**Міністерство освіти і науки України**
**Національний технічний університет**
**«Дніпровська політехніка»**

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

**Кафедра** Програмного забезпечення комп'ютерних систем
(повна назва)

# ПОЯСНЮВАЛЬНА ЗАПИСКА
## кваліфікаційної роботи ступеня

*бакалавра*
(назва освітньо-кваліфікаційного рівня)

**студента** *Пучка Владислава Сергійовича*
(ПІБ)

**академічної групи** *121-20-1*
(шифр)

**спеціальності** *121 Інженерія програмного забезпечення*
(код і назва спеціальності)

**освітньої програми** *Інженерія програмного забезпечення*
(назва освітньої програми)

**на тему:** *Розробка CRM-системи для керування клієнтської та продуктової бази компанії з використанням React та Node.js*

| **Керівники** | **Прізвище, ініціали** | **Оцінка за шкалою** | | **Підпис** |
|---|---|---|---|---|
| | | **рейтинговою** | **Інституційною** | |
| кваліфікаційної роботи | *доц. Спірінцев В.В.* | | | |
| **розділів:** | | | | |
| спеціальний | *доц. Спірінцев В.В.* | | | |
| економічний | *доц. Касьяненко Л.В.* | | | |
| | | | | |
| | | | | |
| **Рецензент** | | | | |
| **Нормоконтролер** | *доц. Мартиненко А.А.* | | | |

**Дніпро**
**2024**

**Міністерство освіти і науки України**
**НТУ «Дніпровська політехніка»**

**ЗАТВЕРДЖЕНО:**
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)

_____ М.О. Алексєєв
(підпис) (прізвище, ініціали)

« » _____ 2024 року

# ЗАВДАННЯ
## на кваліфікаційну роботу

*бакалавра*
(назва освітньо-кваліфікаційного рівня)

**студента** *121-20-1* *Пучка Владислава Сергійовича*
(група) (прізвище та ініціали)

**тема кваліфікаційної роботи** *Розробка CRM-системи для керування клієнтської та продуктової бази компанії з використанням React та Node.js*

затверджена наказом ректора НТУ «ДП» від *23.05.2023* № *469-с*

| Розділ | Зміст виконання | Термін виконання |
|---|---|---|
| ***Спеціальний*** | ***На основі матеріалів проєктно-технологічної та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі. Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми*** | ***14.06.2024 р.*** |
| ***Економічний*** | ***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки*** | ***20.06.2024 р.*** |

Завдання видав _____ *доц. Спірінцев В.В.*
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ *Пучко В.С.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2024 р.*

Термін подання кваліфікаційної роботи до ЕК: *24.06.2024 р.*

# РЕФЕРАТ

Пояснювальна записка: с., 49 рис., 3 дод., 15 джерел.

Об'єкт розробки: CRM-система для керування клієнтської та продуктової бази компанії з використанням React, Node.js

Мета кваліфікаційної роботи: розробка CRM-системи для керування клієнтської та продуктової бази компанії з використанням React та Node.js, що дозволить компанії управляти взаємодією з клієнтами, збирати та аналізувати інформацію про них.

У вступній частині кваліфікаційної роботи висвітлено сучасний стан проблеми, вказана мета дослідження та галузь її застосування, обґрунтовується актуальність теми, конкретизується постановка завдання кваліфікаційної роботи

У першому розділі кваліфікаційної роботи вказуються загальні відомості з предметної області, визначається галузь застосування та призначення розробки, формулюється постановка завдання, детально описуються вимоги до розробленої програми

У другому розділі кваліфікаційної роботи описується етап проектування та розробки програмного засобу. У ньому висвітлюється функціональне призначення програми, архітектура та шаблони проектування, визначаються вхідні та вихідні дані, проводиться опис структури та алгоритмів функціонування програмного засобу, а також інтерфейс, виклик та його завантаження.

В економічному розділі визначається трудомісткість розробленого продукту, проводяться підрахунки вартості роботи для створення програми та розраховується час на його створення.

Практичне значення полягає у розробці веб-додатку, який дозволить керувати клієнтською та продуктовою базою компанії, автоматизує рутинні задачі та має інтуїтивно зрозумілий інтерфейс для швидкого освоєння користувачами

Актуальність розробленої інформаційної системи є значною, бо все більше і більше комерційних компаній переходить на електронні засоби ведення бізнесу

Список ключових слів: ІНФОРМАЦІЙНА СИСТЕМА, ЕЛЕКТРОННА КОМЕРЦІЯ, CRM-СИСТЕМА, БАЗА ДАНИХ, REACT, JAVASCRIPT, NODE.JS, EXPRESS.JS, MONGOOSE, MONGODB, HTML, CSS, NOSQL, FETCH API.

## ABSTRACT

Explanatory note: p., 49 figures, 3 appendices, 15 sources.

Object of development: CRM system for managing the company's client and product base using React, Node.js

The purpose of the qualification work: development of a CRM system for managing the company's customer and product base using React and Node.js, which will allow the company to manage interaction with customers, collect and analyze information about them.

In the introductory part of the qualification work, the current state of the problem is highlighted, the purpose of the research and the field of its application are indicated, the relevance of the topic is substantiated, and the task of the qualification work is specified.

In the first section of the qualification work, general information on the subject area is indicated, the field of application and purpose of the development is determined, the task statement is formulated, and the requirements for the developed program are described in detail

The second section of the qualification work describes the design and development stage of the software tool. It highlights the functional purpose of the program, architecture and design patterns, defines input and output data, describes the structure and algorithms of the software tool, as well as the interface, call and its download.

In the economic section, the labor intensity of the developed product is determined, the cost of work to create the program is calculated, and the time for its creation is calculated.

The practical value is in the development of a web application that will allow you to manage the customer and product base of the company, automate routine tasks and have an intuitive interface for quick learning by users

The relevance of the developed information system is significant, because more and more commercial companies are switching to electronic means of conducting business

List of keywords: INFORMATION SYSTEM, ELECTRONIC COMMERCE, CRM SYSTEM, DATABASE, REACT, JAVASCRIPT, NODE.JS, EXPRESS.JS, MONGOOSE, MONGODB, HTML, CSS, NOSQL, FETCH API.

**ЗМІСТ**

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ПЗ - Програмне забезпечення

ІС - Інформаційна система

ОС - Операційна система

ПК - Персональний комп’ютер

БД - База даних

СУБД - Система управління базами даних

HTML - Hyper Text Markup Language

CSS - Cascading Style Sheets

NoSQL – Not only Structured Query Language

JS – JavaScript

API – Application Programming Interface

# ВСТУП

У сучасному світі бізнесу, який характеризується високою конкуренцією та швидкими темпами розвитку технологій, ефективне управління відносинами з клієнтами є ключовим фактором успіху. Одним з найбільш дієвих інструментів у цій сфері є системи управління взаємовідносинами з клієнтами (CRM-системи). Вступна частина даної кваліфікаційної роботи присвячена аналізу сучасного стану проблеми, визначенню мети дослідження та галузі його застосування, обґрунтуванню актуальності теми, а також конкретизації постановки завдання кваліфікаційної роботи.

CRM-системи являють собою комплекс програмних рішень, спрямованих на автоматизацію та оптимізацію взаємодії з клієнтами. Вони дозволяють підприємствам ефективніше управляти продажами, маркетингом, сервісним обслуговуванням та підтримкою клієнтів. Сучасні CRM-системи пропонують широкий спектр функцій, включаючи управління контактами, аналіз поведінки клієнтів, прогнозування продажів та інші аналітичні інструменти.

Популярність CRM-систем зростає, оскільки компанії прагнуть покращити якість обслуговування клієнтів та підвищити свою конкурентоспроможність. За даними останніх досліджень, використання CRM-систем дозволяє збільшити рівень задоволеності клієнтів та підвищити продажі на 20-30%.

Метою даного дослідження є аналіз ефективності впровадження CRM-систем у різних галузях бізнесу, а також визначення основних факторів, що впливають на успішність їх використання. Дослідження охоплює різні аспекти впровадження CRM-систем, включаючи технічні, організаційні та економічні аспекти.

Практичне значення полягає у розробці веб-додатку, який дозволить керувати клієнтською та продуктовою базою компанії, автоматизує рутинні задачі та має інтуїтивно зрозумілий інтерфейс для швидкого освоєння користувачами

CRM-системи знаходять своє застосування в різних галузях, таких як роздрібна торгівля, фінансові послуги, телекомунікації, охорона здоров'я та інші. Кожна галузь має свої специфічні вимоги та особливості, що впливають на вибір та налаштування CRM-рішень.

Актуальність теми впровадження CRM-систем обумовлена зростаючою потребою підприємств у підвищенні ефективності взаємодії з клієнтами. У сучасних умовах глобалізації та діджиталізації бізнесу, компанії змушені шукати нові шляхи для утримання клієнтів та залучення нових. CRM-системи пропонують дієві інструменти для вирішення цих задач, забезпечуючи комплексний підхід до управління клієнтськими відносинами.

Основними завданнями даної кваліфікаційної роботи є:

− Проведення огляду та аналізу існуючих CRM-систем.

− Визначення основних переваг та недоліків використання CRM-систем у різних галузях бізнесу.

− Аналіз факторів, що впливають на успішність впровадження CRM-систем.

− Розробка CRM-системи, що задовільняє умови для успішного ведення бізнесу.

# РОЗДІЛ 1
# АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

## 1.1. Загальні відомості з предметної галузі

Велика частина успіху бізнесу залежить від оптимізації внутрішніх процесів. Швидка обробка клієнтських заявок, швидке ухвалення управлінських рішень і ефективна реакція на складні запити можуть значно підвищити прибуток підприємства. Одним з ефективних засобів для досягнення цих цілей є використання CRM-систем. Вони дозволяють автоматизувати бізнес-процеси, спрощуючи роботу з клієнтами та управління компанією.

CRM система (Customer Relationship Management) – це програмне забезпечення, завдяки якому різні компанії можуть управляти взаємодією з клієнтами, збирати та аналізувати інформацію про них. Використання такої програми великою мірою впливає на покращення відносин з клієнтами, наслідком якого є підвищення ефективності бізнесу. Основна мета CRM - це забезпечити компанії можливість краще розуміти та задовольняти потреби своїх клієнтів.

Однією з найбільших переваг CRM системи є гнучка організація збереження даних клієнтів. Всю інформацію про клієнта можна структурувати зручним способом, що дозволяє персоналу легко її знаходити та робити певні висновки. Нижче наведено декілька пунктів, які містять приклади даних, на основі яких можна виконати структуризацію:

– Історія взаємодій**:** CRM система зберігає історію взаємодій з кожним клієнтом, такі як дзвінки, електронні листи, зустрічі, покупки тощо. Це дозволяє персоналу мати повний образ клієнта і його взаємодій з компанією, що в свою чергу допомагає у підготовці до подальших взаємодій та наданні більш індивідуалізованого обслуговування.

– Продуктові вподобання**:** CRM система може також зберігати інформацію про продукти чи послуги, якими цікавиться клієнт, або які він вже

придбав. Це допомагає компаніям аналізувати потреби своїх клієнтів та розробляти індивідуальні пропозиції.

– Статус клієнта**:** CRM система дозволяє встановлювати статус клієнта, наприклад, потенційний, новий, успішний тощо. Це допомагає управлінцям визначати пріоритети та розробляти стратегії взаємодії з різними категоріями клієнтів.

Ще однією перевагою CRM системи є автоматизація повсякденних робочих процесів. Вона допомагає виконувати багато рутинних задач, таких як відправлення електронних листів, нагадування дзвінків, аналіз вхідної інформації тощо, що допомагає підвищити ефективність роботи персоналу.

