МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«До захисту допущено»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)
_____ червня 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавр

зі спеціальності 122 - Комп'ютерних наук,
освітньо-наукової програми «Інформатика»
на тему: «Внутрішня система документообігу Кіт Глобал»
здобувача групи ІН-06-2 Сухоноса Віктора Володимировича

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

_____ Віктор СУХОНОС
(підпис)

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

Суми – 2024

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

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

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня магістр

зі спеціальності 122 - Комп'ютерних наук, освітньо-наукової програми «Інформатика»
здобувача групи ІН-06-2 Сухоноса Віктора Володимировича

1. Тема роботи: «Внутрішня система документообігу Кіт Глобал» затверджую наказом по СумДУ від «22» квітня 2024 No 0414-VI
2. Термін здачі здобувачем кваліфікаційної роботи до 13 травня 2024 року
3. Вхідні дані до кваліфікаційної роботи
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
1) Аналіз проблеми предметної області, постановка й формування завдань дослідження. 2) Огляд технологій, що використовуються для розробки CRM-систем. 3) Розробка CRM системи для використання її в компанії. 4) Аналіз результатів.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «____» _____ 20 ___ р.

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

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз проблеми предметної області, постановка й формування завдань дослідження*	22.04.24-23.04.24	
2	*Огляд технологій, що використовуються для розробки CRM-систем.*	23.04.24-25.04.24	
3	*Розробка CRM системи для використання її в компанії*	25.04.24-02.05.24	
4	*Аналіз отриманих результатів*	02.05.24-04.05.24	
5	*Оформлення пояснювальної записки до кваліфікаційної роботи*	04.05.24-08.05.24	

Здобувач вищої освіти _____ Керівник _____

(підпис) (підпис)

АНОТАЦІЯ

Записка: 52 стор., 20 рис., 2 додатки, 31 джерел.

Обґрунтування актуальності теми роботи – Тема кваліфікаційної роботи належить до актуальних питань з управління бізнесом, оскільки в сучасному світі ефективне управління клієнтською базою стає ключовим елементом конкурентоспроможності компаній. Інтегровані системи документообігу, такі як зазначена CRM-система, відіграють стратегічну роль у підтримці цього управління, забезпечуючи конфіденційність, ефективність та якість обробки клієнтської інформації.

Об'єкт дослідження — Процес автоматизованого управління клієнтами компанії.

Мета роботи — Мета роботи полягає в розробці інтегрованої системи документообігу для оптимізації управління клієнтською базою компанії «Кіт Глобал». Ця система має на меті поліпшення ефективності та забезпечення конфіденційності обробки клієнтської інформації шляхом автоматизації процесів зберігання документації, створення угод, додавання завдань та коментування клієнтських профілів.

Методи дослідження — алгоритми для автоматизованого управління клієнтами

Результати — Розроблено інтегровану систему документообігу, спрямовану на оптимізацію управління клієнтами в компанії «Кіт Глобал». Система автоматизує процеси зберігання документації, створення угод, додавання задач та коментування клієнтських профілів. Її унікальність полягає в забезпеченні конфіденційності та ефективності обробки клієнтської інформації в межах компанії.

ІНФОРМАЦІЙНА СИСТЕМА, ДОКУМЕНТООБІГ, КІТ ГЛОБАЛ, NESTJS, REACTJS, MONGODB

ЗМІСТ

ВСТУП.....6
1. ІНФОРМАЦІЙНИЙ ОГЛЯД..... 7
1.1. Поняття і типи CRM.....7
1.2. Огляд аналогів.....10
1.3. Постановка задач..... 15
2. ВИБІР МЕТОДІВ РІШЕННЯ ЗАДАЧІ.....16
2.1. Вибір технологій.....16
2.2. Проектування системи управління клієнтами..... 19
3. РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ ВНУТРІШНЬОГО ДОКУМЕНТООБІГУ..... 21
3.1. Програмна реалізація системи документообігу.....21
3.2. Розробка основного функціоналу..... 22
3.3. Розробка графічного інтерфейсу системи.....25
ВИСНОВКИ.....36
СПИСОК ЛІТЕРАТУРИ.....36
Додаток А. Код клієнтської частини..... 40
Додаток Б. Код серверної частини..... 49

ВСТУП

Актуальність. Тема кваліфікаційної роботи належить до актуальних питань з управління бізнесом, оскільки в сучасному світі ефективне управління клієнтською базою стає ключовим елементом конкурентоспроможності компаній. Інтегровані системи документообігу, такі як зазначена CRM-система, відіграють стратегічну роль у підтримці цього управління, забезпечуючи конфіденційність, ефективність та якість обробки клієнтської інформації.

Об'єкт дослідження. Процес автоматизованого управління клієнтами компанії.

Мета роботи. Мета роботи полягає в розробці інтегрованої системи документообігу для оптимізації управління клієнтською базою компанії «Кіт Глобал». Ця система має на меті поліпшення ефективності та забезпечення конфіденційності обробки клієнтської інформації шляхом автоматизації процесів зберігання документації, створення угод, додавання завдань та коментування клієнтських профілів.

Результати. Розроблено інтегровану систему документообігу, спрямовану на оптимізацію управління клієнтами в компанії «Кіт Глобал». Система автоматизує процеси зберігання документації, створення угод, додавання задач та коментування клієнтських профілів. Її унікальність полягає в забезпеченні конфіденційності та ефективності обробки клієнтської інформації в межах компанії.

1. ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1. Поняття і типи CRM

Управління взаємовідносинами з клієнтами (CRM) як стратегія та технологія пройшло дивовижний еволюційний шлях. Перші технологічні спроби здебальшого мали розчаруючих ініціатив, але в останні роки ми побачили розвиток основоположних концепцій і додатків. Сьогодні CRM – це стратегія, набір тактик та технологія, які стали незамінними у сучасній економіці, оскільки завдяки цьому забезпечується як ефективність процесу утримання лояльності існуючих клієнтів, так і перехід у групу постійних клієнтів тих, хто звернувся до послуг компанії вперше [1].

Інтерес до управління відносинами з клієнтами (CRM) почав зростати ще на початку XXI ст. Тоді ж було зроблено висновок про те, що, незалежно від розміру організації, підприємства все ще мотивовані використовувати CRM задля більш ефективного створення та управління відносинами зі своїми клієнтами, що врешті-решт може призвести як до більшої лояльності клієнтів, так і до прибутковості. Крім того, в нинішніх умовах, внаслідок швидкого зростання Інтернету та пов’язаних з ним технологій, значно збільшилась можливість для маркетингу, що змінило спосіб управління відносинами між компаніями та їхніми клієнтами.

Проте, незважаючи на те, що нині CRM практично в усьому світі розглядається в якості дуже впливового бізнес-підходу, його загальноприйнятого поняття усе ще не існує, на чому свого часу вже наголошувалось [2]. Зокрема, CRM можна розглядати в якості корпоративного підходу до розуміння та впливу на поведінку клієнтів за допомогою значущої комунікації з метою покращення залученості клієнтів, їх утримання, лояльності і прибутковості. Інша точка зору доводить, що CRM є стратегічним використанням інформації, процесів, технологій і людей для управління відносинами клієнта з вашою компанією протягом усього його життєвого циклу. Врешті-решт, можна визначити CRM як таку собі всеосяжну стратегію і процес залучення, утримання та партнерства з

обраними клієнтами для створення вищої цінності для компанії та клієнта. Усе це включає в себе інтеграцію маркетингу, продажів, обслуговування клієнтів і функцій ланцюга постачання організації для досягнення більшої ефективності та результативності в забезпеченні цінності для споживача. Подібний дефінітивний плюралізм, безумовно, свідчить про важливість розгляду CRM як комплексного набору стратегій для управління тими відносинами з клієнтами, що пов’язані із загальним процесом маркетингу, продажів, обслуговування та підтримки всередині організації. Крім того, як уявляється, інформаційні системи і технології можуть бути використані не лише заради підтримки, але й задля інтеграції процесу CRM, оскільки це сприятиме задоволенню потреб клієнта через можливість відстежувати і аналізувати наявні відносини з ним [3].

Виходячи з цього CRM є привабливою сферою для досліджень через її відносну новизну та стрімке зростання.

CRM-системи можуть включати в себе ряд функціональних можливостей, таких як:

- управління контактами: зберігання інформації про поточних і потенційних клієнтів;
- управління продажами: відстеження взаємодій з клієнтами на всіх етапах продажу, від першого контакту до закриття угоди;
- управління обслуговуванням клієнтів: обробка запитів, скарг та інших взаємодій з клієнтами після продажу;
- маркетингова автоматизація: автоматизація маркетингових кампаній, сегментація клієнтів, аналіз ефективності рекламних акцій;
- аналіз даних: збір та аналіз даних про поведінку клієнтів, прогнозування майбутніх потреб, визначення ефективності різних стратегій взаємодії;
- інтеграція з іншими системами: можливість інтеграції з іншими бізнес-системами (наприклад, ERP, електронною поштою, соціальними мережами тощо).

CRM-системи важливі для компаній різних розмірів і сфер діяльності, оскільки вони допомагають зосередити увагу на потребах клієнта, оптимізувати процеси продажу і покращити якість обслуговування.

CRM-системи можна класифікувати за різними ознаками, залежно від їх функціональності, призначення і особливостей використання (див. табл. 1).

Таблиця 1 – Різновиди CRM-систем

Назва	**Особливості**	**Приклади функціональності**
Оперативні CRM	Фокус на автоматизації і покращенні бізнес-процесів, пов’язаних з управлінням взаємовідносинами з клієнтами.	Управління контактами, продажами, обслуговуванням клієнтів.
Аналітичні CRM	Фокус на аналізі даних клієнтів для підтримки прийняття рішень	Аналіз поведінки клієнтів, прогнозування продажів, аналіз ефективності маркетингових кампаній
Колаборативні CRM (або Стратегічні CRM)	Фокус на покращенні комунікацій між різними відділами організації, а також із зовнішніми партнерами	Інтеграція електронної пошти, соціальних мереж, системи для конференц-зв’язку
Кампейн-орієнтовані CRM	Основний акцент на управлінні маркетинговими кампаніями	Управління електронними листами, трекінг ефективності рекламних кампаній, сегментація клієнтів
Мобільні CRM	Доступ до CRM-даних і функцій через мобільні пристрої	Мобільний доступ до інформації про клієнтів, можливість внесення змін в базу даних в реальному часі

1.2. Огляд аналогів

Salesforce – одна з найбільш відомих та багатофункціональних CRM-систем. Вона пропонує широкий спектр інструментів для управління відносинами з клієнтами, аналізу даних та автоматизації маркетингу.

Концепція Salesforce — програмне забезпечення як послуга (SaaS), де замість клієнтського програмного забезпечення, встановленого на комп'ютері кожного користувача, вони отримують доступ до служби через веб-браузер. Це позбавляє кінцевого користувача від клопоту з оновленням клієнтського програмного забезпечення та керування ним поза браузером [4].

Переваги:

- Широкий функціонал та гнучкість настройки.
- Масштабується під різні розміри бізнесу.
- Ефективна інтеграція з іншими системами.
- Потужні аналітичні інструменти.

Недоліки:

- Висока вартість.
- Складність в освоєнні та використанні.

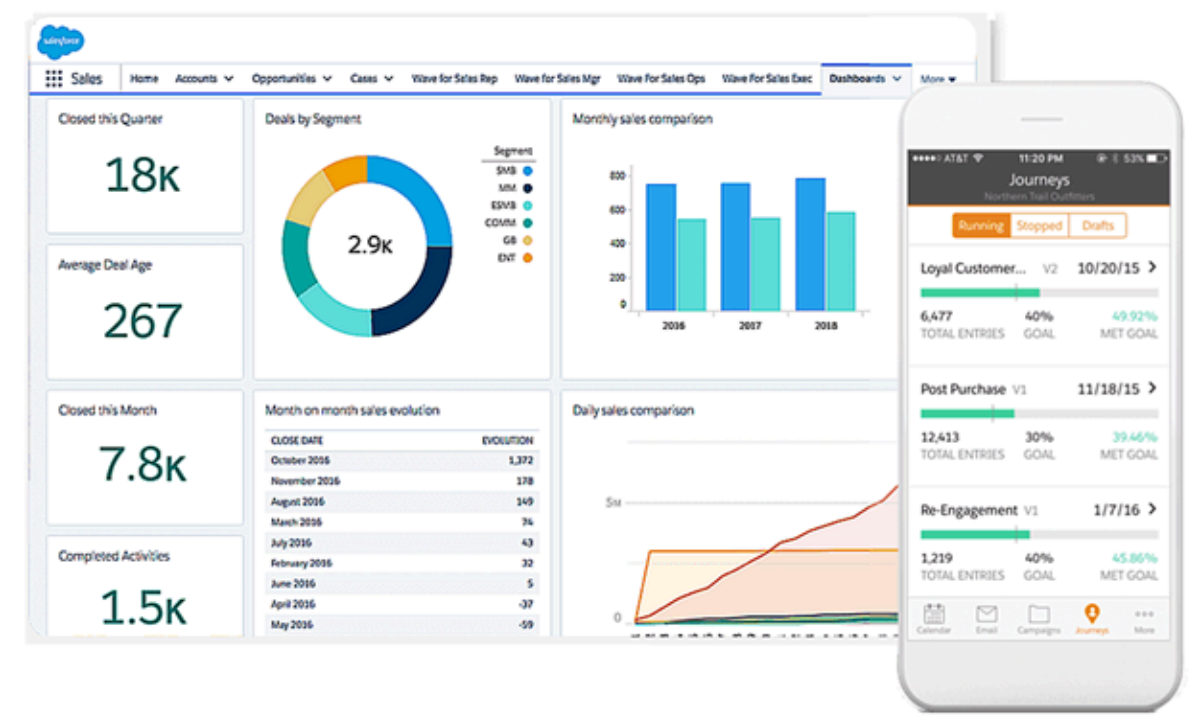

Рисунок 1.1 – Salesforce CRM [5]

HubSpot CRM – це хмарна платформа, яка має інструменти та інтеграції, які виконують багатозадачність для розвитку бізнесу. Він допомагає маркетингу, продажам, управлінню вмістом і обслуговуванню клієнтів у будь-якій точці, надаючи їм кращу стратегію та ресурси

Він надає кілька інструментів: CMS Hub , Marketing Hub , Sales Hub , Services Hub , Operations Hub. Ви можете працювати, інтегруючи його концентратори, або з окремими концентраторами. Таким чином бізнес-потік стає плавнішим за допомогою його інструментів, інтеграції та надзвичайних функцій [6].

