасних проблем та викликів, що стосуються електронного документообігу в організаціях. На основі цього аналізу було обрано та обгрунтовано використання технології блокчейн для створення безпечної та ефективної програмної системи електронного документообігу.

У першому розділі було проведено аналіз предметної області, досліджено технологію блокчейн, виявлено її особливості, переваги та недоліки. Ця технологія впроваджує низку інноваційних механізмів, які роблять систему електронного документообігу надійною та стійкою до різних видів загроз. Тут використовуються потужні методи шифрування та захисту інформації в кожному блоці даних. Це забезпечує конфіденційність інформації і ускладнює доступ до неї неправомірним користувачам. Використання криптографічних хеш-функцій дозволяє перевірити цілісність даних, а будь-які зміни в існуючих блоках стають митно помітними.

Блокчейн працює на принципі розподіленої мережі, де жодна конкретна точка не є центром контролю. Це ускладнює завдання зловмисників, оскільки для атаки на систему потрібно взяти під контроль більше половини вузлів мережі. Коли блок

додається до ланцюга, він стає частиною безперечного запису, який ще називають "ланцюгом блоків". Це забезпечує імутабельність даних, тобто неможливість зміни чи вилучення інформації вже внесеної в блокчейн. Цей принцип забезпечує стійкість до внутрішніх атак та підвищує надійність документів.

Крім того було розглянуто кілька існуючих блокчейн платформ, досліджено їх основні аспекти, переваги, недоліки та відмінності. Серед цих технології *Ethereum* та *Cardano*.

У другому розділі було проведено огляд технологій розробки, з урахуванням їхнього потенціалу для інтеграції з блокчейн. Виявлено, що сучасні інструменти розробки надають зручність та ефективність у створенні веб-систем. Було досліджено клієнт-серверну архітектуру. Клієнт-серверна архітектура поділяється на кілька видів, а саме дворівнеру, трирівневу та багаторівневу.

Дворівнева клієнт-серверна архітектура складається з двох вузлів – сервер, який відповідає за отримання запитів і відправку відповідей клієнту, використовуючи при цьому лише власні ресурси; клієнт, який представляє користувацький інтерфейс. Трирівнева архітектура складається з трьох компонентів: представлення даних – призначений для користувача інтерфейс; прикладний компонент – сервер додатків; керування ресурсами – сервер бази даних, який надає інформацію. Трирівневу архітектуру можна розширити до багаторівневої шляхом додавання додаткових серверів. Багаторівнева архітектура дозволяє підвищити ефективність інформаційної системи, а також оптимізувати розподіл її апаратних і програмних ресурсів. Крім того, досліджено основні аспекти клієнтської та серверної частини системи та розглянуто технології, які використовуються на кожному з рівнів.

У третьому розділі була розглянута програмна реалізація системи електронного документообігу, використовуючи технологію блокчейн. Розроблено програму, яка враховує всі вимоги до безпеки, імутабельності та автоматизації процесів, що забезпечують ефективність функціонування. Сформульовано системні та програмні вимоги до середовища виконання системи. Розглянуто структуру системи та технології, які формують архітектуру програми.

Четвертий розділ присвячений огляду системи електронного документообігу, де було висвітлено результати розробки, розглянуто основні функції системи, як вона взаємодіє з користувачем. Розглянуто алгоритми реєстрації користувачів та створення документів. Особливість алгоритму реєстрації користувачів полягає в тому, що токени авторизації, за допомогою яких система ідентифікує користувача, зберігаються в блокчейні, тим самим захищені від крадіжки та отримання несанкціонованого доступу до системи у випадку отримання доступу до інформації в базі даних. Алгоритм створення документів відрізняється тим, що після успішної перевірки на правильність отриманих від користувача даних текст документу форматується, перевіряється на ін’єкції коду щоб уникнути атак типу *XSS* (*Cross-Site Scripting* – "міжсайтовий скриптинг"). Після цього основна частина хешується та записується в блокчейн, що дозволяє перевірити документ на дійсність, для цього інший документ також хешується тим самим алгоритмом та порівнюється з тим, що знаходиться в блокчейні.

Розробка програмної системи електронного документообігу з використанням блокчейну є актуальним та перспективним кроком у напрямку забезпечення високого рівня безпеки інформації та обміну документами. Блокчейн дозволяє створити децентралізований реєстр, в якому інформація є недоступною для змін, що значно зменшує ризик підміни. Документ отримує свій унікальний цифровий підпис, що виключає можливість підробки та забезпечує впевненість у його автентичності.