Єдиним мінусом, з яким можна зіткнутися, це навчання співробітників роботі в CRM-системах та оплата за їх застосування. У малому та середньому бізнесі можна обійтися безкоштовною версією системи управління, тоді як для великого та середнього бізнесу вже потрібні досконаліші версії цих систем.

Аналіз готових існуючих рішень. Для аналізу була обрана CRM система KeyCRM, користувач – компанія з продажу та доставки палива.

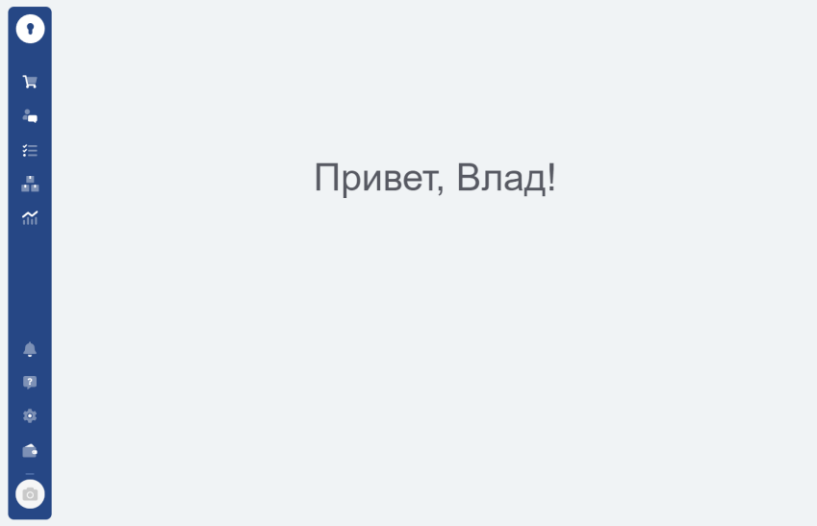


Рис. 1.1. Головна сторінка

На головній сторінці можна побачити привітальний напис та блок меню зліва. Меню складається з таких елементів: «Список лідів (карток клієнтів)», «Чати», «Завдання», «Список товарів», «Аналітика» та блок налаштування.

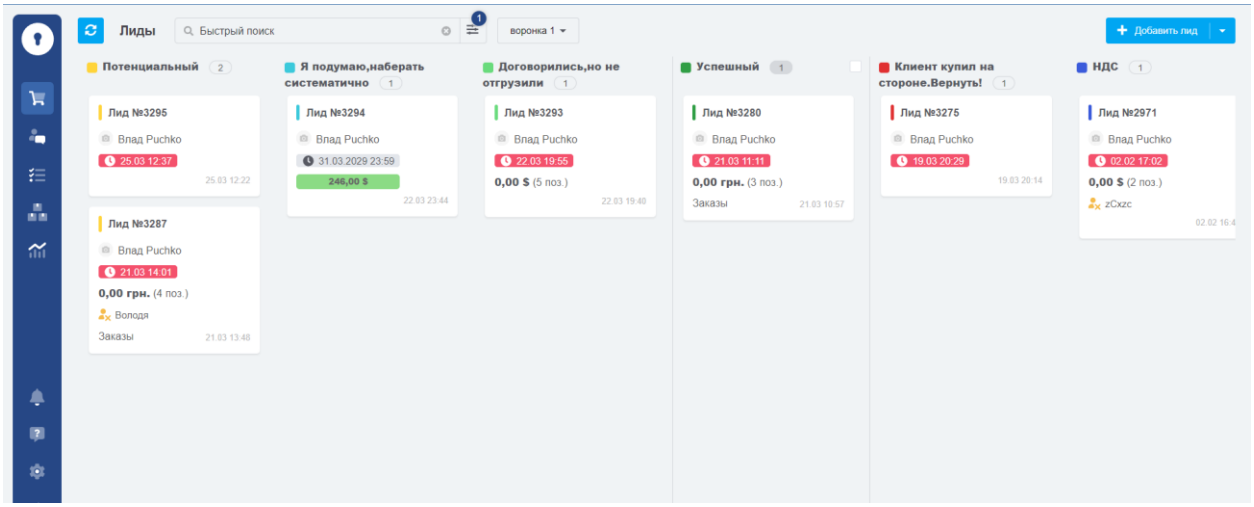


Рис 1.2. Сторінка «Список лідів»

На сторінці «Список лідів» можна побачити поле для пошуку, кнопку що відкриває блок фільтрів (рис. 1.3), кнопку додавання нового ліда та структрне відображення карток клієнтів за певними статусами.

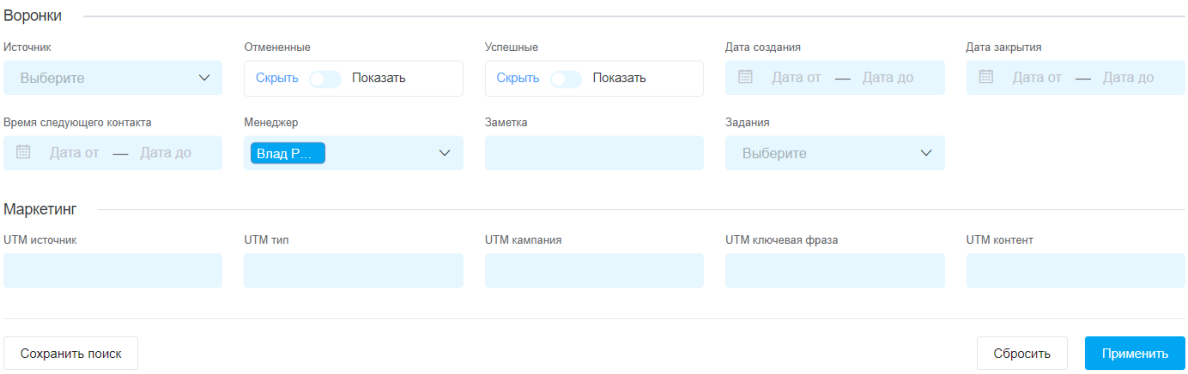


Рис 1.3. Блок фільтрів

На рис. 1.4 можна побачити розкритий лід. На ній можна побачити інформацію про клієнта, інформацію про лід, товари та послуги, що зацікавили клієнта, список оплат та блок коментарів.

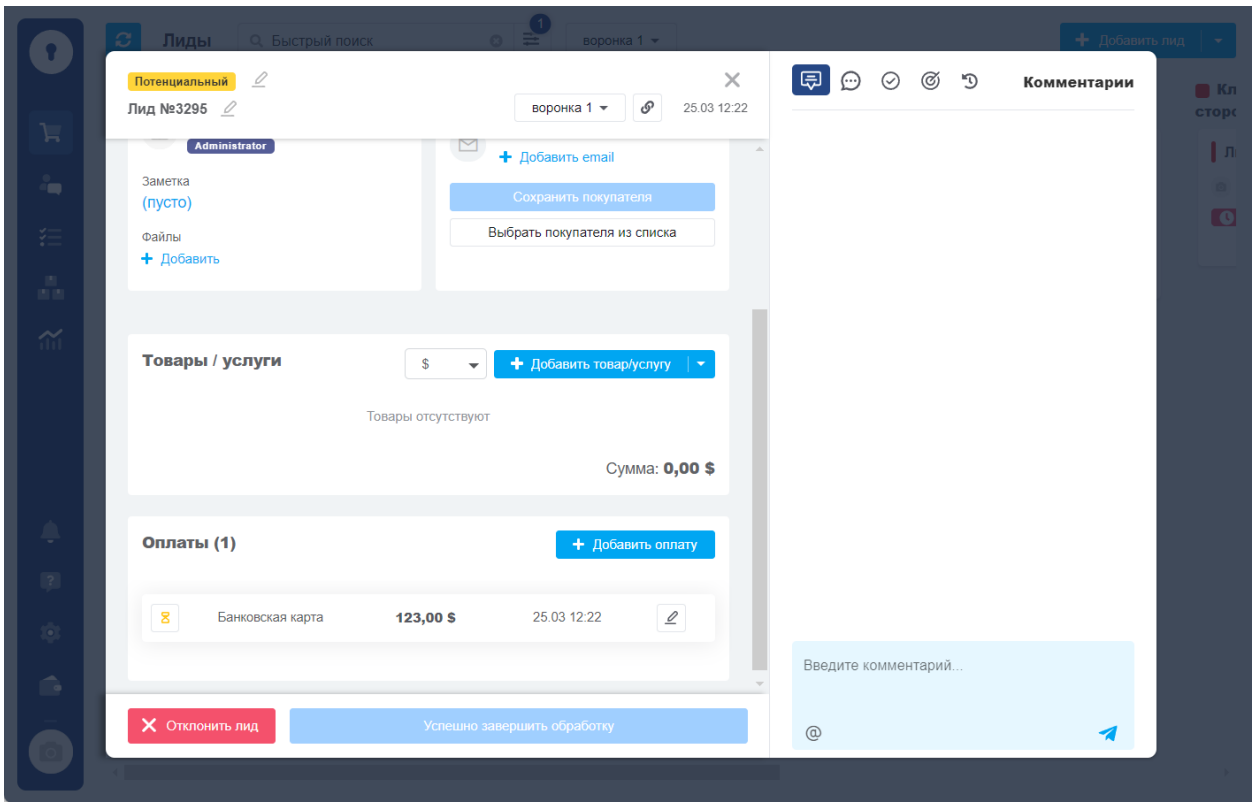


Рис 1.4. Розкритий лід

На рис. 1.5 можна побачити аналітичний розділ.