Переваги:

- Легка в освоєнні та використанні.
- Інтеграція з маркетинговими інструментами HubSpot.
- Функціонал включає безплатну версію.

Недоліки:

- Обмежений функціонал в безплатній версії.
- Вартість може швидко зрости з додаванням додаткових модулів.

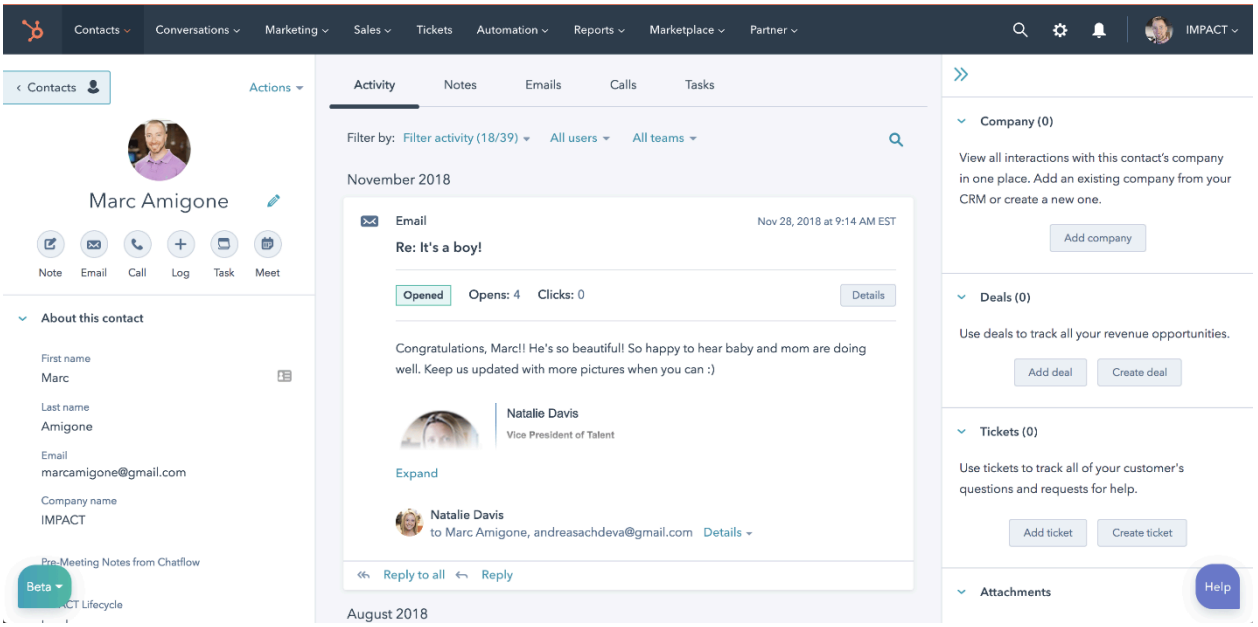

Рисунок 1.2 – HubSpot CRM [7]

Microsoft Dynamics 365 – це набір програм, інструментів і програмного забезпечення, які допомагають вам будувати стабільний бізнес. Він діє як платформа управління взаємовідносинами з клієнтами (CRM) і планування ресурсів підприємства (ERP).

Таким чином, ви можете працювати над досвідом і залученістю своїх клієнтів, одночасно відстежуючи та плануючи використання ресурсів вашої компанії з єдиної платформи [8].

Переваги:

- Інтеграція з продуктами Microsoft, такими як Office 365.
- Гнучкість та масштабованість.
- Підтримка різних мов і валют.

Недоліки:

- Висока вартість для малих організацій.
- Складність в налаштуванні та освоєнні.

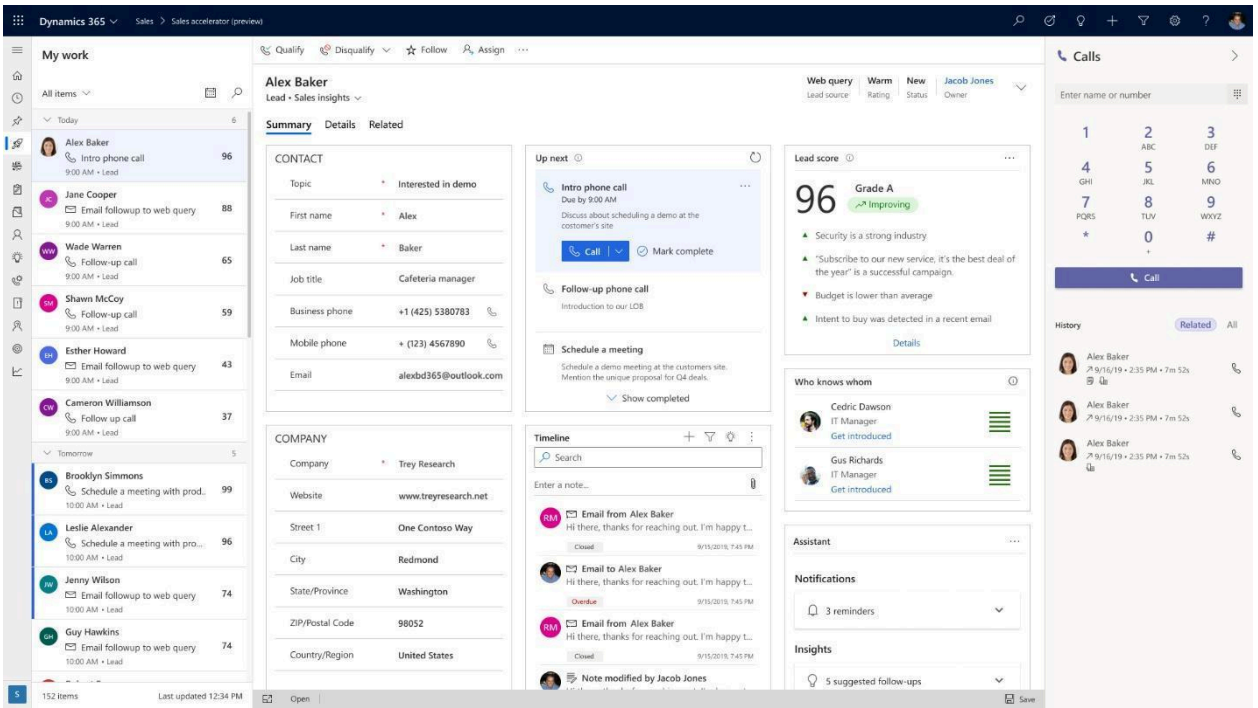

Рисунок 1.3 – Microsoft Dynamics 365 [9]

Zoho CRM – це рішення для управління взаємовідносинами з клієнтами (CRM), яке допомагає компаніям залучати, утримувати та задовольняти клієнтів для розвитку свого бізнесу. Це пропозиція від Zoho Corporation, компанії-виробника програмного забезпечення з широким спектром додатків для бізнесу.

Zoho CRM надає комплексну платформу для керування взаємодією з клієнтами в усіх точках взаємодії, включаючи продажі, маркетинг, обслуговування клієнтів і технічну підтримку. Він пропонує такі функції, як керування контактами, відстеження продажів, автоматизація маркетингу, підтримка клієнтів і розширена аналітика [10].

Переваги:

- Доступна ціна.
- Легкість використання.
- Гнучкість в налаштуваннях.

Недоліки:

- Обмежений функціонал порівняно з більш дорогими системами.
- Інтерфейс може здатися менш сучасним.

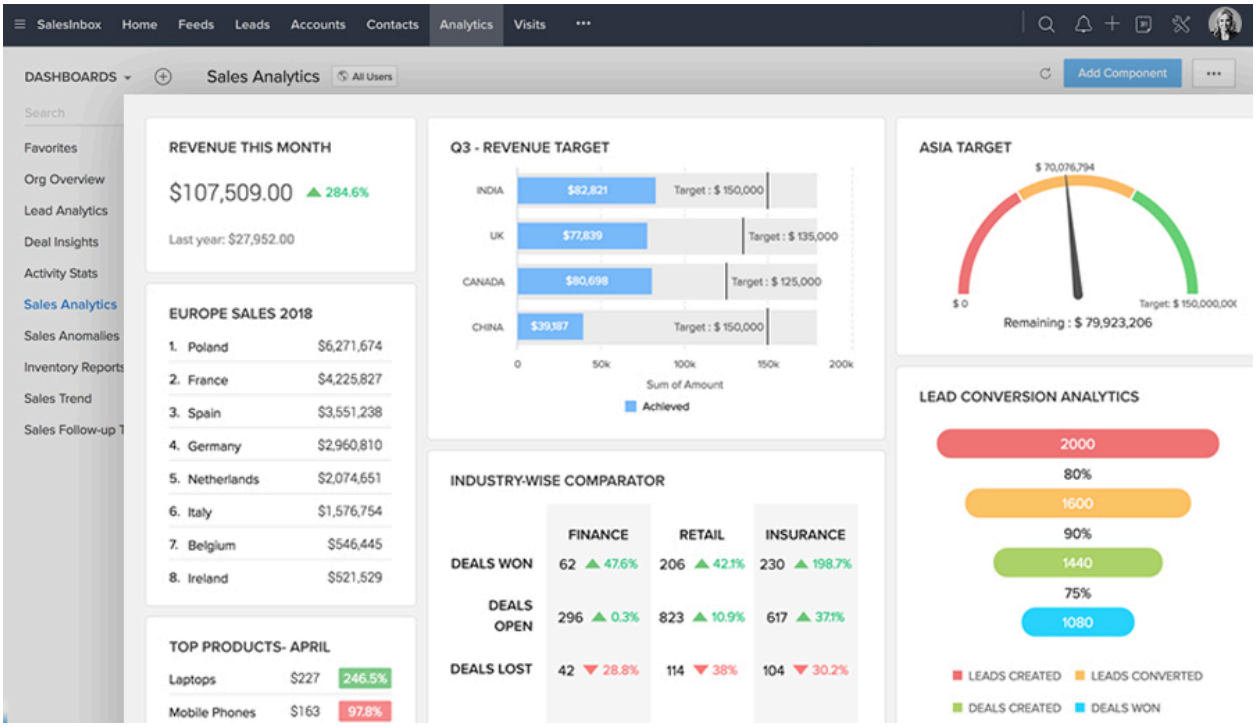

Рисунок 1.4 – Zoho CRM [11]

Pipedrive — це потужна, зручна платформа для управління взаємовідносинами з клієнтами (CRM), призначена в основному для того, щоб допомогти командам продажів керувати потенційними клієнтами та угодами, відстежувати комунікацію та автоматизувати процеси продажів. Його головна мета полягає в тому, щоб стимулювати цілеспрямовані продажі та гарантувати, що можливості не пропадуть крізь щілини [12].

Переваги:

- Орієнтована на процеси продажу.
- Легка в освоєнні та використанні.
- Інтуїтивний інтерфейс.

Недоліки:

- Обмежений функціонал для маркетингу та обслуговування клієнтів.
- Може не підходити для великих організацій зі складними потребами.