Застосування технології блокчейн у програмній системі електронного документообігу є новаторським кроком до забезпечення підвищеного рівня безпеки та довіри. Розроблювана система забезпечує високий рівень шифрування даних, отже їх конфіденційність. Такий підхід є ключовим у побудові надійної інфраструктури електронного документообігу в умовах сьогоднішнього цифрового середовища.

В ході виконання кваліфікаційної роботи було досягнуто прогресу у вирішенні актуальних завдань сучасного електронного документообігу. Застосування технології блокчейн дозволило підвищити рівень безпеки, довіри та швидкості обігу документів.

Перспективи подальшого розвитку включають удосконалення алгоритмів та розширення функціональності системи, зокрема, розгляд можливості інтеграції з іншими системами та вдосконалення механізмів забезпечення конфіденційності даних.

Розроблена програмна система електронного документообігу з використанням технології блокчейн успішно виконує свої завдання з підвищення безпеки та ефективності обробки електронних документів, що відкриває шлях для її впровадження в реальних умовах та внесення значущого внеску в розвиток сучасних систем електронного документообігу.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ПОЛОЖЕННЯ ПРО ДИПЛОМНІ РОБОТИ (ПРОЕКТИ) ВИПУСКНИКІВ НАЦІОНАЛЬНОГО АВІАЦІЙНОГО УНІВЕРСИТЕТУ. Київ: НАУ, 2017.
2. ДОКУМЕНТАЦІЯ. ЗВІТИ У СФЕРІ НАУКИ І ТЕХНІКИ. Структура і правила оформлення. ДСТУ 3008-95. Київ.
3. *Binance academy* [електронний ресурс].- Режим доступу: *https://academy.binance.com/uk/articles* (дата звернення 25.10.2023). - назва з екрана.
4. *Aws Amazon* [електронний ресурс].- Режим доступу: *https://aws.amazon.com* (дата звернення 26.10.2023).- назва з екрана.
5. *Coinbase* [електронний ресурс].- Режим доступу: *https://www.coinbase.com* (дата звернення 26.10.2023).- назва з екрана.
6. *Investopedia* [електронний ресурс].- Режим доступу: *https://wikipedia.org/wiki/* (дата звернення 27.10.2023). - назва з екрана.
7. *QATestLab* [електронний ресурс].- Режим доступу: *https://training.qatestlab.com/blog/technical-articles* (дата звернення 27.10.2023). - назва з екрана.
8. *Itstep* [електронний ресурс].- Режим доступу: *https://cloud.itstep.org/blog_3* (дата звернення 27.10.2023). - назва з екрана.
9. *Google cloud* [електронний ресурс].- Режим доступу: *https://cloud.google.com/learn/what-is-a-relational-database* (дата звернення 28.10.2023). - назва з екрана.
10. *Rdb* [електронний ресурс].- Режим доступу: *https://rdb.dp.ua/uk/chapter_02* (дата звернення 28.10.2023).
11. *Microsoft* [електронний ресурс].- Режим доступу: *https://learn.microsoft.com/en-us/azure/architecture/data-guide/big-data/non-relational-data* (дата звернення 28.10.2023).

12. ExpressSoft [електронний ресурс].- Режим доступу: *https://expresssoft.com.ua/uk* (дата звернення 28.10.2023). - назва з екрана.

13. Мего-інфо [електронний ресурс].- Режим доступу: *http://mego.info/* (дата звернення 29.10.2023). - назва з екрана.

14. Аносов А. А. Вибір СУБД для побудови інформаційних систем. Київ: Знання, 2012. 278 с.

15. Пушников А. Ю. Введення в системи управління базами даних: навч. посібник. Київ: Просвіта, 2013. 85 с.

16. Про захист персональних даних: Закон України від 1 червня 2010 р. № 2297-VI. Верховна Рада України. [Електронний ресурс].- режим доступу: *https://zakon.rada.gov.ua/laws/show/2297-17#Text* (дата звернення 25.10.2023). - Назва з екрана.

17. Бхаргава А. Грокаємо алгоритми. Ілюстрований посібник для програмістів. Київ: Просвіта, 2013. 256 с.

18. Synopsys [електронний ресурс].- Режим доступу: *https://www.synopsys.com/glossary* (дата звернення 29.10.2023). - назва з екрана.