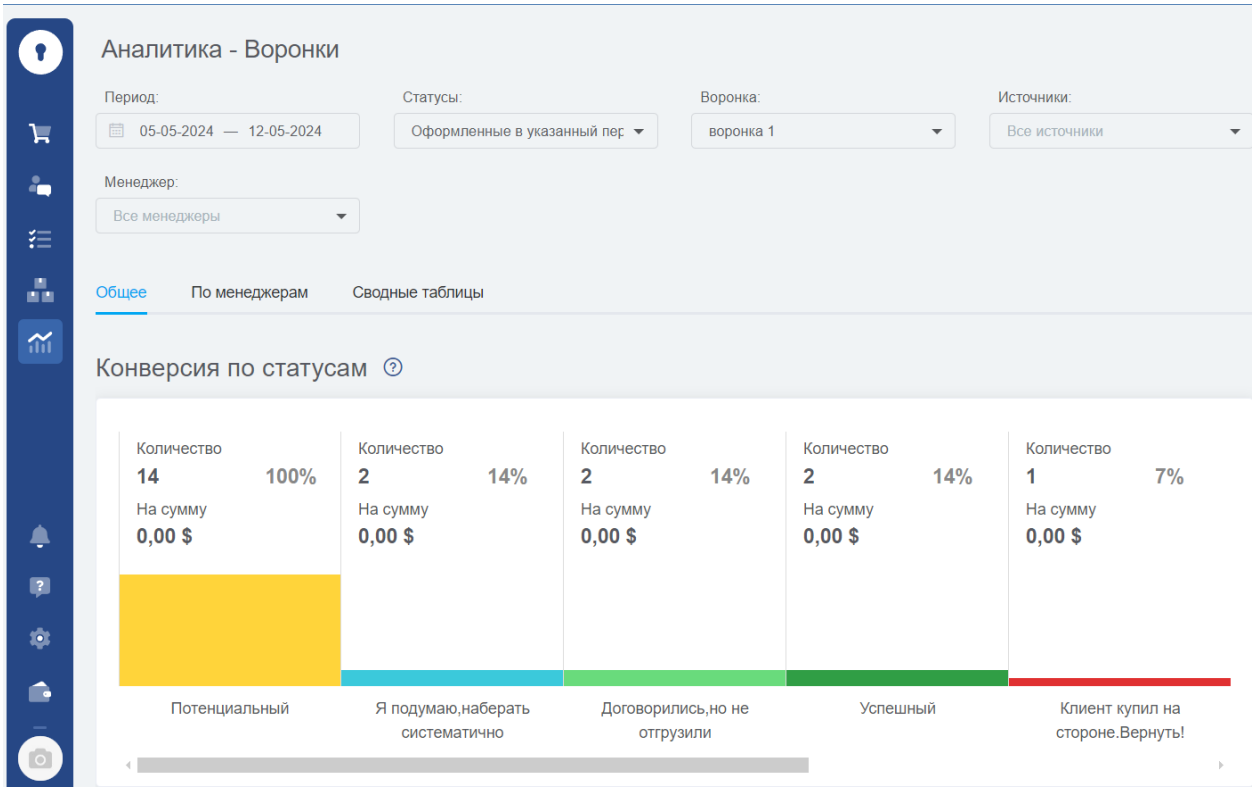


Рис 1.5. Аналітичний розділ

У процесі аналізу KeyCRM я дійшов до висновків, що дана CRM система є доволі зручна у використанні. Вона надає гнучку структуризацію клієнтів, товарів та докладний аналітичний розділ. Проте були виявлені певні недоліки. Основним недоліком є недопрацьований пошук. Завдяки ньому можна лише відсортувати ліди певного менеджера, не має можливості перевірити за номером клієнта, чи є він вже у спільній базі даних. Також немає можливості продублювати товар у картці клієнта, потрібно або видаляти минулий, або створювати нову картку. Ще, при створенні нового ліда, якщо введений номер вже існує у базі, немає попередження про це. Всі ці недоліки мають бути обов'язково виправлені у процесі розробки даної кваліфікаційної роботи.

### 1.2. Призначення розробки та галузь застосування

У межах кваліфікаційної роботи буде створена CRM-система для керування клієнтської та продуктової бази компанії з використанням React, Node.js. Система застосовується у комерційній діяльності.

Необхідність розробки CRM-системи виникла внаслідок кількох ключових причин. По-перше, зростання конкуренції в сучасному бізнес-середовищі, що вимагає ефективного управління взаємовідносинами з клієнтами для утримання та залучення нових клієнтів. По-друге, підвищення очікувань клієнтів, які очікують високого рівня обслуговування та персоналізованого підходу, що потребує використання комплексних інструментів для управління клієнтськими даними. По-третє, необхідність оптимізації бізнес-процесів, оскільки автоматизація продажів, маркетингу та сервісного обслуговування дозволяє знизити витрати та підвищити ефективність роботи компанії. Нарешті, потреба в аналітиці та прогнозуванні, адже CRM-системи надають можливість збирання та аналізу великої кількості даних про клієнтів, що дозволяє приймати обґрунтовані рішення та прогнозувати майбутні тенденції.

CRM-системи знаходять своє застосування в різних галузях бізнесу. У роздрібній торгівлі вони допомагають у управлінні взаємовідносинами з клієнтами, аналізі покупок, створенні персоналізованих пропозицій та програм лояльності. У фінансових послугах CRM використовується для управління контактами з клієнтами, обробки заявок на кредити, аналітики ризиків та управління інвестиціями. У телекомунікаціях системи CRM допомагають в управлінні абонентською базою, підтримці клієнтів, аналізі використання послуг та розробці нових тарифних планів. В охороні здоров'я CRM сприяє управлінню записами пацієнтів, обробці звернень, плануванню візитів та аналізу медичних даних. У виробництві такі системи допомагають в управлінні замовленнями, постачаннями та підтримці клієнтів, аналізі попиту на продукцію та прогнозуванні виробничих потреб. В освіті CRM використовується для управління взаємовідносинами зі студентами та викладачами, обробки заявок на вступ, підтримки студентів та аналізу академічних успіхів. CRM-системи забезпечують комплексний підхід до управління взаємовідносинами з клієнтами, що дозволяє компаніям підвищити свою ефективність, задоволеність клієнтів та конкурентоспроможність на ринку.

### 1.3. Підстава для розробки

Підставами для розробки та виконання кваліфікаційної роботи є:

− освітня програма 121 «Інженерія програмного забезпечення»;

− навчальний план та графік навчального процесу;

− наказ ректора Національного технічного університету «Дніпровська політехніка» № 469-с від 23.05.2024 р;

− завдання на кваліфікаційну роботу на тему «Розробка CRM-системи для керування клієнтської та продуктової бази компанії з використанням React, Node.js.».

### 1.4. Постановка завдання

#### 1.4.1. Постановка завдання до розробки системи

В даній кваліфікаційній роботі розглядається створення CRM-системи для керування клієнтської та продуктової бази на прикладі компанії, що займається продажом та доставкою палива.

Мета роботи: Розробка CRM-системи для керування клієнтської та продуктової бази компанії з використанням React, Node.js.

CRM система створюється для організації збереження даних клієнтів та товарів компанії. Аудиторією є робітники компанії.

Додаток має задовільняти такі вимоги: зручний інтерфейс, висока швидкість роботи, адаптація під екрани різного розміру, оптимізація загального пошуку

#### 1.4.2. Опис інтерфейсу користувача

Інтерфейс користувача ролі «адміністратор»:

− Вітальна сторінка: привітальний надпис з ім’ям користувача.

− Ліди: сторінка зі списком карток клієнтів. Тут можна переглядати, створювати, редагувати, видаляти картки. Також на цій сторінці знаходиться блок фільтрів та пошук по загальній клієнтській базі. У адміністратора на цій сторінці відображені його власні ліди та ліди його менеджерів.

− Список менеджерів: сторінка для створення, редагування та перегляду списку власних менеджерів. Також тут можна призначати та переглядати персональні завдання кожному менеджеру.

− Покупці: сторінка для створення, редагування та перегляду списку постійних покупців.

− Товари: сторінка для створення, редагування та перегляду списку товарів.

− Аналітика: сторінка, що містить діаграму з інформацією щодо «закритих» лідів.

− Видалені: сторінка зі списком видалених карток клієнтів зі сторінки «Ліди». Тут їх можна переглядати та видалити остаточно.

Інтерфейс користувача ролі «менеджер» майже однаковий. Нижче наведені відмінності від інтерфейсу адміністратора:

− Ліди: у менеджера відображаються лише його власні картки.

− Завдання: сторінка, де менеджер може переглянути завдання, призначені адміністратором та створити собі власні.

## 1.5. Вимоги до програми або програмного виробу

### 1.5.1. Вимоги до функціональних характеристик

Для досягнення поставленої мети при розробці CRM системи, мають бути реалізовані такі елементи:

− зручний інтерфейс користувача;

− ролі менеджера, адміністратора та їхня взаємодія;

− пошук клієнтів загальною базою по номеру телефону;

− фільтри для сортування карток клієнтів;

− зручна зміна статусу картки клієнта за допомогою «перетягування» її з однієї колонки в іншу.

### 1.5.2. Вимоги до інформаційної безпеки

При розробці CRM системи, особливу увагу слід приділити заходам забезпечення інформаційної безпеки, оскільки ці системи зберігають великий обсяг конфіденційної інформації про клієнтів та ділові взаємодії. Ось деякі основні вимоги до інформаційної безпеки при розробці CRM систем:

− Аутентифікація та авторизація: Забезпечення надійної системи аутентифікації користувачів, щоб гарантувати, що тільки правомірні користувачі певної ролі мають доступ до системи.

− Шифрування даних: Всі дані, що передаються між користувачем та сервером, а також дані, збережені на сервері, повинні бути зашифровані, щоб запобігти несанкціонованому доступу до них.

− Резервне копіювання даних: Регулярне створення резервних копій даних та забезпечення їх захисту від несанкціонованого доступу, щоб у випадку втрати або пошкодження даних була можливість їх відновлення.

### 1.5.3. Вимоги до складу та параметрів технічних засобів

Нижче наведений склад та параметри технічних засобів для повноцінного функціонування програми:

− пристрої вводу та виводу;

− стабільний доступ до мережі Інтернет;

− веб-браузер (Google Chrome, Safari, Microsoft Edge та ін.) з підтримкою JS;

− операційна система Windows, Linux, MacOS, Android, IOS;

− оперативна пам’ять: 4 ГБ;

− процесор: 2 ядра із частотою 2.2 ГГц.

### 1.5.4. Вимоги до інформаційної та програмної сумісності

Для створення CRM системи на базі React мають задовільнятись такі вимоги:

− Операційна система: Windows XP або вище.

− ОЗУ: 4 ГБ.

− Обсяг пам’яті на жорсткому диску: ≈ 8 – 10 ГБ.

− Веб-браузер.

− Доступ до мережі Інтернет.

− Редактор коду.

− Node.js (express.js, mongoose).

Інтерфейс користувача CRM системи має бути написаний за допомогою JavaScript бібліотеки React, серверна частина має бути реалізована за допомогою express.js - фреймворку web-додатків для Node.js. Дані, що будуть введені до CRM системи, мають зберігатися у NoSQL базі даних, СУБД MongoDB.

# РОЗДІЛ 2
# ПРОЕКТУВАННЯ ТА РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

## 2.1. Функціональне призначення системи

Веб-орієнтований додаток для CRM системи компанії, що займається продажом та доставкою палива, призначений для підтримки та покращення процесів електронної комерції. Тому функціональне призначення системи включає:

− збереження інформації про клієнтів та товари компанії у базі даних;

− додавання, редагування, видалення клієнтів та товарів;

− відображення даних клієнта у вигляді зручних карток та розміщення їх у колонках з відповідним статусом;

− авторизація та автентифікація користувачів;

– пошук клієнтів загальною базою за номером телефону (можна ввести як цілий номер телефону, так і частину);

– блок фільтрів, який містить фільтри за певними критеріями та поле для пошуку «по своїх клієнтах»;

– аналіз даних для прийняття обґрунтованих бізнес-рішень;

– розподілення ролей користувачів;

Експлуатаційне призначення інформаційної системи включає:

– об'єднання всіх даних про клієнтів в одному місці для зручного доступу та управління;

– інтуїтивно зрозумілий інтерфейс для простоти навігації та використання;

– автоматизація відображення клієнтів, яких потрібно опрацювати «сьогодні»;

– підтримка мобільних пристроїв для доступу до CRM «на ходу».

## 2.2. Опис застосованих математичних методів

Під час проектування та розробки веб-додатка не використовувалися складні математичні методи або алгоритми. Розрахунки та обчислення, які використовувалися, обмежувалися простими арифметичними діями.

## 2.3. Опис використаної архітектури та шаблонів проектування

Для розробки CRM системи була обрана клієнт-серверна архітектура з використанням React для клієнтської частини (front-end) та Node.js для серверної частини (back-end). Такий підхід забезпечує масштабованість, дозволяючи легко додавати нових користувачів та функціональність без значного навантаження на систему. Централізація даних на сервері сприяє покращенню управління даними та їх безпеці. Модульність цієї архітектури дозволяє незалежно оновлювати та розвивати клієнтську та серверну частини без впливу на інші компоненти системи.