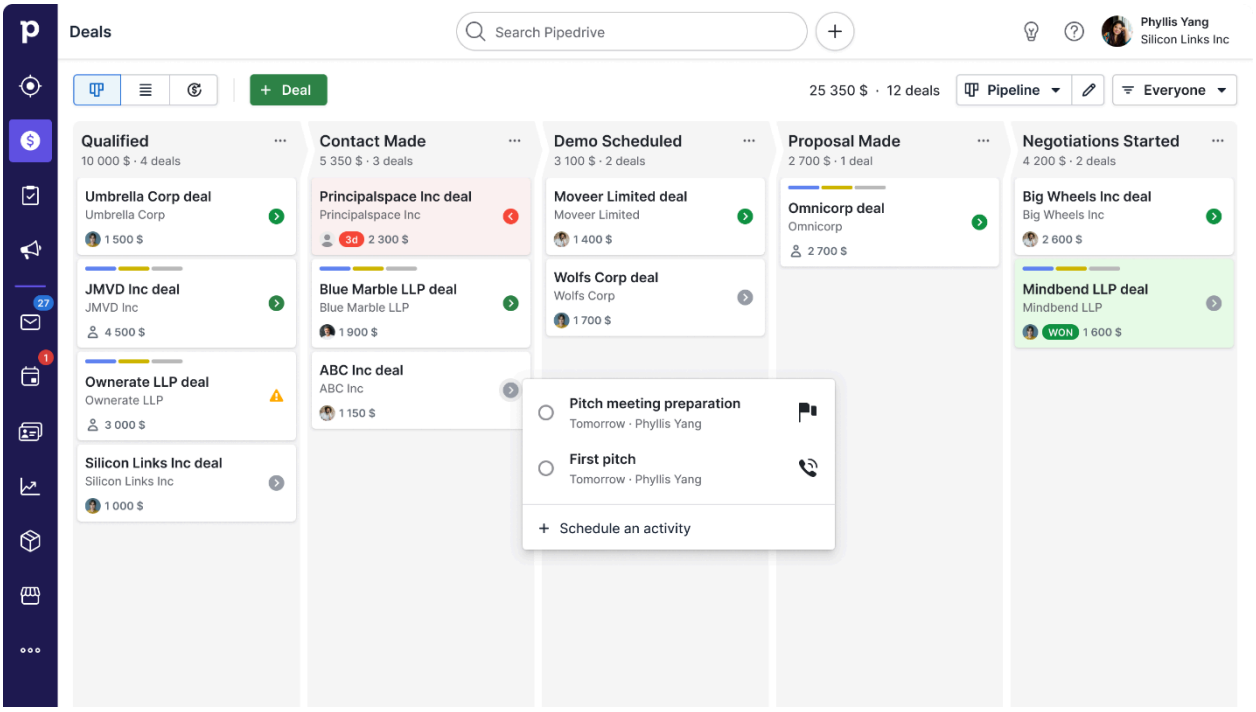

Рисунок 1.5 – Pipedrive [13]

1.3. Постановка задач

Метою роботи є розробка CRM-системи управління клієнтами для компанії «Kit Global LLC». Ця робота відповідає вимогам, передбаченим для освітнього рівня «бакалавр» зі спеціальності «Комп'ютерні науки».

Система повинна забезпечувати наступні функціональні можливості:

- можливість додання та редагування Лідів та перетворення у клієнтів зі збереженням всіх даних про нього;
- можливість додання задач для кожного з Лідів з датою дедлайну;
- можливість закріплення файлів, пов'язаних з клієнтом;
- можливість створення угод на клієнтів;
- відображення єдиної інформації по кожному з клієнтів на всіх пов'язаних з ним сторінках.

Для досягнення поставлених цілей необхідно виконати наступні завдання:

- провести аналіз предметної області та визначити її актуальність;
- здійснити порівняння аналогів системи та визначити основні їх переваги й недоліки;
- визначити оптимальні технології для виконання поставленого завдання;
- реалізувати систему управління клієнтами, використовуючи обрану технологію.

Предметом дослідження є визначення можливостей автоматизації управління клієнтами за допомогою використання спеціалізованої CRM-системи під конкретні потреби бізнесу.

Наукова новизна полягає у розробці спеціалізованої системи з використанням сучасних підходів управління клієнтами в систематизованому вигляді.

Практичне значення полягає у підвищенні ефективності та надійності управління клієнтами Kit Global за допомогою автоматизованої CRM-системи.

2. ВИБІР МЕТОДІВ РІШЕННЯ ЗАДАЧІ

2.1. Вибір технологій

Для реалізації цієї інформаційної системи було обрано мову програмування JavaScript та для забезпечення типізації мову Typescript. Цей вибір було зумовлено тим що Javascript це досить популярна мова для розробки веб сайтів, веб додатків. А також її можна використовувати в якості серверної мови програмування, а саме NodeJS.

Node.js – це платформа, яка дозволяє виконувати JavaScript на серверному рівні. Це забезпечує високу продуктивність і швидкість відгуку завдяки асинхронній архітектурі. Це було ключовим фактором при виборі технології для реалізації серверної частини [14].

Серед серверних фреймворків можна виділити такі:

- Express. Легкий і швидкий фреймворк для Node.js, що забезпечує мінімалістичний набір функцій для створення веб-додатків і API. Відомий своєю гнучкістю, простотою налаштування та великою спільнотою [15].
- NestJS. Прогресивний фреймворк для побудови ефективних і масштабованих серверних додатків на TypeScript, з архітектурою, натхненною Angular. Підтримує модульність і інтеграцію з іншими бібліотеками [16].
- Koa.js. Фреймворк, розроблений творцями Express.js, орієнтований на сучасні ES6+ функції і забезпечує більшу гнучкість та продуктивність через використання асинхронного програмування і middleware [17].
- Hapi.js. Потужний фреймворк з акцентом на конфігураційність і багатофункціональність, ідеальний для створення великих і складних проектів. Відзначається високим рівнем безпеки та підтримкою плагінів [18].
- Sails.js. MVC фреймворк, натхненний Ruby on Rails, що дозволяє швидко створювати RESTful API та реальні додатки.

Автоматично генерує API на основі бази даних і підтримує WebSockets [19].

- Meteor.js. Повноцінний фреймворк для розробки веб- і мобільних додатків з мінімальною кількістю коду, що об'єднує серверну і клієнтську частини в єдину платформу, підтримує реактивне програмування та кросплатформеність [20].
- AdonisJS. MVC фреймворк, натхненний Laravel, що пропонує структуровану архітектуру для швидкої і зручної розробки веб-додатків. Підтримує інтеграцію з базами даних через ORM, а також аутентифікацію та авторизацію [21].

Для реалізації серверної частини проекту було обрано програмний стек на базі Node.js і його фреймворку NestJS. Основними критеріями вибору стали активне використання їх на підприємстві, а також переваги самої технології..

При виборі системи керування базами даних (СКБД) було проведено аналіз кількох популярних баз даних. До прикладу такі:

- MongoDB. Документно-орієнтована база даних, яка забезпечує гнучке і масштабоване управління даними. Вона використовує документи у форматі BSON (розширення JSON), що дозволяє зберігати складні дані та забезпечувати швидкий доступ до них. Завдяки горизонтальному масштабуванню та реплікації, MongoDB може обробляти великі обсяги даних і підтримувати високу доступність [22].
- MySQL. Реляційна база даних з відкритим вихідним кодом, широко використовувана для веб-додатків та інших програмних систем. Вона підтримує SQL для роботи з даними, пропонуючи високу продуктивність, надійність і зручні інструменти для адміністрування. MySQL забезпечує транзакції ACID, що гарантує цілісність даних [23].

- PostgreSQL. Потужна реляційна база даних з відкритим кодом, яка підтримує розширені функції SQL, включаючи транзакції ACID, збережені процедури, тригери та індекси. Вона забезпечує високу відповідність стандартам SQL і можливість розширення за допомогою власних типів даних та функцій [24].
- SQLite. Вбудована реляційна база даних, яка не потребує налаштування сервера і працює як бібліотека в межах програми. Вона ідеально підходить для мобільних додатків, невеликих веб-сайтів та вбудованих систем. SQLite забезпечує простоту використання та високу продуктивність для невеликих і середніх обсягів даних [25].
- Oracle Database. Комерційна реляційна база даних, яка пропонує високу продуктивність, надійність та масштабованість. Вона підтримує широкий спектр функцій для управління даними, включаючи транзакції ACID, розподілену обробку даних, аналітику та управління великими обсягами даних. Oracle Database часто використовується в корпоративних середовищах [26].
- Microsoft SQL Server. Реляційна база даних, розроблена корпорацією Microsoft, що забезпечує високу продуктивність, безпеку та інтеграцію з іншими продуктами Microsoft. Вона підтримує транзакції ACID, аналітичні запити, та зручні інструменти для адміністрування та розробки [27].
- Cassandra. Розподілена база даних, що забезпечує високу доступність і масштабованість без єдиної точки відмови. Вона використовує архітектуру типу "ключ-значення" і особливо підходить для обробки великих обсягів даних у реальному часі. Cassandra часто використовується для великих розподілених систем, де потрібна висока доступність даних [28].

- Redis. In-memory база даних типу "ключ-значення", яка забезпечує надзвичайно швидкий доступ до даних. Вона підтримує різні типи даних, такі як списки, множини та хеші, і використовується для кешування, управління сеансами, черг повідомлень та інших задач, що потребують швидкого доступу до даних [29].
- Neo4j. Графова база даних, яка спеціалізується на управлінні та обробці даних, що мають графову структуру. Вона дозволяє легко моделювати та виконувати запити до складних мережевих взаємозв'язків. Neo4j часто використовується у випадках, де важливо відображати та аналізувати зв'язки між даними, наприклад, у соціальних мережах та рекомендаційних системах [30].

В якості системи керування базами даних (СКБД) було обрано MongoDB, котра використовується на підприємстві.

Для клієнтської частини веб-додатку було обрано фреймворк React [31], бо він також активно використовується на підприємстві. Для початку розробки від підприємця було надано корпоративний шаблон, який потім був налаштований під CRM-систему, та було додано декілька нових елементів, які були відсутні в шаблоні (наприклад сторінка коментарів, докладніше написано в розділі 3).

2.2. Проектування системи управління клієнтами

Необхідні сутності в системі. Система управління клієнтами має відображати кілька ключових сутностей для ефективної роботи:

- Ліди: потенційні клієнти, які ще перебувають на етапі розгляду пропозицій. Для кожного Ліда повинно бути надано можливість додавати задачі, документи та коментарі, що будуть доступними на всіх сторінках цього Ліда.

- Контакти: це клієнти, які вже скористались послугами компанії. Вони можуть мати одну чи декілька пов’язаних із собою угод.
- Угоди: послуги або продукти, які клієнт замовляє або замовив. Кожна угода повинна бути прив’язана до конкретного контакту.

Функціонал для кожної сутності:

- Задачі: Кожна задача має мати поля для назви та дати виконання з можливістю позначити задачу як «виконана» за допомогою спеціальної кнопки.
- Документи: Для документів має бути реалізована можливість завантаження файлів на сервер, які можна буде потім переглядати та завантажувати.
- Коментарі та історія змін: Сторінка повинна містити історію всіх взаємодій з клієнтом, а також надавати можливість додавати примітки.
- Перетворення Ліда в контакт: Після підтвердження замовлення Лід повинен перетворюватися в контакт, з можливістю автоматичного додавання до нього угод і задач, якщо це необхідно.
- Контактна сторінка: Створення нового контакту можливе лише через форму перетворення Ліда, щоб уникнути дублювання інформації. Кожен контакт повинен мати поле для статистики, яке буде показувати суму успішно виконаних та потенційних угод.
- Сторінка угод: При створенні нової угоди, необхідно обрати прив’язаний контакт та вказати суму з валютою.
- Загальний список задач: Для зручності управління задачами необхідно мати можливість перегляду всіх задач в одному списку.

Кожен з цих функціональних елементів відіграє важливу роль в загальній системі управління клієнтами. Вони спроєктовані так, щоб максимізувати ефективність взаємодій з клієнтами на всіх етапах співпраці: від потенційного інтересу до виконання замовлення та подальшої співпраці.

3. РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ ВНУТРІШНЬОГО ДОКУМЕНТООБІГУ

3.1. Програмна реалізація системи документообігу

Під час створення системи для управління клієнтами в компанії, було обрано технологічний стек, що активно використовується в нашому підприємстві. Моєю метою було створити зручну та надійну систему.

Вибір технологій.

Backend:

- В якості серверної технології було обрано фреймворк NestJS з використанням мови програмування TypeScript. Вибір був зумовлений активним використанням цього фреймворку в компанії, а також його гнучкістю.
- В якості бази даних було використано MongoDB, яка дозволяє ефективно зберігати та маніпулювати даними про клієнтів, задач, угод та документів.
- Контейнеризація: Для зручності розгортання та управління сервером було використано Docker.
- Аутентифікація: Система аутентифікації базується на JWT (JSON Web Tokens). Це дає можливість генерувати унікальні закодовані дані, які відповідають певному користувачеві в базі даних.

Frontend:

- Бібліотека: Для розробки веб інтерфейсу було обрано React через його ефективність та широкий функціонал.
- Форми: Для створення форм було використано бібліотеку react-hook-form. Це зробило процес роботи з формами більш гнучким та контрольованим.
- HTTP-клієнт: Для здійснення HTTP-запитів до сервера було використано бібліотеку Axios.