19. Builtin [електронний ресурс].- Режим доступу: *https://builtin.com/blockchain* (дата звернення 29.10.2023). - назва з екрана.

20. Oracle [електронний ресурс].- Режим доступу: *https://www.oracle.com* (дата звернення 29.10.2023). - назва з екрана.

21. Вчасно ЕДО [електронний ресурс].- Режим доступу: *https://vchasno.ua/shcho-take-elektronnyi-dokumentoobig* (дата звернення 29.10.2023). - назва з екрана.

22. FossDoc [електронний ресурс].- Режим доступу: *https://fossdoc.com/elektronniy-dokumentooborot* (дата звернення 29.10.2023). - назва з екрана.

23. Softline [електронний ресурс].- Режим доступу: *https://softline.company/ua/news* (дата звернення 29.10.2023). - назва з екрана.

24. *Signy* [електронний ресурс].- Режим доступу: [електронний ресурс].- Режим доступу: *https://softline.company/ua/news/* (дата звернення 29.10.2023). - назва з екрана. (дата звернення 29.10.2023). - назва з екрана.
25. *Egera* [електронний ресурс].- Режим доступу: [електронний ресурс].- Режим доступу: *https://egera.com/uk/shcho-take-blokcheyn* (дата звернення 29.10.2023). - назва з екрана. (дата звернення 29.10.2023). - назва з екрана.
26. *Deals* [електронний ресурс].- Режим доступу: [електронний ресурс].- Режим доступу: *https://dealssign.com/blog/elektronnij-dokumentoobig-vidi-sistem-ta-yixni-funkciyi* (дата звернення 29.10.2023). - назва з екрана. (дата звернення 29.10.2023). - назва з екрана.