Клієнтська частина, реалізована за допомогою React, відповідає за інтерфейс користувача, відображення даних та взаємодію з користувачем через браузер. Серверна частина, створена на базі Node.js, обробляє запити від клієнтів, виконує бізнес-логіку, взаємодіє з базою даних, обробляє та повертає результати клієнтам.

Для побудови клієнтської частини CRM системи використовується шаблон проектування MVC (Model-View-Controller). Це рішення обґрунтоване зручністю підтримки та розширення, оскільки зміни до окремих компонентів системи можна вносити без впливу на інші. Розподіл обов'язків між компонентами (Model, View, Controller) сприяє кращій організації коду. Також, використання MVC полегшує тестування окремих компонентів системи, що підвищує якість програмного забезпечення.

У рамках MVC:

– Model відповідає за бізнес-логіку та роботу з даними.

– View відповідає за відображення інформації користувачу через React компоненти.

– Controller обробляє вхідні запити від користувача, керує взаємодією між Model та View.

Для забезпечення єдиного екземпляра об'єкта, наприклад, для управління підключенням до бази даних або глобальних налаштувань CRM системи, використовується шаблон "Одинак" (Singleton). Цей шаблон гарантує наявність єдиного екземпляра класу і забезпечує централізований доступ до загальних ресурсів, таких як налаштування конфігурації або логування.

Шаблон "Фабричний метод" (Factory Method) застосовується для створення об'єктів різних типів без вказування конкретних класів об'єктів. Це дозволяє легко додавати нові типи об'єктів у систему. Використання цього шаблону забезпечує гнучкість, полегшуючи додавання нових типів об'єктів та зміну логіки їх створення. Інкапсуляція деталей створення об'єктів спрощує підтримку та розвиток системи.

Для розробки CRM системи обрана ітеративно-інкрементальна модель. Цей підхід забезпечує гнучкість до змін, дозволяючи швидко реагувати на зміну вимог та умов ринку. Регулярні ітерації дозволяють отримувати постійний зворотний зв'язок від користувачів та замовників, що сприяє підвищенню якості системи. Кожна ітерація включає фази аналізу, проектування, реалізації та тестування, що забезпечує високий рівень контролю якості.

Процес розробки включає кілька етапів. Спочатку здійснюється аналіз вимог, потім проектування архітектури та дизайну системи. Після цього відбувається реалізація, де написання коду здійснюється відповідно до вимог та проектування. Наступним кроком є тестування, де перевіряється код та його відповідність вимогам. Заключний етап - це оцінка та інтеграція, де оцінюються результати ітерації та інтеграція з попередніми версіями системи.

Обрана клієнт-серверна архітектура з використанням React та Node.js забезпечує високу масштабованість, модульність та зручність підтримки CRM

системи. Використання шаблону MVC для клієнтської частини допомагає чітко розділити відповідальність між компонентами системи, полегшуючи їх підтримку та розвиток. Застосування шаблонів проектування "Одинак" та "Фабричний метод" дозволяє ефективно управляти ресурсами та створенням об'єктів. Ітеративно-інкрементальна модель життєвого циклу програмного забезпечення забезпечує гнучкість та високий рівень контролю якості, що є критично важливим для успішної реалізації CRM системи.

### 2.4. Опис використаних технологій та мов програмування

Для створення веб-додатку CRM-системи були обрані:

Мова програмування:

– “front-end” частина: JavaScript;

– “back-end” частина: JavaScript.

База даних:

– NoSQL database.

Список обраних технологій:

– бібліотека React;

– база даних MongoDB;

– фреймворк web-додатків для Node.js - Express;

– Mongoose;

– Fetch API.

Огляд інструментів, обраних для розробки CRM-системи

JavaScript. JavaScript – це високорівнева, прототипна, інтерпретована мова програмування з динамічною типізацією, яка є однією з основних технологій веб-розробки, поряд з HTML та CSS. Вона була розроблена Бренданом Айком у 1995 році за 10 днів. Спочатку мова називалася Mocha, потім її перейменували на LiveScript, і врешті-решт вона стала відомою як JavaScript. Назва JavaScript була обрана з маркетингових міркувань, оскільки в той час Java була популярною мовою програмування, але JavaScript і Java

мають мало спільного. Автор серії книг «Ви не знаєте JavaScript» виразився про це так: «JavaScript з Java спільного не більше, ніж у луна-парку з Місяцем [1].»

Динамічна типізація – це одна з ключових особливостей мови програмування JavaScript. Цей термін означає, що значення в межах програми можуть змінювати свій тип. Тобто, тип змінної визначається не під час компіляції, а у процесі виконання програми. Наприклад, числове значення за необхідності перетворюється на рядкове або навпаки.

Прототипною цю мову називають тому, що під час створення коду на JavaScript періодично створюються об’єкти, які називаються «прототипами». У JavaScript майже все є об'єктом, і кожен об'єкт має посилання на інший об'єкт, який є його "прототипом". Якщо об'єкт запитує властивість, а вона відсутня, JavaScript автоматично здійснить його пошук у прототипі. Цей процес може продовжуватися з ланцюжком прототипів до тих пір, поки функція не буде знайдена або не буде досягнутий кінець ланцюжка

Ще однією важливою особливістю JavaScript є підтримка асинхронного програмування. Воно реалізується через механізми колбеків, промісів (Promises) та асинхронних функцій (async/await). Це дозволяє обробляти операції, які займають тривалий час, такі як завантаження даних з сервера, без блокування основного потоку виконання.

JavaScript – одна з найважливіших і найпопулярніших мов програмування у світі. Його універсальність, простота використання, потужні функції та підтримка великої спільноти зробили його незамінним інструментом для розробників. Згідно зі статистикою «Рейтинг мов програмування 2023», яка була приведена за 2022 рік [2], JavaScript посідає перше місце, як можна побачити на рис. 2.1.

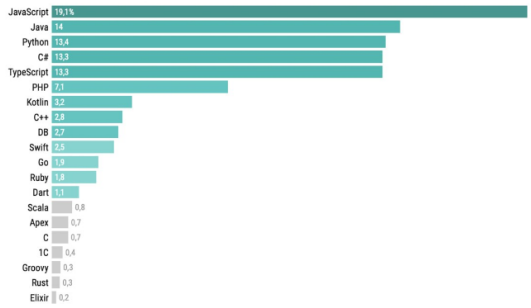


Рис. 2.1. Статистика популярних мов програмування за 2023 р.

До появи цієї мови веб-сторінки в інтернеті були статичними, нагадували звичайні електронні документи. З появою JavaScript вони набули функціональної різноманітності. Незалежно від вибору створюваної програми, JavaScript надає всі інструменти та функції, необхідні для успішної розробки.

NoSQL database. NoSQL database (або «Not Only SQL») – це група баз даних, які відрізняються від традиційних реляційних баз даних (SQL) способом організації, зберігання та доступом до даних. NoSQL бази даних не використовують традиційну строгу модель зберігання даних у вигляді таблиці та не використовують мову запитів SQL.

Основні характеристики NoSQL баз даних:

– schema-less: NoSQL дозволяють зберігати різні типи даних без необхідності заздалегідь визначати схему таблиць;

– висока продуктивність: NoSQL бази даних пропонують високу швидкодію, особливо для простих операцій з даними. Це досягається шляхом спрощення структури даних та відмови від складних механізмів транзакцій.

– підтримка різних типів даних: NoSQL бази даних можуть зберігати дані різної структурованості [3].

Типи NoSQL баз даних:

– Document Stores: Дані зберігаються у вигляді документів, зазвичай у JSON або BSON форматі. Приклади: MongoDB, Couchbase.

– Graph Databases: Дані моделюються у вигляді графів, де вузли представляють об'єкти, а ребра - взаємозв'язки між ними. Приклади: Neo4j, Amazon Neptune.

– Column-Family Stores: Дані зберігаються у вигляді колекції стовпців, які можуть бути груповані в рядки. Приклади: Apache Cassandra, Apache HBase.

– Key-Value Stores: Кожному ключу (або ідентифікатору) відповідає значення. Приклади: Amazon DynamoDB, Redis.

На рис. 2.2 можна побачити графічне представлення зберігання даних у різних типах NoSQL баз даних.

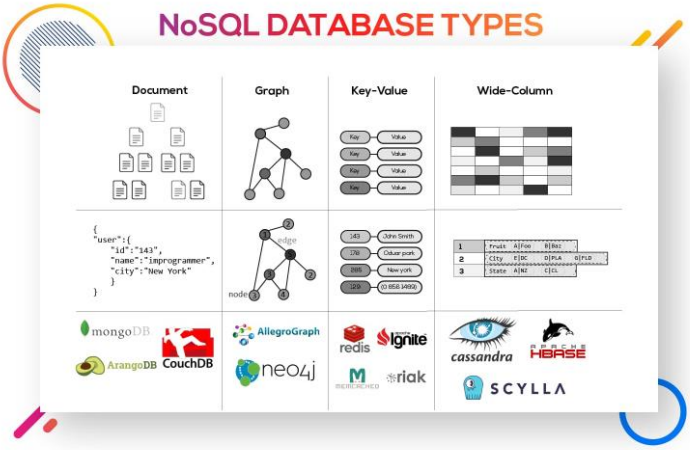


Рис. 2.2. Типи NoSQL – баз даних

Загалом, NoSQL бази даних можуть бути відмінним вибором для проектів, де потрібна гнучкість, швидкодія та масштабованість.

Бібліотека React. У межах кваліфікаційної роботи, для створення інтерфейсу веб-додатку була обрана бібліотека React. React – це бібліотека JavaScript для побудови користувацьких інтерфейсів, розроблена компанією Facebook. Вона призначена для створення компонентів користувацького інтерфейсу, які є багаторазовими, зручними для тестування та підтримки.

Через масштаб React часто називають фреймворком, хоча це не вірно. React дає мову шаблонів та деякі callback-функції для відтворення HTML. Весь результат роботи React це HTML. Зв'язки HTML/JavaScript, звані компонентами, займаються тим, що зберігають свій внутрішній стан у пам'яті, але в результаті на виході видається HTML. Зрозуміло, не можна побудувати

динамічний додаток, що повно функціонує, тільки з React. Ця бібліотека фокусується на одній основній задачі – роботі з інтерфейсом.

React працює на основі віртуального DOM (Document Object Model) і компонентної моделі розробки. Віртуальний DOM використовується для ефективного оновлення користувацького інтерфейсу. При зміні даних, замість оновлення всього DOM дерева, React оновлює лише ті частини, які були змінені. Це дуже вагомо впливає на оптимізацію додатків. При розробці інтерфейсу користувача, компонентна модель дозволяє розбивати його на модулі. Завдяки цьому проект має чітку структуру, що позитивно впливає на процес розробки та обслуговування додатків різної величини. Також, створені компоненти можна перевикористовувати, що дозволяє уникати дублювання коду для схожих блоків.

React забезпечує односторонній потік даних зверху вниз. Ця концепція полегшує розуміння та моніторинг потоку даних у додатку. React компонент витягує дані з батьківського компонента за допомогою властивостей(props) і використовує стан(state) для оновлення даних. Такий підхід спрощує чітке і передбачуване управління даними в додатку [4].

База даних MongoDB. У проекті кваліфікаційної роботи, для зберігання даних, була обрана база даних MongoDB. MongoDB — це безсхемна (NoSQL) база даних, яка зберігає дані у вигляді документів формату JSON-подібного BSON (Binary JSON). MongoDB була розроблена для обробки великого обсягу даних, забезпечуючи високу продуктивність, доступність та масштабованість. Бази даних у  MongoDB складаються з колекцій, які містять в собі документи, які в свою чергу складаються з полів. Поля є парами ключ-значення. Також, за потреби, колекції можна індексувати задля підвищення швидкості сортування даних. На рис. 2.3 можна побачити невелике порівняння SQL з MongoDB [5].