- State Management: Обробка та управління станами додатка відбувається за допомогою Redux Saga. Це дозволяє ефективно обробляти асинхронні події та взаємодіяти з сервером.

План реалізації:

- Планування та архітектура: Спочатку було зроблено оцінку обсягу роботи та вибрана архітектура додатка.
- Настройка проєкту: Здійснено ініціалізацію проєкту, налаштовано базові конфігурації для NestJS, Docker та інших інструментів, а також спроектована концептуальна модель бази даних.
- Розробка backend-частини: Створено API для роботи з клієнтами, задачами, угодами, документами та коментарями. Реалізовано систему аутентифікації на основі JWT.
- Розробка frontend-частини: Інтегровано API в React-додаток, реалізовано користувацький інтерфейс для управління даними.
- Тестування: Здійснено комплексне тестування системи на наявність помилок та розбіжності з бізнес-вимогами.
- Публікація: Завершальний етап містив настройку Docker контейнера та розгортання системи на сервері компанії.

3.2. Розробка основного функціоналу

Функціональність коментарів та подій.

Однією з важливих функцій системи є можливість залишати коментарі та фіксувати події. Цей механізм було реалізовано з використанням динамічних шаблонів, що зберігаються в базі даних.

Реалізація:

Зберігання даних. При додаванні коментаря або події в базу даних зберігається не готовий текст, а тільки шаблон з додатковою інформацією. Наприклад, для події створення Ліда, зберігається шаблон «Лід {Ім’я клієнта} додано користувачем {Ім’я адміністратора}» та ідентифікатор клієнта та адміністратора.

Агрегація даних. При відображенні коментарів або подій, система застосовує агрегацію для збирання потрібних даних і підставляє їх у шаблон. Зокрема, ім’я адміністратора, що зробив зміни, може бути динамічним чином змінено у всіх відповідних коментарях та подіях, якщо сам адміністратор змінить своє ім’я в системі.

Приклад використання.

Коли створюється новий Лід, система автоматично генерує коментар з використанням шаблону. Якщо адміністратор з іменем «Віктор» створює Лід на ім’я «Іванов І.», то коментар буде виглядати як «Лід Іванов І. додано користувачем Віктор». Якщо пізніше «Віктор» змінить своє ім’я на «Віталій», текст коментаря автоматично оновиться на «Лід Іванов І. додано користувачем Віталій».

Ця функціональність значно полегшує управління історією взаємодії з клієнтами та робить систему більш гнучкою та адаптивною.

Інтеграція Swagger

Для спрощення розробки та документування API було інтегровано Swagger, що значно полегшує тестування API та робить систему більш прозорою для інших розробників які будуть надалі її розробляти.

Створення власних декораторів

Загальний декоратор.

Я створив власний декоратор в NestJS, який одночасно декорує метод та параметр функції. Це забезпечує додаткову гнучкість та можливість повторного використання коду.

Декоратор для валідації MongoId.

Цей декоратор виконує подвійну функцію. При запуску програми він перевіряє: чи правильно використовується декоратор на різних точках доступу, і, за необхідності, виводить повідомлення у консоль про можливі некоректності, такі як наявність більше одного параметра у точці доступу.

Під час роботи програми декоратор перевіряє валідність MongoId, отриманого від клієнта, і конвертує його в потрібний формат. У разі виявлення невірного MongoId, клієнт отримує повідомлення про помилку у форматі {'status': 422, 'message': '{paramName} is not valid'}.

Ці додаткові інструменти та оптимізації не лише підвищують якість коду, але й роблять процес розробки більш ефективним, забезпечуючи високий рівень безпеки та надійності системи.

Декоратор для пагінації.

На базі загального декоратора також було створено ще один специфічний декоратор для пагінації. Цей декоратор виконує наступні функції:

- Отримання Query Параметрів: Декоратор приймає в query поля limit, page, sortby, order.
- Валідація: Всі отримані поля проходять процес валідації для перевірки їх правильності.
- Перетворення: Після валідації, декоратор перетворює ці поля у sort, skip, limit, які потім використовуються в MongoDB для агрегації даних.
- Інтеграція з Swagger: Щоб полегшити розробку та документування, декоратор автоматично додає у Swagger ApiQuery з отриманими полями, а також ApiOkResponse з типом відповіді, що буде згенеровано.

Цей декоратор значно спрощує процес роботи з пагінацією, автоматизує валідацію і перетворення даних, і забезпечує зручне документування через Swagger. Завдяки цьому, робота з пагінованими запитами стає набагато ефективнішою і надійнішою.

Стратегія видалення записів

Сховане видалення для збереження консистентності. Однією з ключових фіч системи є специфічний підхід до видалення записів. Замість

безповоротного видалення записів з бази даних, було імплементовано стратегію «схованого» видалення.

Реалізація. Коли користувач системи видаляє запис (наприклад, інформацію про клієнта, задачу, угоду або коментар), дані фактично не видаляються з бази даних. Замість цього, запис отримує певний статус, що робить його «невидимим» для звичайних операцій системи. Таким чином, вся історія взаємодій зберігається, і система продовжує працювати без помилок пов’язаних з відсутністю очікуваних записів.

Цей підхід має кілька переваг:

- Безпека Даних: Жодна важлива інформація не буде втрачена, що є критично важливим для бізнес-процесів.
- Консистентність: Зберігається цілісність даних та забезпечується уникнення помилок, які могли б виникнути при фізичному видаленні.
- Відновлення: Це також полегшує можливість відновлення «видалених» записів, якщо це стане необхідним.

Така стратегія значно підвищує надійність системи і дозволяє уникнути непередбачених проблем, пов’язаних з видаленням даних.

3.3. Розробка графічного інтерфейсу системи.

Сторінка входу

Сторінка входу представлена полями для введення e-mail та пароля. Після заповнення цих полів, користувач може натиснути кнопку «Увійти» для доступу до системи управління клієнтами.

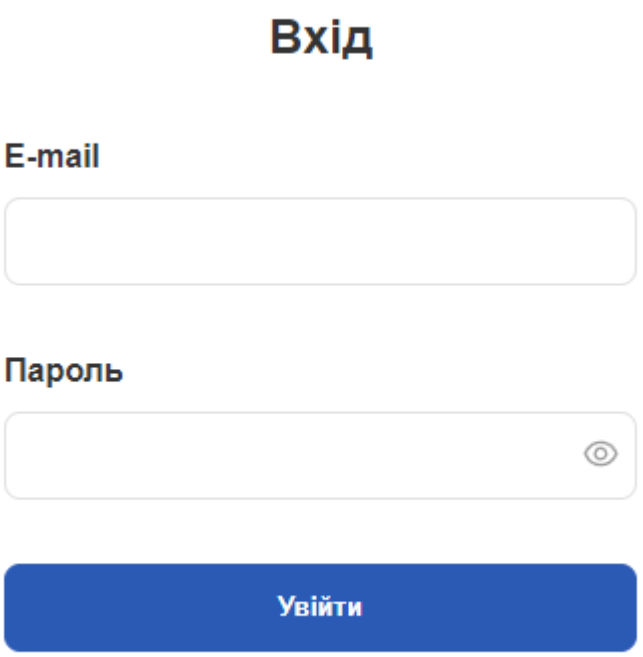

Рисунок 3.1 – Форма входу

Сторінка зі списком Лідів

На цій сторінці відображається таблиця з інформацією про нових клієнтів, які ще не зробили замовлення, але розглядають таку можливість. Лідів можна також сортувати за часом створення. Користувач може перейти до сторінки редагування відповідного Ліда або створення нового.

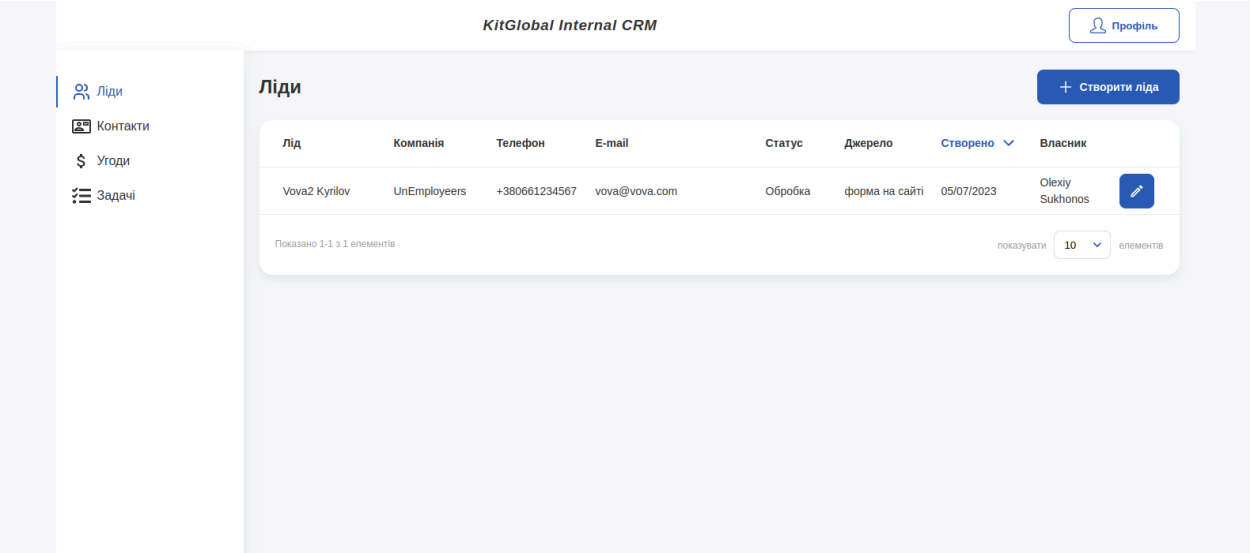

Рисунок 3.2 – Сторінка зі списком Лідів

Сторінка з контактами

Ця сторінка також містить таблицю, але тут перераховуються вже чинні контакти. Контакти можна тільки редагувати, оскільки їх створення відбувається автоматично при перетворенні Ліда.

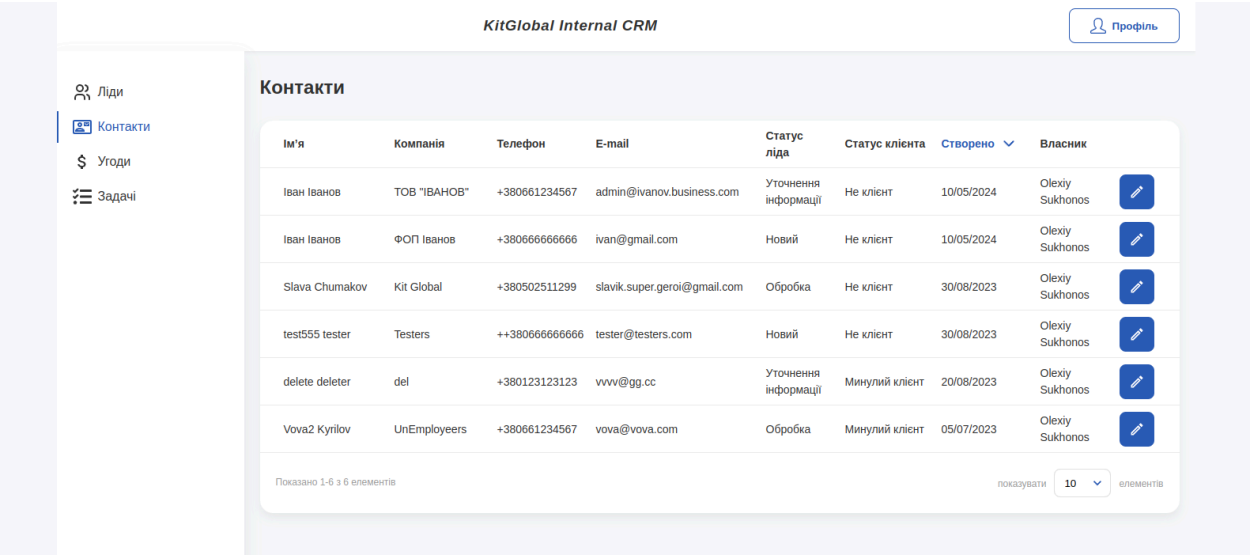

Рисунок 3.3 – Сторінка з контактами

Сторінка з угодами

На цій сторінці користувач бачить список угод у вигляді таблиці, в якій можна відстежувати стадію кожної угоди та потенційний дохід від неї.