ДОДАТОК А

Лістинг коду клієнтської частини

```
<?php
/**
 * @var $user \app\Common\Models\User
 * @var $docs \app\Common\Models\Documents[]
 * @var $filters array
 */

use app\Front\App;

?>

<div class="container-fluid filters-group my-4">
    <div class="row">
        <div class="col-2 d-flex">
            <a href="/" class="btn btn-danger mr-2" title="Reset filters"><i class="fa-solid fa-rectangle-xmark"></i></a>
            <form class="d-flex" method="get">
                <input class="form-control me-2" name="searchFilter" type="search" placeholder="Search" value="<?= (isset($filters['searchFilter']) && !empty($filters['searchFilter'])) ? str_replace('+', ' ', $filters['searchFilter']) : null?>" aria-label="Search">
                <button class="btn btn-success" type="submit"><i class="fa-solid fa-magnifying-glass"></i></button>
            </form>
        </div>
        <div class="col-3" style="">
```

```
        </div>
    </div>
</div>

<table class="table table-hover">
    <thead>
    <tr>
        <th scope="col">ID</th>
        <th scope="col">Name</th>
        <th scope="col">Author</th>
        <th scope="col">Date create</th>
        <th scope="col">Date update</th>
        <th scope="col">Actions</th>
    </tr>
    </thead>
    <tbody>
    <?php if (!empty($docs)):?>
    <?php foreach ($docs as $doc):?>
    <?php /** @var $doc \app\Common\Models\Documents */ ?>
        <tr>
            <td><?= $doc->id ?></td>
            <td><?= $doc->title ?></td>
            <td><?= $doc->getAuthorFullName() ?></td>
            <td><?= $doc->created_at ?></td>
                <td><?= $doc->updated_at == $doc->created_at ? '' : $doc->updated_at
?></td>
            <td>
                <a class="btn btn-sm btn-outline-primary" href="/docs/view?id=<?= $doc->id
?>" title="View"><i class="fa-regular fa-eye"></i></a>
```

```
                <a class="btn btn-sm btn-outline-warning rename-btn" href="" title="Rename"
data-id="<?= $doc->id ?>" data-name="<?= $doc->title ?>" data-toggle="modal"
data-target="#renameModal"><i class="fa-solid fa-pencil"></i></a>
                                <a class="btn btn-sm btn-outline-danger delete-btn" href=""
data-toggle="modal" data-id="<?= $doc->id ?>" data-target="#deleteModal"
title="Delete"><i class="fa-solid fa-trash"></i></a>
                <a class="btn btn-sm btn-outline-info share-btn" href="" data-toggle="modal"
data-id="<?= $doc->id ?>" data-target="#shareModal" title="Share"><i
class="fa-solid fa-share-from-square"></i></a>
            </td>
        </tr>
    <?php endforeach;?>
    <?php else:?>
    <div class="alert alert-dark" role="alert">
        No documents available.
    </div>
    <?php endif;?>
    </tbody>
</table>

<div class="modal fade" id="deleteModal" tabindex="-1" role="dialog"
aria-labelledby="modalLabel" aria-hidden="true">
    <div class="modal-dialog" role="document">
        <div class="modal-content">
            <div class="modal-header">
                        <h5 class="modal-title" id="modalLabel">Delete document #<span
class="modal-doc-id"></span></h5>
                            <button type="button" class="close" data-dismiss="modal"
aria-label="Close">
                <span aria-hidden="true">&times;</span>
```

```
            </button>
        </div>
        <div class="modal-body">
            Are you sure you want to delete the document?
        </div>
        <div class="modal-footer">
                                    <button type="button" class="btn btn-secondary"
data-dismiss="modal">Close</button>
            <a href="" class="btn btn-danger modal-delete-btn">Delete document</a>
        </div>
      </div>
   </div>
</div>

<div class="modal fade" id="renameModal" tabindex="-1" role="dialog"
aria-labelledby="renameModalLabel" aria-hidden="true">
   <div class="modal-dialog" role="document">
      <div class="modal-content">
        <div class="modal-header">
            <h5 class="modal-title" id="renameModalLabel">Rename document #<span
class="modal-doc-id"></span></h5>
                            <button type="button" class="close" data-dismiss="modal"
aria-label="Close">
            <span aria-hidden="true">&times;</span>
            </button>
        </div>
        <div class="modal-body">
            <form id="rename-form" action="" method="post">
            <div class="form-group">
                <label for="name" class="form-label">Document name</label>
```

```
                    <input type="text" class="form-control" name="name" id="name"
placeholder="Document name">
                </div>
            </form>
        </div>
        <div class="modal-footer">
                                <button type="button" class="btn btn-secondary"
data-dismiss="modal">Close</button>
                        <button class="btn btn-primary modal-rename-btn">Rename
document</button>
        </div>
    </div>
  </div>
</div>

<div class="modal fade" id="shareModal" tabindex="-1" role="dialog"
aria-labelledby="shareModalLabel" aria-hidden="true">
  <div class="modal-dialog" role="document">
    <div class="modal-content">
      <div class="modal-header">
            <h5 class="modal-title" id="shareModalLabel">Share document #<span
class="modal-doc-id share"></span></h5>
                        <button type="button" class="close" data-dismiss="modal"
aria-label="Close">
            <span aria-hidden="true">&times;</span>
          </button>
        </div>
        <div class="modal-body">
          <form action="" method="post">
            <div class="form-group">
```

```
                <label for="search-user" class="form-label">Search user</label>
<!--                            <input type="text" id="search-user" class="form-control"
autocomplete="off">-->
                <div class="dropdown">
                                <input type="text" class="form-control" id="search-user"
autocomplete="off" data-toggle="dropdown">
                                <div class="dropdown-menu w-100" id="suggestions-list"
aria-labelledby="search-user">
                    </div>
                </div>
                <div id="suggestions-container">
                    <ul id="suggestions-list" class="list-group"></ul>
                </div>
              </div>
            </form>
        </div>
        <div class="modal-footer">
                                    <button type="button" class="btn btn-secondary"
data-dismiss="modal">Close</button>