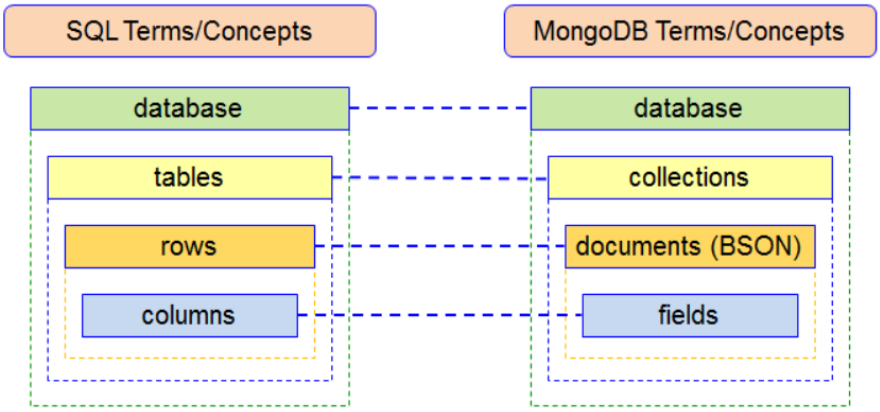


Рис. 2.3. Порівняння SQL з MongoDB

Фреймворк web-додатків для Node.js – Express**.** Для організації серверної частини CRM-системи був обраний Express.js Express.js — це мінімалістичний і гнучкий веб-фреймворк для Node.js, який надає надійний набір інструментів і можливостей для створення веб-додатків та API. Він був розроблений для спрощення процесу написання серверного коду, роблячи його більш структурованим і організованим. Express.js є одним з найпопулярніших фреймворків у світі Node.js завдяки своїй простоті, легкості у використанні та широким можливостям розширення.

Один з ключових аспектів Express.js полягає в тому, що він дозволяє розробникам створювати різні маршрути для обробки HTTP-запитів різних типів (GET, POST, PUT, DELETE та ін.). Це досягається через зрозумілий і лаконічний API, який дозволяє визначати маршрути та відповідні обробники у простий спосіб. Express.js також підтримує middleware, що дозволяє додавати функціональність до програми на різних етапах обробки запиту, таких як автентифікація, логування, обробка помилок та ін.

Express.js не нав'язує жорстку структуру проекту, що надає розробникам свободу організовувати свій код у спосіб, який найкраще підходить для їхнього проекту. Це робить його ідеальним вибором для різноманітних проектів — від простих однофайлових серверів до складних додатків з багатошаровою архітектурою. Завдяки цьому, Express.js часто використовується як основа для побудови повноцінних веб-фреймворків або для інтеграції з іншими

бібліотеками та інструментами, такими як шаблонізатори, системи керування сесіями та базами даних.

Однією з переваг використання Express.js є його сумісність з екосистемою Node.js, що дозволяє легко використовувати численні доступні пакети з npm. Це значно розширює можливості розробки та скорочує час на реалізацію нових функцій.

У підсумку, Express.js є потужним інструментом для створення серверних додатків, забезпечуючи легкість у використанні, гнучкість у налаштуванні та широкий набір можливостей для побудови сучасних веб-додатків та API [6].

Mongoose. Для створення структури документа колекції MongoDB був обраний Mongoose. Mongoose — це об'єктно-документний моделюючий (ODM) інструмент, розроблений спеціально для роботи з MongoDB і Node.js. Він надає розробникам зручний та інтуїтивно зрозумілий інтерфейс для взаємодії з MongoDB, спрощуючи процеси визначення схем, валідації, перетворення та бізнес-логіки даних.

Однією з ключових особливостей Mongoose є можливість визначення схем для документів у MongoDB. Схема визначає структуру документа, вказуючи типи даних для кожного поля, а також правила валідації та налаштування за замовчуванням. Це дозволяє забезпечити цілісність даних і уникнути помилок, які можуть виникнути через невідповідність структури даних. На основі визначених схем Mongoose створює моделі, які представляють собою об'єкти, що відповідають документам у колекції MongoDB. Моделі дозволяють виконувати CRUD (Create, Read, Update, Delete) операції над документами, а також використовувати складні запити.

Mongoose також надає можливості для реалізації відношень між документами, що можуть включати в себе вкладені документи (embedded documents) та посилання (references). Це дозволяє моделювати складніші структури даних і забезпечує більш гнучкі можливості для організації даних.

Ще однією важливою функцією Mongoose є middleware [7], або проміжне ПЗ, яке дозволяє виконувати додаткову логіку на різних етапах обробки документа, таких як збереження, оновлення або видалення.

Крім того, Mongoose має потужну систему валідації, яка дозволяє визначати правила перевірки для полів документів. Це включає перевірки на тип, діапазон значень, обов'язковість поля, унікальність та інші умови. Завдяки цьому можна бути впевненим у коректності даних, що зберігаються в базі [8].

Fetch API. Fetch API — це сучасний інтерфейс для виконання HTTP-запитів у веб-браузерах, який замінює застарілий об'єкт XMLHttpRequest. Він забезпечує простий і зрозумілий спосіб запитувати ресурси, такі як дані з сервера, та обробляти відповіді за допомогою промісів. Використання промісів робить роботу з асинхронними операціями більш зручною та передбачуваною, дозволяючи легко управляти успішними відповідями та помилками.

Fetch API підтримує всі типи HTTP-запитів, включаючи GET, POST, PUT, DELETE та інші [9], що дозволяє виконувати будь-які необхідні операції з сервером. Для отримання даних з сервера використовується метод fetch, який повертає проміс, що вирішується у відповідь на запит. Відповіді можна обробляти в різних форматах, таких як текст, JSON, блоби, що робить Fetch API універсальним для роботи з різними типами даних.

Асинхронність Fetch API дозволяє продовжувати виконання коду без очікування завершення запиту, що покращує продуктивність і зручність користування веб-додатком. Наприклад, при виконанні GET-запиту для отримання даних з API, якщо відповідь успішна, вона може бути перетворена у формат JSON для подальшої обробки. У разі помилки вона перехоплюється і обробляється відповідним блоком коду [10].

## 2.5. Опис структури програми та алгоритмів її функціонування

### 2.5.1. Структура системи

Веб-додаток інформаційної системи складається з таких компонентів:

Клієнтська частина:

– Форми створення та редагування елементів:

– AdminForm.js;

– Alert.js;

– ClientRegistrationForm.js;

– EditDate.js;

– EditEmail.js;

– editForm.js;

– EditFormBuyer.js;

– EditFormProduct.js;

– EditForms.js;

– EditFormTask.js;

– EditManager.js;

– EditManagerForm.js;

– EditMobile.js;

– EditName.js;

– EditNotice.js;

– EditNoticeMode.js;

– EditPayment.js;

– EditPaymentMode.js;

– EditProduct.js;

– EditSecondMobile.js;

– EditStatus.js;

– LeedInfo.js;

– LeedInfoClose.js;

– TaskForm.js;

– TaskFormRegister.js.

– Сторінка входу до системи:

– Login.js;

– Інтерфейс головної сторінки (менеджера та адміністратора):

– adminPage.js;

– managerPage.js;

– adminPageStyle.css;

– BuyerBlock.js;

– ClientsBlock.js;

– DeletedClientsListBlock.js;

– Diagram.js;

– Overlay.js;

– ProductBlock.js;

– View.js;

– LeftMenu.js.

– Методи для завантаження елементів з бази даних:

– gettingData.js.

– Методи додавання, редагування, видалення елементів:

– methods.js.

– Метод пошуку:

– Search.js.

Серверна частина:

– Колекції та схеми документів:

– buyers.js;

– clients.js;

– closeClients.js;

– manager.js;

– Notice.js;

– products.js;

– tasks.js.

– Сервер:

– server.js.

Веб-орієнтований додаток CRM-системи побудований за архітектурним шаблоном MVC (Model-View-Controller). Він розділяє додаток на три основні компоненти: Model (Модель) – сутності бази даних; View (Представлення) – візуалізація даних, у проекті організована за допомогою бібліотеки React; Controller (Контролер) – сервер, який налагоджує зв'язки між клієнтською частиною та базою даних, у проекті реалізований за допомогою express.js. За допомогою MVC забезпечується чітке розділення завдань у додатку, що полегшує його розробку, тестування та підтримку [11].

Алгоритм обміну даними між клієнтською та серверною сторонами. Алгоритм обміну даними між клієнтською та серверною сторонами у CRM-системі можна розділити на кілька етапів:

– Підключення та налаштування серверу:

– Сервер створюється за допомогою Express.js.

– Підключаються необхідні модулі, такі як mongoose для роботи з MongoDB, passport для аутентифікації та інші.

– Створюється HTTP-сервер та WebSocket сервер [12] для змін у реальному часі.

– Налаштування MongoDB:

– Підключення до бази даних MongoDB за допомогою mongoose. Приклад запиту:

```
mongoose.connect('mongodb+srv://asd155619:y99Ikl4KuNS62Ms1@cluster0.gzp8twh.mongodb.net/?retryWrite
s=true&w=majority&appName=Cluster0',
{ useNewUrlParser: true, useUnifiedTopology: true });
mongoose.connection.once('open', () => {
  console.log('Connected to MongoDB');
});
```

– Створення моделей для роботи з базою даних (User, Manager, Client, Buyer, Product, Notice, CloseClient, Task).

– Налаштування Passport для аутентифікації та авторизації:

– Встановлення стратегії аутентифікації (LocalStrategy).

– Визначення функцій serializeUser та deserializeUser для зберігання користувача в сесії.

– Використання Passport для перевірки електронної пошти та паролю під час входу.

– Використання Passport для перевірки електронної пошти та паролю під час входу.

– Перевірка ролі користувача (адміністратор чи менеджер) для доступу до певних маршрутів та функціональності.

Приклад:

```
passport.use(new LocalStrategy({ usernameField: 'email' }, async (email, password, done) => {
  try {
    const user = await Manager.findOne({ email });

    if (!user) {
      return done(null, false, { message: 'Incorrect email.' });
    }

    const isPasswordValid = password === user.password;

    if (!isPasswordValid) {
      return done(null, false, { message: 'Incorrect password.' });
    }

    return done(null, user);
  } catch (err) {
    return done(err);
  }
}));

passport.serializeUser((user, done) => {
  done(null, user.id);
});

passport.deserializeUser(async (id, done) => {
  try {
    const user = await User.findById(id);
```

```
    done(null, user);
  } catch (err) {
    done(err);
  }
});
```

– Роутинг та обробка запитів:

– Визначення маршрутів для реєстрації різних типів користувачів (менеджерів, клієнтів, покупців, продуктів, нотаток, завдань).

– Маршрутизація веб-додатку відповідно до ролі користувача. Приклад:

```
<Router>

    <div>
    <DndProvider backend={HTML5Backend}>
      <Routes>
        <Route path="/" element={<RegistrationPage />}></Route>
          <Route path="/director" element={<DirectorPage />} />
          <Route path="/manager" element={<ManagerPage />} />
          <Route path="/admin" element={<AdminPage />} />
      </Routes>
      </DndProvider>
    </div>

    <img src="../" alt="" />
  </Router>
```

– Обробка POST запитів для реєстрації даних у базі даних.

– Обробка GET запитів для отримання інформації про роль користувача та інші дані.

– Обробка DELETE запитів для видалення елементів.