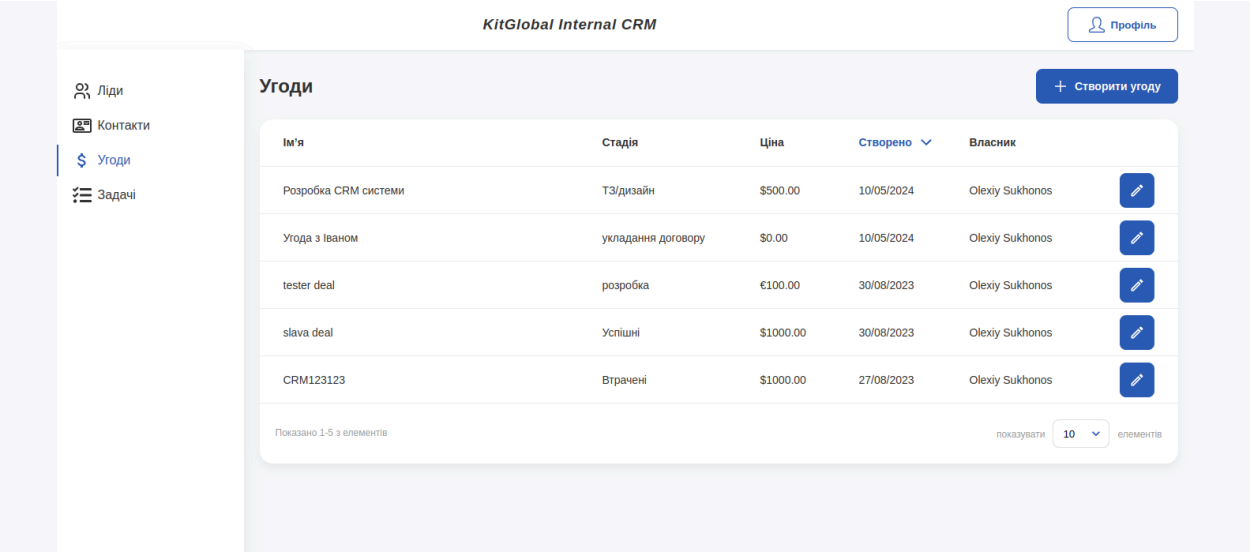

Рисунок 3.4 – Сторінка з угодами

Сторінка з усіма задачами

Тут відображаються всі задачі, незалежно від асоційованого з ними Ліда, контакту або угоди. Задачі можна створювати, редагувати та видаляти. Користувач також може позначити задачу як виконану.

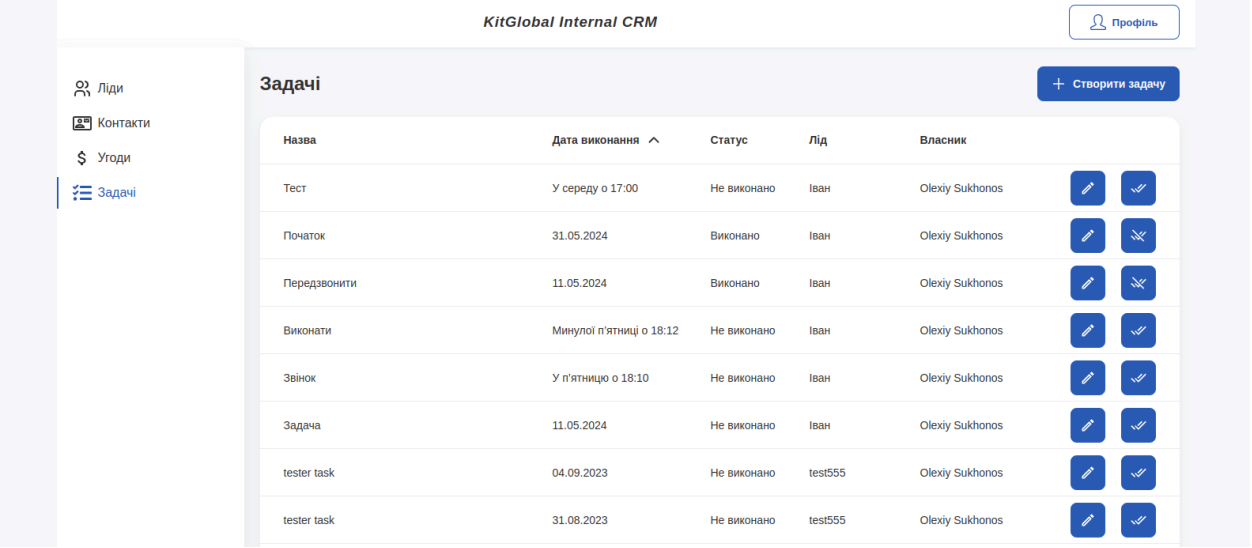

Рисунок 3.5 – Сторінка з усіма задачами

Сторінка створення Ліда

На цій сторінці користувач може створити новий Лід, заповнивши необхідні поля: ім'я, прізвище, назва компанії, веб-сайт, номер телефону,

електронна пошта, статус і джерело. Форма для введення цієї інформації проста та зрозуміла.

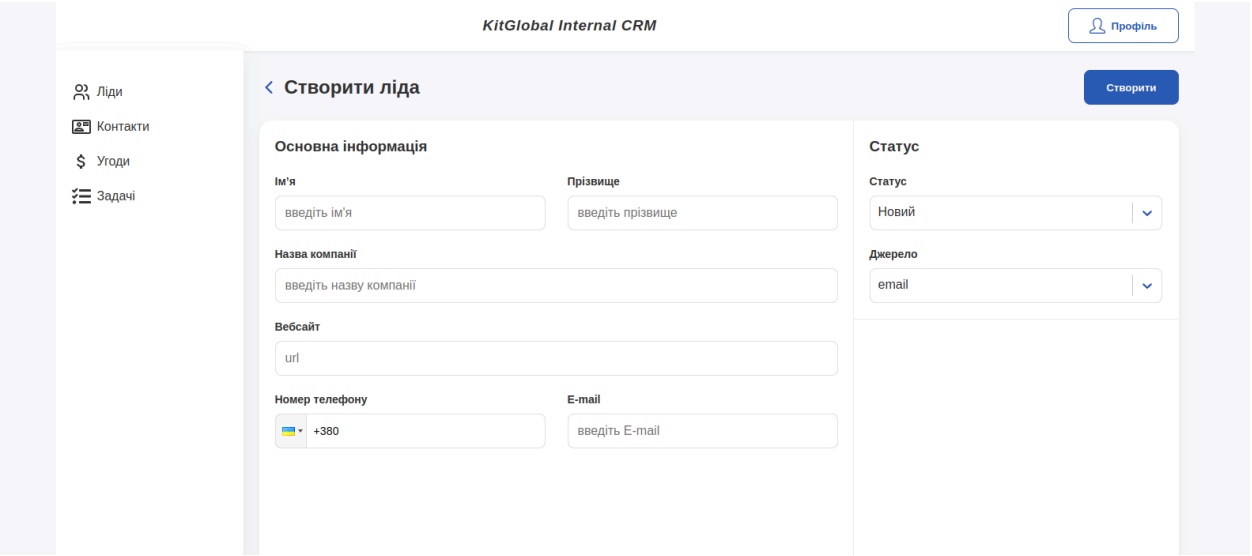

Рисунок 3.6 – Сторінка створення Ліда

Сторінка редагування Ліда

На цій сторінці можна оновити інформацію про наявного Ліда. Поля для редагування такі ж, як і при створенні. Окрім цього, є модальне вікно для перетворення Ліда в клієнта, в якому можна створити задачу та угоду.

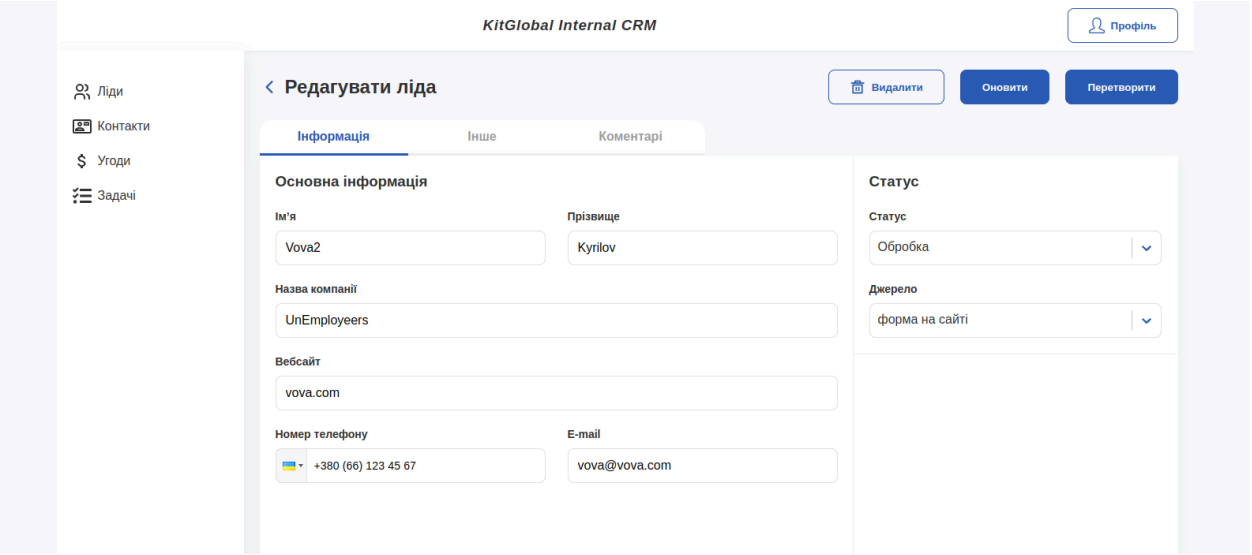

Рисунок 3.7 – Сторінка редагування Ліда

Модальне вікно для перетворення Ліда

Після натискання на кнопку «Перетворити Ліда» в модальному вікні користувачу пропонується декілька опцій. Тут можна створити нову угоду, вказавши її назву, а також створити нову задачу, вказавши її назву та дату, до якої її потрібно виконати. Це вікно дає можливість одночасно перетворити Ліда в контакт, створити з ним угоду та задати задачу для подальшої роботи.

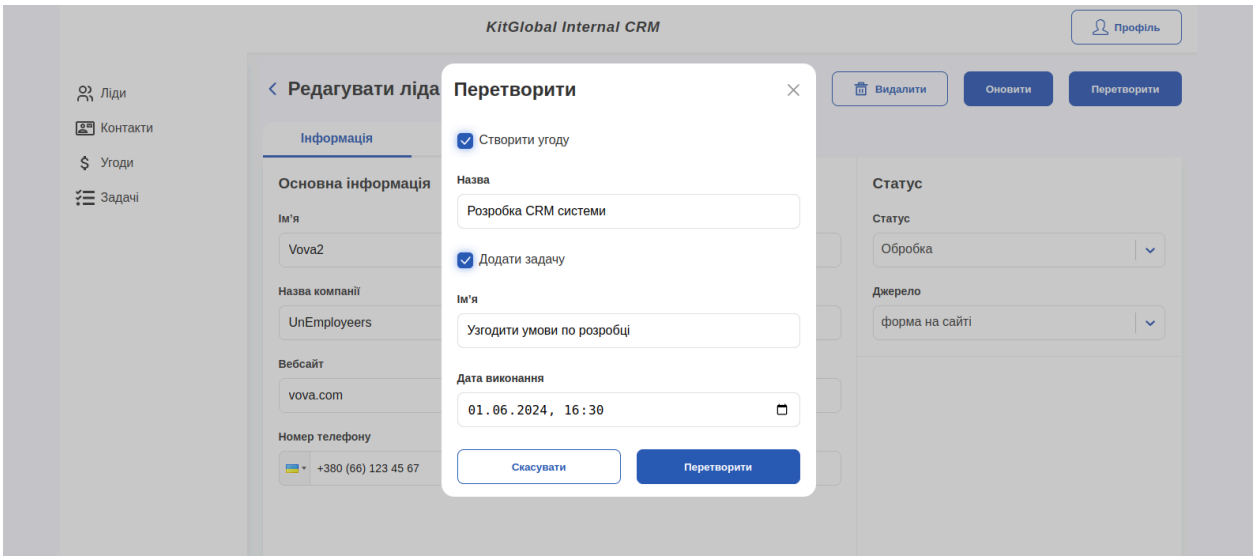

Рисунок 3.8 – Модальне вікно для перетворення Ліда

Сторінка для створення нового контакту відсутня в системі. Функціонал створення нового контакту замінений опцією перетворення Ліда на контакт на відповідній сторінці Ліда для запобігання дублювання клієнта

Сторінка редагування контакту

Ця сторінка дозволяє редагувати інформацію про наявний контакт. Окрім базових полів, є додаткові параметри для визначення статусу контакту (клієнт, не клієнт, минулий клієнт) та його перспективності (перспективний чи ні). Поле статусу заблоковано, оскільки, після перетворення Ліда, його статус може бути тільки «Конвертований». Також тут відображається сумарний дохід від усіх угод.