        </div>
      </div>
   </di    v>
</div>

<script>
   jQuery(document).ready(function ($) {

      $.ajax({
         url: 'https://dev4-task.i-tech.space/tasks',
         method: 'get',
```

```
        contentType: 'json',
        headers: {
                                                                    token:
'1ca3a826c3b3021d5df2b5ef3d39b522346cb466358e560786b3ee9f397620c8'
        },
        success: function (res) {
            console.log('Success: ');
            res.forEach((item, index) => {
                console.log(index);
                console.log(item);
            })
        },
        error: function (res) {
            console.log('Error: '+res);
        }
    });

    $('.delete-btn').click(function (event) {
        let docId = $(this).data('id');
        $('.modal-delete-btn').attr('href', '/docs/delete?id=' + docId);
        $('.modal-doc-id').html(docId);
    });

    $('.rename-btn').click(function (event) {
        let docId = $(this).data('id');
        let docName = $(this).data('name');
        $('.modal-doc-id').html(docId);
        $('#name').val(docName);

        $('.modal-rename-btn').click(function (event) {
```

```
            event.preventDefault();
            let form = $('#rename-form');
            form.attr('action', '/docs/update?id=' + docId);
            form.submit();
        });

    });

    $('.share-btn').click(function (event) {
        event.preventDefault();
        $('.modal-doc-id').html($(this).data('id'));
    });

        var availableData = ["Apple", "Banana", "Orange", "Pineapple", "Grapes",
"Cherry"];
    $("#search-user").on("input", function () {
        var inputValue = $(this).val().toLowerCase();
        if (inputValue.length > 0) {
            $.ajax({
                url: "/users/get-users",
                method: "POST",
                data: {search: inputValue},
                success: function (response) {
                    // console.log(response);
                    updateSuggestionsList(response);
                },
                error: function (error) {
                    console.log("Error:", error);
                }
            });
```

```
        } else {
            updateSuggestionsList([]);
        }
    });

    function updateSuggestionsList(suggestions) {
        var suggestionsList = $("#suggestions-list");
        suggestionsList.empty();

        if (suggestions.length > 0) {
            suggestions.forEach(function (item) {
                        var listItem = $("<div class='dropdown-item search-user-item'
data-doc_id='' data-user_id='"+item.user_id+"'>" + item.fullName + "</div>");
                suggestionsList.append(listItem);

                $('.search-user-item').click(function (){
                    $.ajax({
                        url: "/docs/share-doc",
                        method: "GET",
                        data: {
                            user_id: $(this).data('user_id'),
                            doc_id: $('.modal-doc-id.share').text()
                        },
                        success: function (res) {
                            console.log('Success '+res);
                        },
                        error: function (res) {
                            console.log('Error '+res);
                        }
                    });
```

```
                $('#shareModal').modal('hide');
            });

            listItem.click(function () {
                $("#inputData").val(item);
                suggestionsList.empty();
            });
        });
    } else {
        var listItem = $("<div class='dropdown-item'>No suggestions</div>");
        suggestionsList.append(listItem);
    }
  }
});
</script>

<div class="container pt-3">
    <h3>Create document</h3>
    <div id="message">
    </div>
    <form action="/docs/create" method="post">
        <div class="form-group">
            <label for="doc-name">Name</label>
            <input type="text" class="form-control" id="doc-name" placeholder="Name">
        </div>
        <div class="form-group">
            <textarea name="text" id="text" cols="30" rows="30"></textarea>
        </div>
        <input type="submit" value="Create" id="create-btn" class="btn btn-primary">
    </form>
```

```
<script>
    tinymce.init({
        selector: '#text',
        height: 500,
          toolbar: 'undo redo | fontfamily fontsize | bold italic underline strikethrough | link image media table mergetags | align lineheight | tinycomments | checklist numlist bullist indent outdent | emoticons charmap | removeformat',
        tinycomments_mode: 'embedded',
        tinycomments_author: 'JetFrost',
        mergetags_list: [
            { value: 'Anton', title: 'First Name' },
            { value: 'bugaj.anton01@gmail.com', title: 'Email' },
        ],
                    ai_request: (request, respondWith) => respondWith.string(() => Promise.reject("See docs to implement AI Assistant")),
    });
    jQuery(document).ready(function($) {
        $('#create-btn').click(function (event) {
            let btn = $(this);
            btn.addClass('disabled');
            event.preventDefault();
            $.ajax({
                url: '/docs/create',
                method: 'post',
                dataType: 'json',
                data: {
                    name: $('#doc-name').val(),
                    text: tinymce.get('text').getContent()
                },
```

```
            success: function (res) {
                let status = res.status;
                if (status == 'error'){
                    btn.removeClass('disabled');
                                        $('#message').html('<div class="alert alert-danger"
role="alert">'+res.message+'</div>');
                }else if (status == 'success') {
                    location.href = '/';
                }else {
                    btn.removeClass('disabled');
                                        $('#message').html('<div class="alert alert-danger"
role="alert">Something went wrong.</div>');
                    console.log(res);
                }
            },
            error: function (res) {
                btn.removeClass('disabled');
                                        $('#message').html('<div class="alert alert-danger"
role="alert">Something went wrong.</div>');
                console.log(res);
            }
        });
    })
})

</script>

</div>
```