– Обробка PUT запитів для редагування елементів.

– Взаємодія з базою даних:

– При кожному запиті (наприклад, реєстрація користувача або створення продукту) виконується відповідний запит до бази даних MongoDB через mongoose.

– Використовуються моделі для створення нових записів та збереження їх у базі даних.

– Обробка помилок при взаємодії з базою даних та відправка відповідних відповідей клієнту.

– Реалізація змін у реальному часі за допомогою WebSocket:

– Підключення користувачів до WebSocket сервера. Приклад:

```
wss.on('connection', ws => {
 ws.on('message', m => {
  console.log('Повідомлення від клієнта:', m);

  wss.clients.forEach(client => {
   if (client !== ws && client.readyState === WebSocket.OPEN) {
    client.send(m);
   }
  });
 });

 ws.on('close', () => {
  console.log('Клієнт від'єднався'); });
```

– Обробка повідомлень від користувачів та розсилка оновлень іншим підключеним користувачам.

Приклад взаємодії сторін на прикладі входу у систему, відображення вже існуючих карток клієнтів та додавання нових. При вході у систему користувач вводить логін та пароль у форму (рис. 2.4). Після введення даних та натискання кнопки «Увійти», надходить запит до сервера із введеними даними за маршрутом '/login'. Також відбувається GET-запит, за допомогою fetch, з отриманням ролі користувача, який повертає роль, ідентифікатор – managerID, ключ – managerKey, ключ адміністратора(якщо користувач - адміністратор), логін – userName, ім’я – user.

Приклад запиту:

– На стороні клієнта:

```
const getUserRole = async (email) => {
  try {
```

```
    const roleResponse = await fetch(`/api/userRole?email=${email}`);
    const { role, key, manID, manKey, nameUser } = await roleResponse.json();

    localStorage.setItem('adminKey', key);
    localStorage.setItem('managerID', manID)
    localStorage.setItem('managerKey', manKey)

    localStorage.setItem('userName', email)
    localStorage.setItem('userRole', role)
    localStorage.setItem('user', nameUser)

    return role;
  } catch (error) {
    console.error('Error during getUserRole:', error);
    return null;
  }
};
```

– На стороні сервера:

```
app.get('/api/userRole', async (req, res) => {
  const { email } = req.query;

  try {
    const user = await Manager.findOne({ email });

    if (!user) {
      res.status(404).json({ error: 'User not found' });
      return;
    }

     res.json({ role: user.role, key: user.adminKey, manID: user.managerID, manKey: user.managerKey, email:
user.email, nameUser: user.nameManager });

  } catch (error) {
    console.error('Error while fetching user role:', error);
    res.status(500).json({ error: 'Internal server error' });
  }
});
```

На стороні сервера отримується введений email, за яким відбувається пошук користувача у колекції Manager. Якщо користувача знайдено, повертається об’єкт з потрібними даними у форматі JSON. Якщо користувача не знайдено, з’являється відповідне повідомлення. Після отримання ролі користувача, відбувається перехід на відповідну сторінку.

Приклад запиту:

```
const handleLogin = async () => {
  try {
    const response = await fetch('/login', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify(loginData),
    });

    if (response.ok) {
      const userRole = await getUserRole(loginData.email);
      const redirectPath = userRole === 'admin' ? `/admin` : '/manager';
      window.location.href = redirectPath;

    } else {
      alert('Login failed. Please check your credentials and try again.');

    }
  } catch (error) {
    console.error('Error during login:', error);
  }
};
```

Після входу відбувається завантаження сторінки. У цей час відбувається завантаження даних за ключем та ідентифікатором користувача. Ліди завантажуються за відповідними статусами (Приклад маршруту запиту - api/getClients?status=wholesale)

Приклад запиту:

```
const getDataNew = async ( status = '') => {
 try {
  const response = await fetch(`/api/getClients?status=${status}`);

  if (response.ok) {
    return await response.json();
  } else {
    console.error('Failed to fetch users:', response.statusText);
  }
 } catch (error) {
  console.error('Error fetching users:', error);
 }
```

```
};
```

Після завантаження дані потрапляють до відповідних станів (state). Приклад методу завантаження та встановлення даних до стану:

```
const fetchData = async () => {
  try {
    const responseData1 = await getDataNew(statusFilter);
    const responseDataAgreed = await getDataNew(statusFilterAgreed);
    const responseDataInProcessing = await getDataNew(statusFilterInProcessing);
    const responseDataCancel = await getDataNew(statusFilterCancel);
    const responseDataSuccesful = await getDataNew(statusFilterSuccesful);
    const responseReturn = await getDataNew(statusFilterReturn);
    const responseDataNds = await getDataNew(statusFilterNds);
    const responseWholesale = await getDataNew(statusFilterWholesale);
    const responseManagers = await getManagers();
    const responseBuyers = await getBuyers();
    const responseProducts = await getProducts();
    const responseNotices = await getNotices();
    const responseCloseClients = await getCloseClients();
    const responseTasks = await getTasks();

    if (
      Array.isArray(responseData1) &&
      responseManagers &&
      responseBuyers &&
      responseProducts &&
      responseNotices &&
      responseCloseClients &&
      responseTasks
    ) {
      setData((prevData) => {
                const responseData = [...responseData1, ...responseDataAgreed, ...responseDataInProcessing,
...responseDataCancel, ...responseDataSuccesful, ...responseReturn, ...responseDataNds, ...responseWholesale]
        const existingData = Array.isArray(prevData) ? prevData : [];
            const uniqueData = responseData.filter((newItem) => !existingData.some((item) => item._id ===
newItem._id));
        const combinedData = [...existingData, ...uniqueData];
        const uncompletedTaskToday = combinedData.filter((task) => {
          const taskDate = new Date(task.selectedDate);
          const formattedDate = new Date(formattedDateTime);
          return taskDate < formattedDate;
        });
        const taskToday = combinedData.filter((task) => {
          const taskDate = new Date(task.selectedDate);
          const formattedDateYes = new Date(formattedDateTime);

          const formattedDate = new Date(formatDateTomorrow);
```

```
          const formattedDateYet = new Date(formatDateYet);
          return formattedDateYet < taskDate && taskDate < formattedDate;
        });

        setTaskToday(taskToday)
        console.log(uncompletedTaskToday)
        return combinedData;
      });
            const responseData = [...responseData1, ...responseDataAgreed, ...responseDataInProcessing, ...responseDataCancel, ...responseDataSuccesful, ...responseReturn, ...responseDataNds, ...responseWholesale]

      setDataManagers(responseManagers);
      setDataBuyers(responseBuyers);
      setDataProducts(responseProducts);
      setDataNotices(responseNotices);
      setCloseClients(responseCloseClients);
      setDataTasks(responseTasks)
            setMyDataLeed(responseData.filter((person) => storedAdminKey === person.managerKey || storedAdminKey === person.managerID))
      setDataLoaded(true);
    } else {
      console.error('Some data failed to load');
    }
  } catch (error) {
    console.error('Error fetching data:', error);
  }
 };
```

За допомогою JSX дані потрапляють на сторінку користувача:

```
<div className="table_block">
      <div className="statusBlock">
            <ul className="statusCols" onDragOver={(event) => allowDrop(event)} onDrop={(event) => handleDrop(event, 'new')}>
         <div className="status_name">
          <h4>Потенціальний ({renderStatusElements("new", newStatusColsRef, 100000000).length})</h4>
            <div className="upload_el" onClick={() => handleLoadMoreWithStatusFilter('new', setStatusFilter, setLoadedItems)}><img className="update_img" src={UpdateImg} /></div>

         </div>
         <div className="status_els" ref={newStatusColsRef}>
          {renderStatusElements("new", newStatusColsRef, visibleItemsNew)}

         </div>

        </ul>
            <ul className="statusCols" onDragOver={(event) => allowDrop(event)} onDrop={(event) => handleDrop(event, 'in_processing')}>
```

```
        <div className="status_name">
                        <h4>Я подумаю, набирати систематично ({renderStatusElements("in_processing",
inProcessingStatusColsRef, 10000000000).length})</h4>
                <div  className="upload_el"  onClick={()  =>  handleLoadMoreWithStatusFilter('in_processing',
setStatusFilterInProcessing,  setLoadedItemsInProcessing)}><img  className="update_img"  src={UpdateImg}
/></div>

        </div>
        <div className="status_els" ref={inProcessingStatusColsRef}>
          {renderStatusElements("in_processing", inProcessingStatusColsRef, visibleItemsInProcessing)}

        </div>
       </ul>
            <ul  className="statusCols"  onDragOver={(event)  =>  allowDrop(event)}  onDrop={(event)  =>
handleDrop(event, 'agreed')}>
        <div className="status_name">
             <h4>Домовилися, але не відвантажили ({renderStatusElements("agreed", agreedStatusColsRef,
1000000000000).length})</h4>
                  <div  className="upload_el"  onClick={()  =>  handleLoadMoreWithStatusFilter('agreed',
setStatusFilterAgreed, setLoadedItemsAgreed)}><img className="update_img" src={UpdateImg} /></div>

        </div>
        <div className="status_els" ref={agreedStatusColsRef}>
          {renderStatusElements("agreed", agreedStatusColsRef, visibleItemsAgreed)}

        </div>
       </ul>
            <ul  className="statusCols"  onDragOver={(event)  =>  allowDrop(event)}  onDrop={(event)  =>
handleDrop(event, 'successful')}>
        <div className="status_name">
                              <h4>Успішні  ({renderStatusElements("successful",  succesfulStatusColsRef,
1000000000000).length})</h4>
                <div  className="upload_el"  onClick={()  =>  handleLoadMoreWithStatusFilter('successful',
setStatusFilterSuccesful, setLoadedItemsSuccesful)}><img className="update_img" src={UpdateImg} /></div>

        </div>
        <div className="status_els" ref={succesfulStatusColsRef}>
          {renderStatusElements("successful", succesfulStatusColsRef, visibleItemsSuccessful)}

        </div>
       </ul>
            <ul  className="statusCols"  onDragOver={(event)  =>  allowDrop(event)}  onDrop={(event)  =>
handleDrop(event, 'return')}>
        <div className="status_name">
                <h4>Клієнт купив в іншому місці ({renderStatusElements("return", returnStatusColsRef,
1000000000000).length})</h4>
                  <div  className="upload_el"  onClick={()  =>  handleLoadMoreWithStatusFilter('return',
setStatusFilterReturn, setLoadedItemsReturn)}><img className="update_img" src={UpdateImg} /></div>

        </div>
        <div className="status_els" ref={returnStatusColsRef}>
```

```
          {renderStatusElements("return", returnStatusColsRef, visibleItemsReturn)}

        </div>
       </ul>
            <ul  className="statusCols"  onDragOver={(event)  =>  allowDrop(event)}  onDrop={(event)  =>
handleDrop(event, 'nds')}>
        <div className="status_name">
         <h4>НДС ({renderStatusElements("nds", ndsStatusColsRef, 10000000000).length})</h4>
         <div className="upload_el" onClick={() => handleLoadMoreWithStatusFilter('nds', setStatusFilterNds,
setLoadedItemsNds)}><img className="update_img" src={UpdateImg} /></div>

        </div>
        <div className="status_els" ref={ndsStatusColsRef}>
         {renderStatusElements("nds", ndsStatusColsRef, visibleItemsNds)}

        </div>
       </ul>
            <ul  className="statusCols"  onDragOver={(event)  =>  allowDrop(event)}  onDrop={(event)  =>
handleDrop(event, 'wholesale')}>
        <div className="status_name">
         <h4>Опт ({renderStatusElements("wholesale", wholesaleStatusColsRef, 10000000000).length})</h4>
                <div  className="upload_el"  onClick={()  =>  handleLoadMoreWithStatusFilter('wholesale',
setStatusFilterWholesale,    setLoadedItemsWholesale)}><img    className="update_img"    src={UpdateImg}
/></div>

        </div>
        <div className="status_els" ref={wholesaleStatusColsRef}>
         {renderStatusElements("wholesale", wholesaleStatusColsRef, visibleItemsWholesale)}

        </div>
       </ul>
        {showCancelStatus && <ul className="statusCols cancel" onDragOver={(event) => allowDrop(event)}
onDrop={(event) => handleDrop(event, 'cancel')}>
        <div className="status_name">
         <h4>Відмінені ({renderStatusElements("cancel", cancelStatusColsRef, 1000000000000).length})</h4>
                <div  className="upload_el"  onClick={()  =>  handleLoadMoreWithStatusFilter('cancel',
setStatusFilterCancel, setLoadedItemsCancel)}><img className="update_img" src={UpdateImg} /></div>

        </div>
        <div className="status_els" ref={cancelStatusColsRef}>
         {renderStatusElements("cancel", cancelStatusColsRef, visibleItemsCancel)}

        </div>
       </ul>}
      </div>
     </div>
```

Підсумуючи, загальний алгоритм взаємодії різних частин CRM-системи можна описати так:

– Фронтенд: Користувач заповнює форму додавання, редагування елемента та надсилає її на сервер, або видаляє елемент.