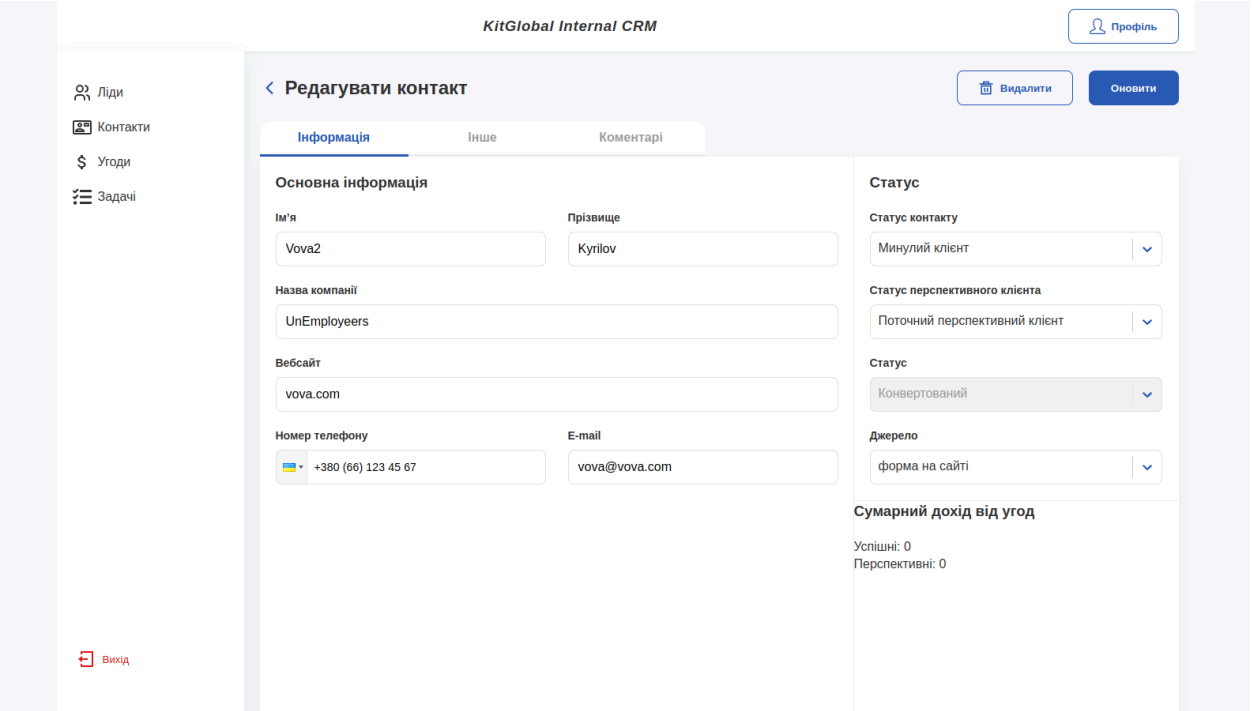

Рисунок 3.9 – Сторінка редагування контакту

Сторінка редагування угоди

Ця сторінка містить поля для редагування інформації про угоду: її назву, обраний контакт, ціну та валюту, статус і джерело. Ви також можете переглядати та редагувати пов’язані задачі, завантажені документи та коментарі. При зміні статусу угоди на «скасований» або «упущений», з’являється додаткове вікно, де можна обрати причину такого статусу зі списку. Це дозволяє зберегти додаткову інформацію для аналітики та подальшого вивчення причин невдач.

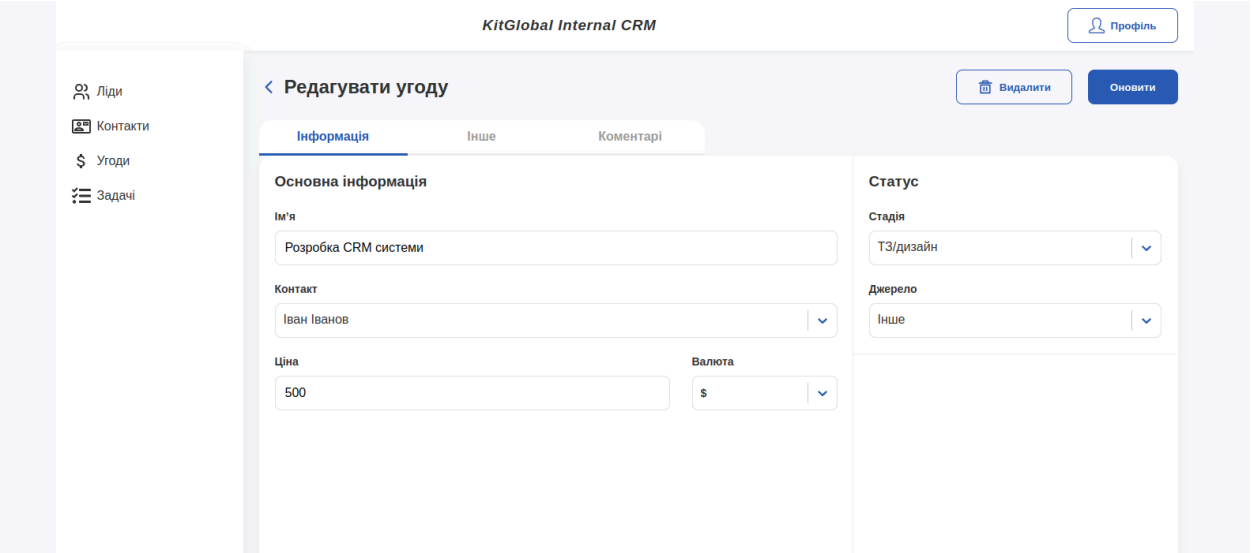

Рисунок 3.10 – Сторінка редагування угоди

Сторінка з задачами та документами

Ці меню доступні на кожній сторінці редагування (Ліда, контакту чи угоди). Тут можна створювати, редагувати та відмічати задачі в якості виконаних, а також завантажувати документи. Після виконання завдання, воно автоматично позначається червоним кольором та закреслюється.

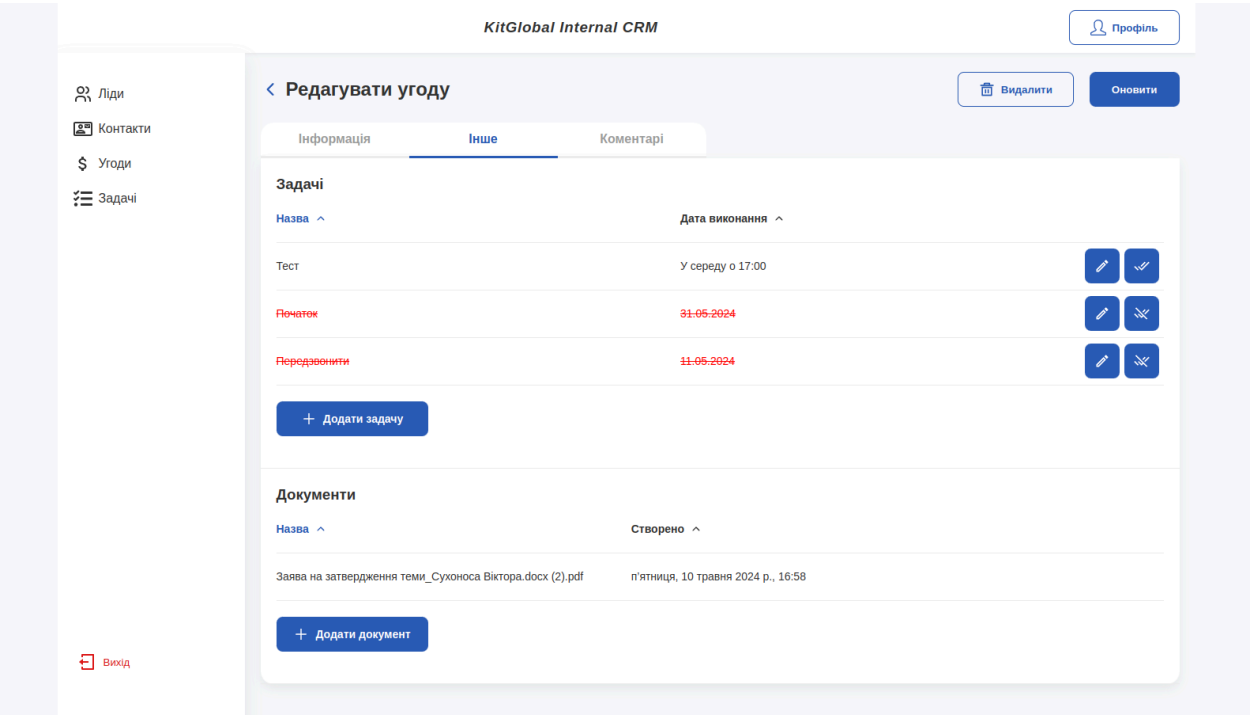

Рисунок 3.11 – Сторінка з задачами та документами

Модальне вікно створення та редагування задач

У модальному вікні, для створення та редагування задач, користувачу надається можливість ввести назву задачі. Далі: є поле для вибору відповідного Ліда, з яким пов’язана ця задача. Також присутнє поле, в якому можна вказати дату і час, до якого задачу необхідно виконати. Це дозволяє систематизувати роботу і краще планувати взаємодію з клієнтами.

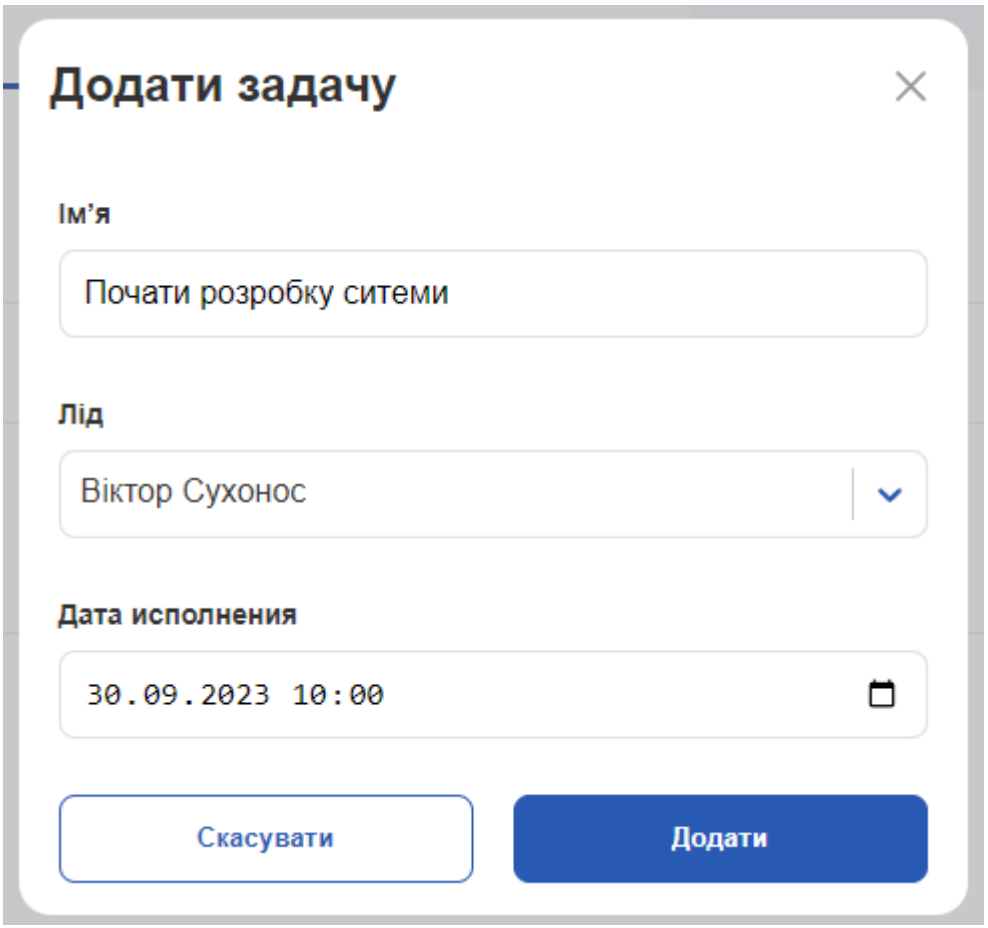

Рисунок 3.12 – Модальне вікно створення задачі

Модальне вікно завантаження документів

При натисканні на кнопку «Додати документ», з’являється модальне вікно для завантаження файлів. В цьому вікні користувачу пропонується вибрати файл зі свого комп’ютера для подальшого завантаження в систему. Після вибору файлу і підтвердження завантаження, документ стає доступним для перегляду та завантаження на комп’ютер.

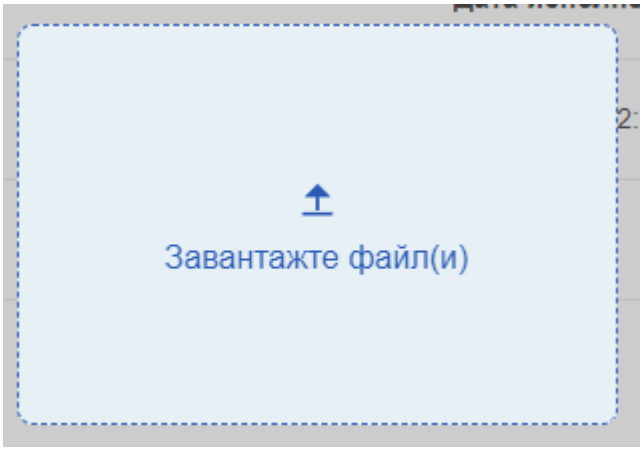

Рисунок 3.13 – Модальне вікно завантаження документів

Сторінка з коментарями й подіями

Цей функціонал також доступний на кожній сторінці редагування (Ліда, контакту, угоди). Тут адміністратор може залишити коментарі та переглядати історію подій, які відбулися з Лідом, контактом чи угодою. Зокрема, можна дізнатися дату створення Ліда, перетворення його на контакт та інші важливі події.

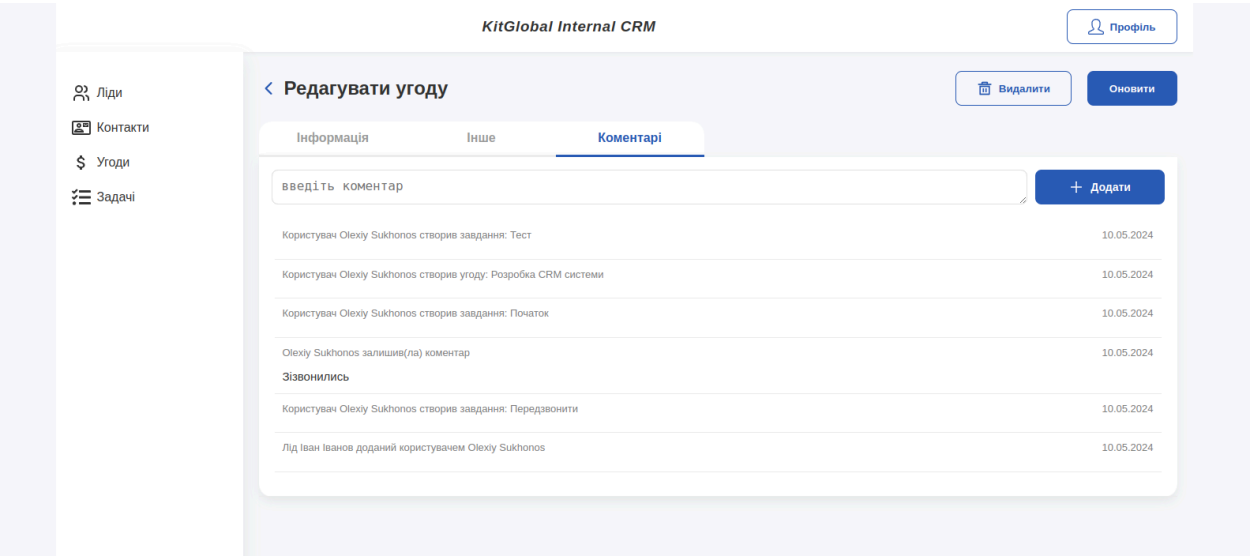

Рисунок 3.14 – Сторінка з коментарями й подіями

Сторінка профілю

Цей функціонал дозволяє змінювати такі налаштування профілю як: ім’я адміністратора, його прізвище, e-mail та, якщо буде потрібно, то одразу і

пароль. При зміні ім’я адміністратора, автоматично змінюються всі коментарі де підставляються оновлені дані.

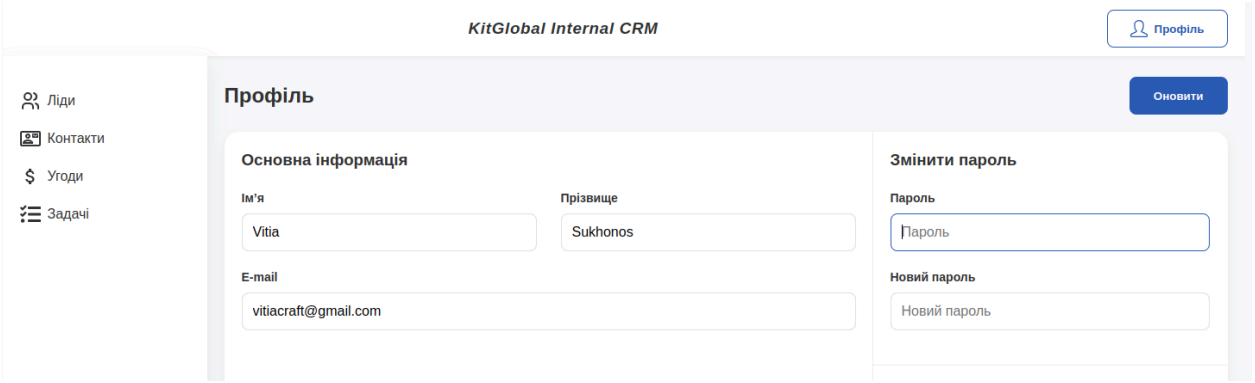

Рисунок 3.15 – Сторінка профілю

Також у верхній частині екрану (хедері) можна побачити кнопку для переходу на сторінку профілю.

Тож в результаті було розроблено програмну реалізацію системи управління клієнтами для компанії Кіт Глобал. В подальшому її можна буде вдосконалити та додати новий функціонал який необхідний для більш якісного управління клієнтами

ВИСНОВКИ

В ході написання кваліфікаційної роботи бакалавра було проведено аналіз доступних на ринку рішень з метою зрозуміти їх можливості та обмеження. Це допомогло нам в розробці власної інформаційної системи для управління клієнтами на основі фреймворку NestJS.

В ході виконання практичної частини кваліфікаційної роботи було створено внутрішню інформаційну систему для управління клієнтами, яка взаємодіє з базою даних та включає аутентифікацію за допомогою JWT-токенів. Система була розроблена на основі фреймворку NestJS, з використанням MongoDB для зберігання даних, і Docker для ізоляції сервісів від інших сервісів компанії.

Ця система не лише виконує базові функції управління Лідами, контактами, угодами та задачами, але й пропонує розширені можливості для аналізу та оптимізації діяльності. Це містить в собі додавання коментарів, завантаження документів, і можливості для персоналізації інформації про клієнтів.

Серед додаткових функцій варто відзначити модальні вікна для конвертації Лідів та створення угод, реалізовану систему пагінації з використанням власного декоратора, а також опції для управління кількістю відображених елементів на сторінці. Також було реалізовано функціонал для зберігання причин скасованих або упущених угод.

Загалом, ця робота дала глибокі навички в розробці систем для управління клієнтами, роботі з базами даних, UI/UX дизайну, а також в тонкощах програмування та системного аналізу. Цей досвід виявився не тільки безцінним для мене як для розробника, але і дуже корисним для будь-якої компанії, яка прагне оптимізувати свої внутрішні процеси та підвищити ефективність управління своєю клієнтською базою.

СПИСОК ЛІТЕРАТУРИ

1. Оцінка ефективності системи управління взаємовідносинами з клієнтами на підприємстві [Електронний ресурс]. URL: https://ir.kneu.edu.ua/server/api/core/bitstreams/c6b8d29d-4dc9-4e12-bdbd-34feddc7c07a/content
2. Система управління взаємовідносинами з клієнтами: теоретичний аспект [Електронний ресурс]. URL: https://dspace.nuft.edu.ua/server/api/core/bitstreams/615f1870-043d-4382-bf3e-fcefeba7017f/content
3. Сутність управління взаємовідносинами з клієнтами за допомогою інформаційно-аналітичних crm-систем [Електронний ресурс] URL: https://eprints.kname.edu.ua/61604/1/Ch._3_-_%D1%8D%D0%BD_2021-10-11.pdf
4. What is Salesforce? A Beginner's Guide - TechTarget.com [Електронний ресурс]. URL: https://www.techtarget.com/searchcustomerexperience/definition/Salesforcecom.
5. Salesforce.com. Salesforce: The Customer Company [Електронний ресурс]. URL: https://www.salesforce.com/.
6. HubSpot CRM Features And Benefits Guide | MakeWebBetter [Електронний ресурс]. URL: https://makewebbetter.com/blog/hubspot-crm-features-and-benefits/.
7. HubSpot. Marketing Hub, Sales Hub, Service Hub, and CRM [Електронний ресурс]. URL: https://www.hubspot.com/.
8. What Is Microsoft Dynamics 365? | Resco [Електронний ресурс]. URL: https://www.resco.net/learning/dynamics-365/.
9. Microsoft. Microsoft Dynamics 365 [Електронний ресурс]. URL: https://www.microsoft.com/en-us/dynamics365.

10. What is Zoho CRM | Definition and meaning [Електронний ресурс]. URL: https://www.adnaliza.com/en/dictionary/zoho-crm/.
11. Zoho Corporation. Zoho CRM | Top-rated Sales CRM Software by Customers [Електронний ресурс]. URL: https://www.zoho.com/crm/.
12. What is Pipedrive - The basics of Pipedrive the CRM... [Електронний ресурс]. URL: https://cjwray.com/what-is-pipedrive/.
13. Pipedrive. Sales CRM & Pipeline Management Software [Електронний ресурс]. URL: https://www.pipedrive.com/.
14. Astrology Page. Що таке node.js? - визначення з техопедії - Аудіо 2023 [Електронний ресурс]. URL: https://uk.theastrologypage.com/node-js.
15. Express - Node.js web application framework [Електронний ресурс]. URL: https://expressjs.com/.
16. HashDork. Підручник Nest.JS для початківців у 2023 році [Електронний ресурс]. URL: https://hashdork.com/uk/nest-js-tutorial/.
17. Koa - Next generation web framework for Node.js [Електронний ресурс]. URL: https://koajs.com/
18. Hapi - The simple, secure framework developers trust [Електронний ресурс]. URL: https://hapi.dev/
19. Sails - Realtime MVC Framework for Node.js [Електронний ресурс]. URL: https://sailsjs.com/
20. Meteor - The JavaScript App Platform [Електронний ресурс]. URL: https://www.meteor.com/
21. AdonisJS - A fully featured web framework for Node.js [Електронний ресурс]. URL: https://adonisjs.com/
22. DevZone. Основи MongoDB [Електронний ресурс]. URL: https://devzone.org.ua/post/osnovi-mongodb.
23. MySQL - The world's most popular open source database [Електронний ресурс]. URL: https://www.mysql.com/
24. PostgreSQL - The World's Most Advanced Open Source Relational Database [Електронний ресурс]. URL: https://www.postgresql.org/

25. SQLite - Home Page [Електронний ресурс]. URL: https://www.sqlite.org/
26. Oracle Database - Oracle's enterprise relational database product [Електронний ресурс]. URL: https://www.oracle.com/database/
27. Microsoft SQL Server - SQL Server 2022 [Електронний ресурс]. URL: https://www.microsoft.com/sql-server
28. Apache Cassandra - Open Source NoSQL Database with Apache Cassandra [Електронний ресурс]. URL: https://cassandra.apache.org/
29. Redis - Redis: In-memory data structure store [Електронний ресурс]. URL: https://redis.io/
30. Neo4j - The Leader in Graph Databases [Електронний ресурс]. URL: https://neo4j.com/
31. React. Посібник: знайомство з React [Електронний ресурс]. URL: https://uk.legacy.reactjs.org/tutorial/tutorial.html.