– Маршрутизація (Express): запит надходить на відповідний маршрут;

– Контролер: Маршрут обробляє запит та викликає відповідний контролер, який приймає дані з запиту (req.body).

– Модель (Mongoose): Контролер створює новий елемент, або редагує, або видаляє за допомогою моделі та зберігає зміни у базі даних.

– Відповідь клієнту: Після успішного збереження контролер відправляє відповідь клієнту з повідомленням про успішну операцію та відбуваються певні зміни на інтерфейсі користувача.

–WebSocket: Якщо необхідно повідомити іншим клієнтам про нову подію, інформація надсилається всім підключеним клієнтам через WebSocket.

### 2.5.2. Структура бази даних

База даних CRM системи, представленої у кваліфікаційній роботі, є нереляційною. Дані зберігаються у вигляді документів, у форматі BSON(Binary JSON). База даних розподілена на колекції – набори документів.

Перелік колекцій та їх атрибутів. База даних містить такі колекції: managers, clients, closeclients, buyers, notices, products, tasks.

Колекція managers містить дані користувачів системи. Структура документа:

– _id: ідентифікатор користувача у колекціїї;

– nameManager: ім’я користувача;

– email: електронна пошта користувача, використовується у якості логіну при вході у систему;

– password: пароль, використовується при вході у систему;

– role: роль користувача, може бути admin або manager;

– adminKey(є лиже у ролі admin): персональний ключ адміністратора, за яким відбувається ідентифікація його менеджерів;

– managerKey(є лиже у ролі manager): містить ключ адміністратора, за яким закріплюється належність менеджера до його адміністратора;

– managerID: ідентифікатор, за допомогою якого закріплюється належність клієнта, товару та ін. до користувача.

Колекція clients містить дані про клієнтів. Структура документа:

– _id: ідентифікатор клієнта у колекції;

– clientName: містить ім’я клієнта;

– email: електронна пошта клієнта;

– phone: номер телефону клієнта;

– secondPhone: додатковий номер телефону клієнта:

– managerID: ідентифікатор, за допомогою якого закріплюється належність клієнта до користувача;

– managerKey: ключ, за допомогою якого на сторінці адміністратора відображаються картки клієнтів всіх його менеджерів;

– status: статус клієнта;

– product: список товарів, які замовив клієнт;

– payment: список оплат, які клієнт здійснив або повинен здійснити;

– notice: замітка до картки клієнта;

– selectedDate: дата та час, коли потрібно здійснити контакт з клієнтом;

– dateOfCreated: дата та час створення картки клієнта.

Колекція closeclients містить дані про видалених клієнтів, слугує у якості корзини, звідки можна видалити картку клієнта назавжди. Структура документа:

– _id: ідентифікатор клієнта у колекції;

– clientName: містить ім’я клієнта;

– email: електронна пошта клієнта;

– phone: номер телефону клієнта;

– secondPhone: додатковий номер телефону клієнта:

– managerID: ідентифікатор, за допомогою якого закріплюється належність клієнта до користувача;

– managerKey: ключ, за допомогою якого на сторінці адміністратора відображаються картки клієнтів всіх його менеджерів;

– status: статус клієнта;

– product: список товарів, які замовив клієнт;

– payment: список оплат, які клієнт здійснив або повинен здійснити;

– notice: замітка до картки клієнта;

– selectedDate: дата та час, коли потрібно здійснити контакт з клієнтом;

– dateOfCreated: дата та час створення картки клієнта.

Колекція buyers містить дані про збережених клієнтів. При створенні ліда можна обрати зі списку для швидкого заповнення даних. Структура документа:

– _id: ідентифікатор клієнта у колекції;

– name: містить ім’я клієнта;

– phone: номер телефону клієнта;

– email: електронна пошта клієнта;

– notice: замітка;

– managerID: ідентифікатор, за допомогою якого закріплюється належність клієнта до користувача;

– managerKey: ключ, за допомогою якого на сторінці адміністратора відображаються клієнти всіх його менеджерів.

Колекція notices містить дані про коментарі у певній картці клієта. Структура документа:

– _id: ідентифікатор коментаря у колекції;

– noticeID: ідентифікатор, за допомогою якого закріплюється належність коментаря до картки клієнта;

– content: контент коментаря;

– noticeDate: дата та час створення коментаря;

Колекція notices містить дані про товари. Структура документа:

– _id: ідентифікатор товару у колекції;

– name: назва товару;

– cost: ціна товару;

– count: кількість товару на складі.

Notices містить дані про задачі, встановлені адміном менеджеру. Структура документа:

– _id: ідентифікатор задачі у колекції;

– taskLine: опис поставленої задачі;

– startDate: дата початку виконання;

– endDate: дата кінця виконання;

– createdDate: дата та час створення задачі;

– managerID: ідентифікатор, за допомогою якого закріплюється належність задачі до користувача;

–managerKey: ключ, за допомогою якого на сторінці адміністратора відображаються задачі всіх його менеджерів;

– taskStatus: статус виконання задачі.

## 2.6. Обґрунтування та організація вхідних та вихідних даних програми

Вхідні дані надходять до CRM-системи шляхом введення їх форму та завантаженням з бази даних. До вхідних даних належить вся інформація клієнтської та продуктової бази. Дані вводяться користувачем у відповідні поля форми. Нижче наведено кілька прикладів:

– введення даних при вході користувача (рис. 2.4)

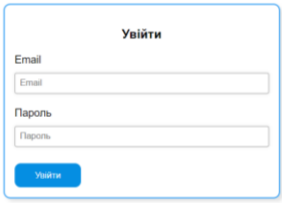


Рис. 2.4. Сторінка входу

– введення даних при створенні клієнтської картки (рис. 2.5)

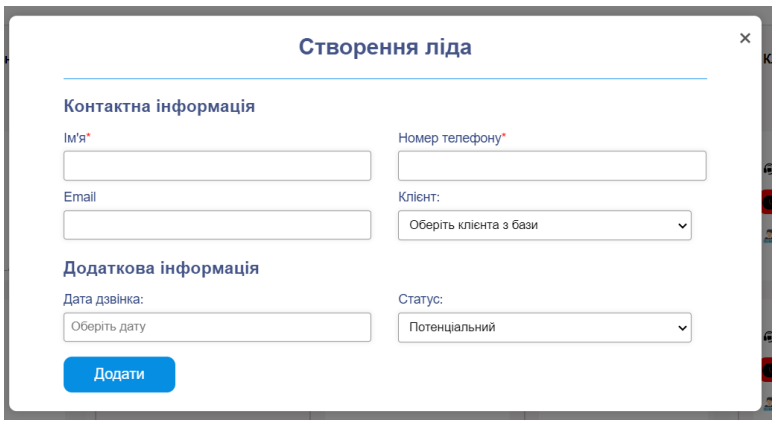


Рис. 2.5. Форма створення клієнтської картки

– введення даних при створенні менеджера (рис. 2.6)

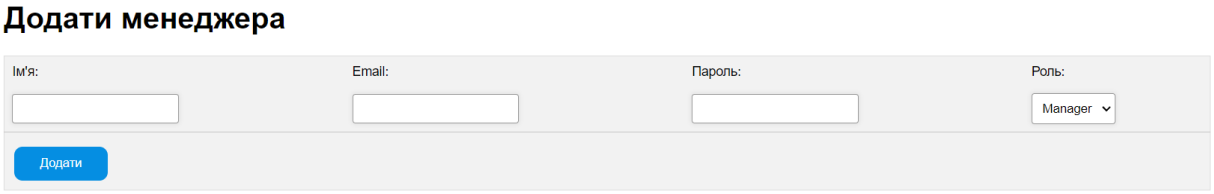


Рис. 2.6. Форма створення менеджера

До вихідних даних належить:

– сторінка «Ліди»;

– сторінка «Товари»;

– сторінка «Клієнти»;

– сторінка «Аналітика»;

– сторінка «Видалені»;

– сторінка «Менеджери» (лише у адміністратора);

– сторінка «Задачі» (лише у менеджера);

## 2.7. Опис розробленого програмного продукту

### 2.7.1. Використані технічні засоби.

Веб-додаток CRM-системи було розроблено та протестовано на портативному персональному комп'ютері з такими характеристиками:

– Процесор: AMD Ryzen 5 5600h 3.30 Ghz;

– Відеокарта: Nvidia Geforce RTX 3050, 4 ГБ;

– Екран: 1920x1080 Full HD, 15,6", 144 Гц;

– Об'єм ОЗП: 16ГБ;

– Накопичувач: SSD 512ГБ;

### 2.7.2. Використані програмні засоби

У ході розробки веб-орієнтованого додатку були використані такі програмні засоби:

– Visual Studio Code;

– Node.js;

– MongoDB Atlas;

– MongoDB Database Tools;

Visual Studio Code. Для розробки проекту кфаліфікаційної роботи був обраний редактор коду Visual Studio Code. Visual Studio Code (VS Code) — це безкоштовний, кросплатформний редактор коду, розроблений корпорацією Microsoft. Він призначений для розробників програмного забезпечення і підтримує велику кількість мов програмування та технологій. Основні переваги VS Code включають високу продуктивність, гнучкість і можливість налаштування. Редактор має інтуїтивно зрозумілий інтерфейс, який забезпечує зручний доступ до всіх необхідних інструментів та функцій. VS Code також підтримує конфігурацію робочого середовища за допомогою налаштувань та тем, що дозволяє розробникам адаптувати редактор під свої потреби. Важливою особливістю є можливість роботи з віддаленими репозиторіями і серверами. [13]