Додаток А. Код клієнтської частини

Сторінка списку лідів:

```
import { IoIosArrowUp } from 'react-icons/io';
import { PageWrapper, HeadingWrap, BottomContainer, Pagination,
SelectLimit } from 'components';
import { useAppSelector, useToggle, useAppDispatch } from 'hooks';
import 'react-datepicker/dist/react-datepicker.css';
import { AiOutlinePlus } from 'react-icons/ai';
import { useEffect, useState } from 'react';
import { t } from 'i18next';
import { GlobalStyle } from 'utils/globalStyle';
import { NewLeadsLink, TableHeadLead, TableLineLead } from
'./MyLeads.styled';
import { allLeadsAction } from 'store/Lead/actions';
import { nanoid } from 'nanoid';
import { MdOutlineModeEditOutline } from 'react-icons/md';
import { format } from 'date-fns-tz';
import { LinkToEdit } from '..';

const MyLeads: React.FC = () => {
  const [sortParams, setSortParams] = useState({ growth: true, sortBy:
'createdAt' });
 const { isOpen } = useToggle();
  const { totalCount, LeadsList, limit, page } = useAppSelector(state =>
state.leadsState);
 const totalPage = Math.ceil(totalCount / +limit);
 const dispatch = useAppDispatch();

 const handleLimitPage = (e: React.ChangeEvent<HTMLSelectElement>) => {
   dispatch(allLeadsAction.changeLimit(e.currentTarget.value));
 };

 const onPageClick = ({ selected }: { selected: number }) => {
   window.scrollTo({ top: 0, behavior: 'smooth' });
   dispatch(allLeadsAction.changePage(selected + 1));
 };
 useEffect(() => {
   if (+page > totalPage && totalPage !== 0) {
     dispatch(allLeadsAction.changePage(1));
   }
   dispatch(
     allLeadsAction.getLeadsAction({
       sortBy: sortParams.sortBy,
```

```
      order: sortParams.growth ? -1 : 1,
      limit,
      page,
    })
  );
}, [sortParams, limit, page, totalPage]);

return (
  <>
    {' '}
    <HeadingWrap>
      <h2>{t('addressLink.leads')}</h2>
      <NewLeadsLink to="/new-leads">
        {' '}
        <AiOutlinePlus />
        {t('create.createLead')}
      </NewLeadsLink>
    </HeadingWrap>
    <PageWrapper>
      <GlobalStyle isModalOpen={isOpen} />
      <TableHeadLead>
        <li>{t('main.lead')}</li>
        <li>{t('main.company')}</li>
        <li>{t('main.phone')}</li>
        <li>{t('main.email')}</li>
        <li>{t('main.status')}</li>
        <li>{t('main.source')}</li>
        <li
          onClick={() => {
            setSortParams(prev => {
              return { ...prev, growth: !prev.growth, sortBy: 'createdAt'
};

            });
          }}
          id={sortParams.sortBy === 'createdAt' ? 'active' : 'inactive'}
          className={sortParams.growth ? 'up' : 'down'}
          style={{ cursor: 'pointer' }}
        >
          {t('main.created')}
          <IoIosArrowUp />
        </li>
        <li>{t('main.owner')}</li>
      </TableHeadLead>
      <TableLineLead>
```

```
          {LeadsList.map(item => {
            return (
              <li key={nanoid(4)} title={item.name}>
                {' '}
                <div>{item.name}</div>
                <div>{item.company}</div>
                <div>{item.phone}</div>
                <div>{item.email}</div>
                <div>{t(`status.lead.${item.status}`)}</div>
                <div>{t(`source.${item.source}`)}</div>
                <div>{format(new Date(item.createdAt), 'dd/MM/yyyy')}</div>
                <div>{item.owner}</div>
                <div>
                  <LinkToEdit
                    to={`/edit-lead/${item._id}`}
                    onClick={() => {
                      window.scrollTo({ top: 0, behavior: 'smooth' });
                    }}
                  >
                    <MdOutlineModeEditOutline />
                  </LinkToEdit>
                </div>
              </li>
            );
          })}
        </TableLineLead>
        {totalPage !== 0 && (
          <BottomContainer>
            <div className={LeadsList.length < 1 ? 'visually-hidden' : ''}>
              {' '}
              {t('main.shown')}{' '}
              {page === 1
                ? 1
                : page === totalPage
                ? (+page - 1) * +limit + 1
                : +page * +limit + 1 - +limit}
                 -{+page === 1 ? LeadsList.length : +page === +totalPage ?
totalCount : +limit * +page}{' '}
              {t('main.of')} {totalCount} {t('main.items')}
            </div>
            <div>
              {totalPage > 1 && (
                <Pagination
                  onPageChange={onPageClick}
```

```
                pageCount={totalPage}
                initialPage={+page - 1}
              />
            )}
          </div>
          <div>
            {' '}
            {t('main.show')}
            <SelectLimit value={limit} onChange={handleLimitPage}>
              <option value="2">2</option>
              <option value="5">5</option>
              <option value="10">10</option>
              <option value="15">15</option>
              <option value="20">20</option>
            </SelectLimit>
            {t('main.items')}
          </div>
        </BottomContainer>
      )}
    </PageWrapper>
  </>
 );
};
export default MyLeads;
```

Діалогове вікно перетворення ліда у контакт:

```
import {
 CreateTaskWrap,
 BottomButtonWrapModal,
 HeadingCreatingTaskWrap,
 NewSevicesInputsWrap,
} from './MyLeads.styled';
import { t } from 'i18next';
import { RxCross2 } from 'react-icons/rx';
import { Controller, useForm } from 'react-hook-form';
import { Checkbox, InputCustom } from 'components';
import { ErrorMessage, InputWrap } from 'pages/Leads/MyLeads.styled';
import { useAppDispatch } from 'hooks/hooks';
import { TFormsTask } from './ModalAddTask';
import { allTasksAction } from 'store/Task/actions';
import { TNewTask } from 'store/Task/types';
import { CheckboxWrap } from './MyLeads.styled';
import { allContactsAction } from 'store/Contact/actions';
```

```
import { TDealForm } from 'pages/Deals/types';
import { useNavigate } from 'react-router-dom';

type TProps = {
 close: () => void;
 leadView: string;
};

type TTransformForm = {
 addDeal: boolean;
 deal: TDealForm;
 addTask: boolean;
 task: TFormsTask;
};

export const ModalTransform: React.FC<TProps> = ({ close, leadView }) => {
 const dispatch = useAppDispatch();
 const {
   handleSubmit,
   setValue,
   watch,
   control,
   formState: { errors },
 } = useForm<TTransformForm>();
 const navigate = useNavigate();

 const onSubmit = async (data: TTransformForm) => {
   console.log(data);
   if (data.addTask) {
     const task: TNewTask = {
       data: {
         name: data.task.name,
         lead: leadView,
         todoDate: data.task.todoDate,
       },
       fetch: {
         lead: leadView,
       },
     };
     dispatch(allTasksAction.createTaskAction(task));
   }
   dispatch(
     allContactsAction.transformContactAction({
       id: leadView,
```

```
      createDeal: data.addDeal
        ? {
            name: data.deal.name,
            price: 0,
            currency: '$',
            stage: 'make_deal',
            source: 'empty',
            leadId: leadView,
          }
        : null,
      navigate,
    })
  );
  close();
};

const minLengthName = 3;
return (
  <CreateTaskWrap onSubmit={handleSubmit(onSubmit)}>
    {' '}
    <HeadingCreatingTaskWrap>
      <h2>{t('main.transform')}</h2>
      <RxCross2 onClick={close} />
    </HeadingCreatingTaskWrap>
    <NewSevicesInputsWrap>
      {' '}
      <InputWrap style={{ width: '434px', gap: 0 }}>
        <CheckboxWrap>
          <Controller
            name="addDeal"
            control={control}
            render={({ field: { onChange, value } }) => (
              <Checkbox value={value} onChange={onChange} />
            )}
          />
          <label onClick={() => setValue('addDeal', !watch('addDeal'))}>
            {t('create.createDeal')}
          </label>
        </CheckboxWrap>
      </InputWrap>
      {watch('addDeal') && (
        <>
          <InputWrap>
            <Controller
```

```
              name="deal.name"
              rules={{
                required: t('errors.required')!,
                minLength: {
                  value: minLengthName,
                  message: t('errors.minLength', { minLengthName }),
                },
              }}
              control={control}
              render={({ field: { onChange, value } }) => (
                <InputCustom
                  placeholder={t('main.enterTitle')!}
                  onChange={onChange}
                  value={value || ''}
                  width="434px"
                  title={t('main.title')!}
                />
              )}
            />{' '}
            <ErrorMessage className="error__wrap">
                                              {errors?.task?.name   &&   <p
className="error__p">{errors.task.name.message}</p>}
            </ErrorMessage>
          </InputWrap>
        </>
      )}
      <InputWrap style={{ width: '434px', gap: 0 }}>
        <CheckboxWrap>
          <Controller
            name="addTask"
            control={control}
            render={({ field: { onChange, value } }) => (
              <Checkbox value={value} onChange={onChange} />
            )}
          />
          <label onClick={() => setValue('addTask', !watch('addTask'))}>
            {t('create.addTasks')}
          </label>
        </CheckboxWrap>
      </InputWrap>
      {watch('addTask') && (
        <>
          <InputWrap>
            <Controller
```

```
              name="task.name"
              rules={{
                required: t('errors.required')!,
                minLength: {
                  value: minLengthName,
                  message: t('errors.minLength', { minLengthName }),
                },
              }}
              control={control}
              render={({ field: { onChange, value } }) => (
                <InputCustom
                  placeholder={t('main.enterTitle')!}
                  onChange={onChange}
                  value={value || ''}
                  width="434px"
                  title={t('main.name')!}
                />
              )}
            />{' '}
            <ErrorMessage className="error__wrap">
                                          {errors?.task?.name   &&   <p
className="error__p">{errors.task.name.message}</p>}
            </ErrorMessage>
          </InputWrap>
          <InputWrap>
            <Controller
              name="task.todoDate"
              rules={{
                required: t('errors.required')!,
              }}
              control={control}
              render={({ field: { onChange, value } }) => (
                <InputCustom
                  placeholder={t('main.typeEmployeeName')!}
                  onChange={onChange}
                  value={value || ''}
                  width="434px"
                  title={t('main.completeDate')!}
                  type="datetime-local"
                />
              )}
            />{' '}
            <ErrorMessage>
              {errors?.task?.todoDate && (
```

```
                                                                    <p
className="error__p">{errors.task.todoDate.message}</p>
                )}
              </ErrorMessage>
            </InputWrap>
          </>
        )}
      </NewSevicesInputsWrap>
      <BottomButtonWrapModal>
        <button onClick={close} type="button">
          {t('main.abolition')}
        </button>
        <button type="submit">{t('main.transform')}</button>
      </BottomButtonWrapModal>
    </CreateTaskWrap>
  );
};
```

Додаток Б. Код серверної частини

Обробник коментарів:

```
import { PipelineStage } from 'mongoose';
import { LookupPipeline } from 'src/common/pipelines/lookup.pipeline';
import { Comment } from 'src/models/Comment';

export class CommentPipeline {
 private static lookupPipeline = new LookupPipeline<Comment>();

 static lookupOwner(): PipelineStage[] {
   return [
     ...this.lookupPipeline.lookupField({
       collection: 'User',
       localField: 'owner',
     }),
     {
       $addFields: {
         owner: { $concat: ['$owner.name', ' ', '$owner.surname'] },
       },
     },
   ];
 }

 static lookupLead(): PipelineStage[] {
   return [
     ...this.lookupPipeline.lookupField({
       collection: 'Lead',
       localField: 'lead',
     }),
     {
       $addFields: { lead: { $concat: ['$lead.name', ' ', '$lead.surname']
} },
     },
   ];
 }
}

import { Injectable } from '@nestjs/common';
import { Model, Types } from 'mongoose';
import { PostCommentDTO } from './dto/PostCommentDTO';
import { Comment, CommentDocument } from 'src/models/Comment';
import { InjectModel } from '@nestjs/mongoose';
import { CommentPipeline } from './comment.pipeline';
```

```
import { OnEvent } from '@nestjs/event-emitter';
import { CommentTemplateEnum } from
'src/comment/enums/CommentTemplateEnum';
import { PostSysComment } from './dto/PostSysCommentDTO';
import { ThreadPipeline } from 'src/common/pipelines/if_then.pipeline';

@Injectable()
export class CommentService {
 constructor(
   @InjectModel(Comment.name)
   private readonly commentModel: Model<CommentDocument>,
 ) {}

 async create(comment: PostCommentDTO) {
   comment.sysHeader = CommentTemplateEnum.TEXT;

   return new this.commentModel(comment).save();
 }

 async list(lead: Types.ObjectId) {
   const comments = await this.commentModel.aggregate<Comment>([
     { $match: { lead } },
     ...ThreadPipeline(
       [
         {
           $match: {
             sysHeader: {
               $regex: '%_OWNER',
             },
           },
         },
         ...CommentPipeline.lookupOwner(),
         {
           $project: {
             lead: 0,
           },
         },
       ],
       [
         {
           $match: {
             sysHeader: {
               $regex: '%_LEAD',
             },
```

```
          },
        },
        ...CommentPipeline.lookupLead(),
        {
          $project: {
            owner: 0,
          },
        },
      ],
    ),
  ]);
  return comments.map((e) => {
    const { sysHeader: sys, text, createdAt, ...other } = e;
    const sysHeader = sys
      .replace('%_LEAD', other.lead as any)
      .replace('%_OWNER', other.owner as any);
    return {
      sysHeader,
      text,
      createdAt,
    };
  });
}

@OnEvent('comment')
handleOrderCreatedEvent({
  template,
  owner,
  lead,
  args,
}: PostSysComment<'CREATE_TASK'>) {
  let sysHeader = CommentTemplateEnum[template] as string;
  if (args) {
    Object.keys(args).forEach(
      (arg) =>
                (sysHeader  =  sysHeader.replace(`%_${arg.toUpperCase()}`,
args[arg])),
    );
  }

  new this.commentModel({
    sysHeader,
    owner,
    lead,
```

```
    }).save();
  }
}
import { PipelineStage } from 'mongoose';
export const ThreadPipelineWithOptions = (
  { groupBy = '_id' }: { groupBy?: any },
  ...threads: PipelineStage[][]
): PipelineStage[] => [
  {
    $facet: threads.reduce(
      (prev, cur, index) => ({ ...prev, [`thread_${index}`]: cur }),
      {},
    ),
  },
  {
    $project: {
      list: {
        $concatArrays: threads.map((_, index) => `$thread_${index}`),
      },
    },
  },
  {
    $unwind: {
      path: '$list',
    },
  },
  {
    $group: {
      _id: `$list.${groupBy}`,
      list: {
        $mergeObjects: '$list',
      },
    },
  },
  {
    $replaceRoot: {
      newRoot: '$list',
    },
  },
];

export const ThreadPipeline = (
  ...threads: PipelineStage[][]
): PipelineStage[] => ThreadPipelineWithOptions({}, ...threads);
```