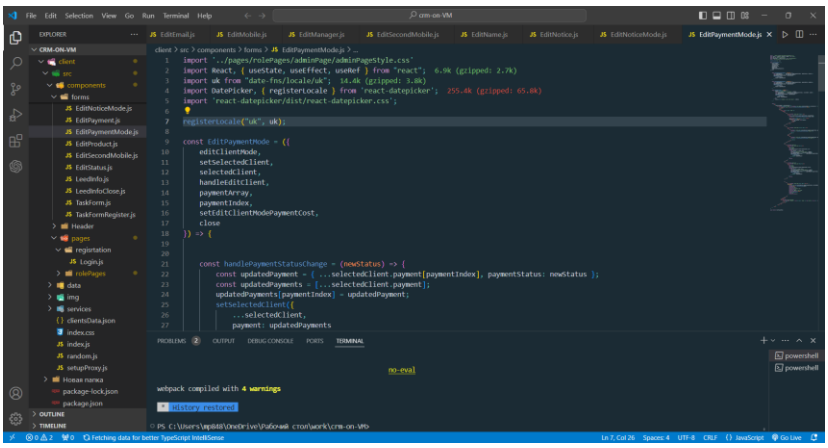

Рис. 2.7. Інтерфейс Visual Studio Code

Node.js. Node.js — це відкрите серверне середовище, яке виконує JavaScript-код поза браузером. Воно засноване на движку V8 від Google, який використовується в браузері Chrome для інтерпретації JavaScript-коду. Node.js дозволяє розробникам використовувати JavaScript для написання серверних скриптів, що робить можливим створення динамічних веб-сайтів і веб-додатків. Node.js має багату екосистему модулів і пакетів, доступних через менеджер пакетів npm, що значно полегшує розробку складних додатків. Завдяки своїй гнучкості, швидкодії та зручності використання, Node.js здобув широку популярність серед розробників. [14]

MongoDB Atlas. MongoDB Atlas — це багатохмарна служба баз даних від розробників MongoDB. Atlas спрощує розгортання баз даних і керування ними, водночас пропонуючи універсальність, необхідну для створення стійких і продуктивних глобальних додатків у хмарних провайдерах. [15]

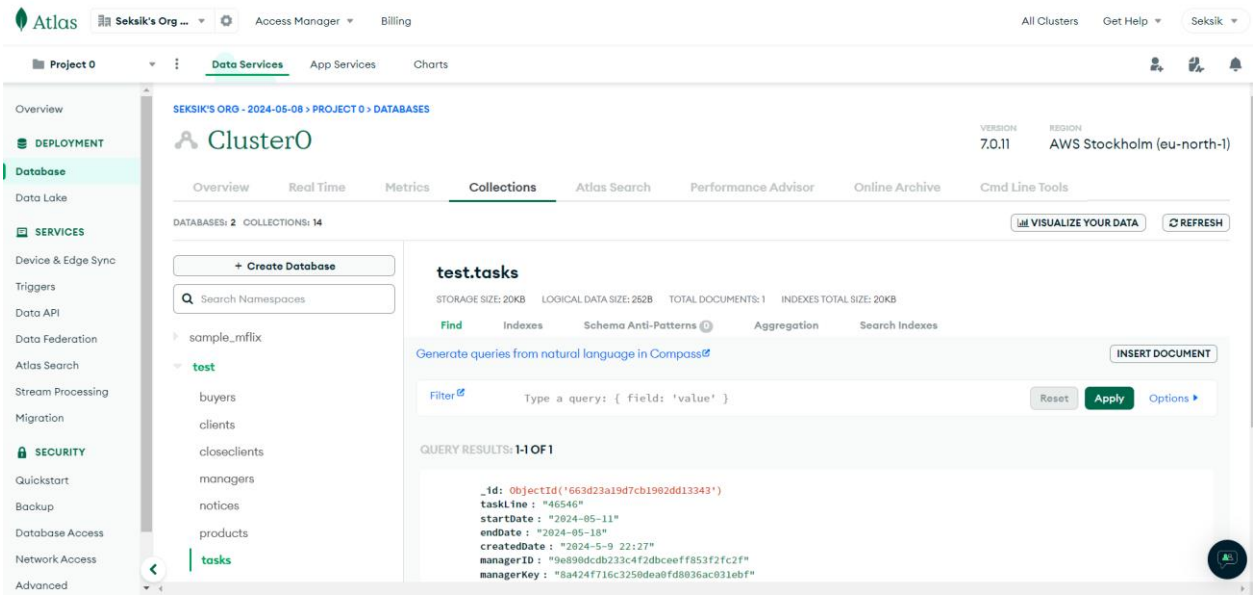


Рис. 2.8. Інтерфейс MongoDB Atlas

MongoDB Database Tools. MongoDB Database Tools – це набір утиліт командного рядка для роботи з розгортанням MongoDB. Інструменти бази даних включають такі двійкові файли:

– Двійковий імпорт/експорт:

– mongodump: Створює двійковий експорт вмісту бази mongod-даних.

– mongorestore: Відновлює дані з дампа бази даних mongodump у mongod або mongos

– bsondump: Перетворює файли дампа BSON на JSON.

– Імпорт / Експорт даних:

– mongoimport: Імпортує вміст із експортованого файлу Extended JSON, CSV або TSV.

– mongoexport: Експортує дані, що зберігаються в екземплярі mongod, у форматі JSON або CSV.

– Діагностичні засоби:

– mongostat: Надає швидкий огляд статусу поточного запущеного mongod або екземпляра mongos.

– mongotop: Надає огляд часу, який примірник mongod витрачає на читання та запис даних.

– Інструменти GridFS:

– mongofiles: Підтримує маніпулювання файлами, що зберігаються у вашому екземплярі MongoDB, в об’єктах GridFS. [16]

### 2.7.3. Виклик та завантаження програми.

У процесі розробки програма викликалася з ПК та орендованого хмарного серверу у AWS(EC2 Instance). Для виклику та завантаження веб-додатку на локальному ПК необхідно виконати наступне:

– Встановити Node.js;

– Налаштувати проксування запитів з клієнтської частини до серверу;

– Запустити паралельно React-додаток та сервер;

– Ввести у адресний рядок браузера URL: localhost:3000/;

– Якщо адреса э коректною, у вікні браузеру буде відкрито додаток;

Для виклику та завантаження веб-додатку з хмарного серверу необхідно виконати наступне:

– Встановити NVM для гнучкого контролю над версіями Node.js та npm;

– Встановити Nginx для статичного контенту;

– Встановити pm2 для запуску сервера;

– Створити конфігураційні файли для коректної взаємодії клієнтської та серверної частини;

– Запустити Nginx та pm2;

– Ввести у адресний рядок браузера ip-адресу, яка була виділена під час реєстрації;

– Якщо адреса э коректною, у вікні браузеру буде відкрито додаток;

### 2.7.4. Опис інтерфейсу користувача.

При завантаженні CRM-системи користувач потрапляє на сторінку входу у систему (рис. 2.9), де йому потрібно ввести логін та пароль.

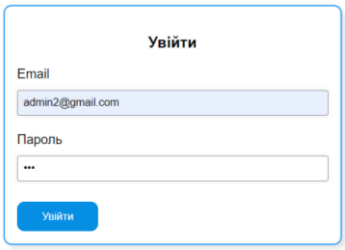


Рис. 2.9. Сторінка входу у систему

Якщо були введені неправильні дані, на екран буде виведено повідомлення про це (рис. 2.10)

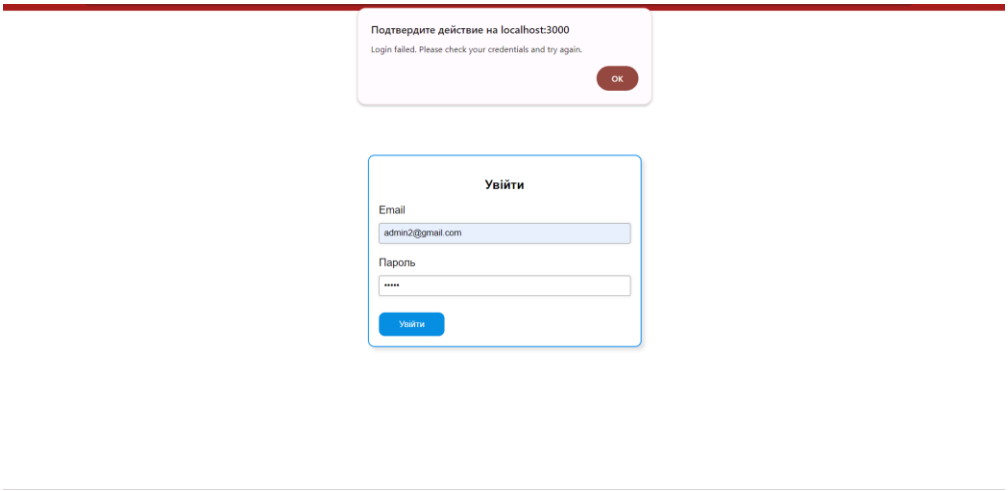


Рис. 2.10. Повідомлення про неправильно введені дані

Якщо дані було введено правильно, користувач потрапляє на головну сторінку системи (рис. 2.11). Вхід було здійснено адміністратором.

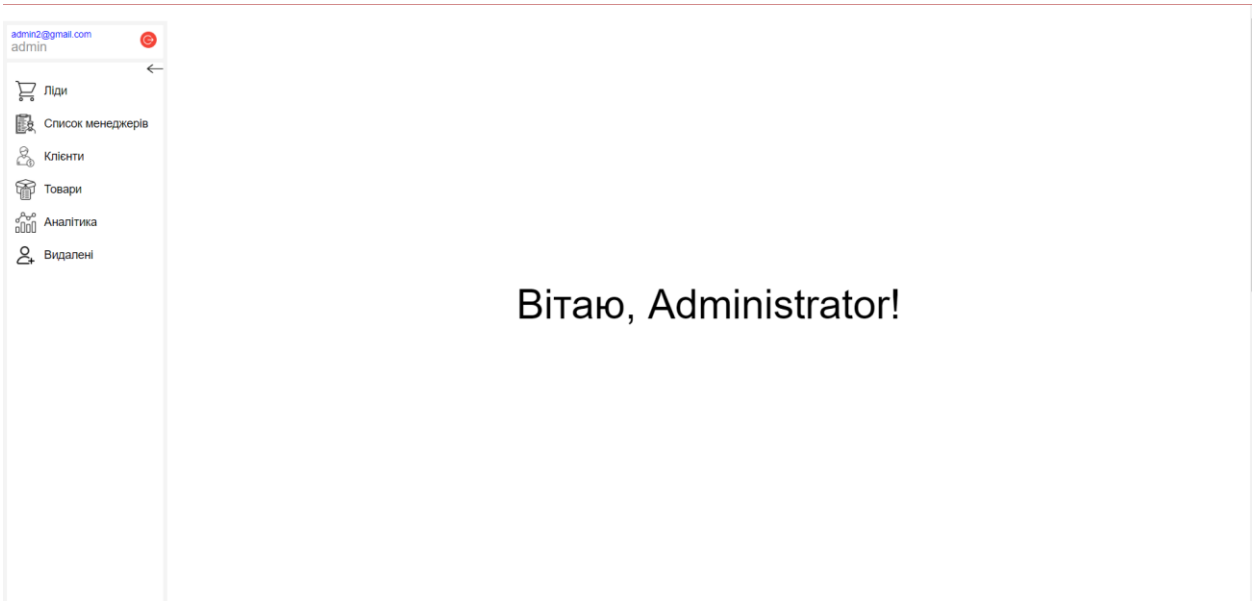


Рис. 2.11. Головна сторінка системи

Вона складається з двох частин - блока меню (ліва частина) та блока контенту(права частина). За потреби, блок меню можна зменшити (рис. 2.12)

Рис. 2.12. Зменшений блок меню

Вкладка «Ліди». Натиснувши на кнопку «Ліди», з’являється блок керування картками клієнтів (рис. 2.13)

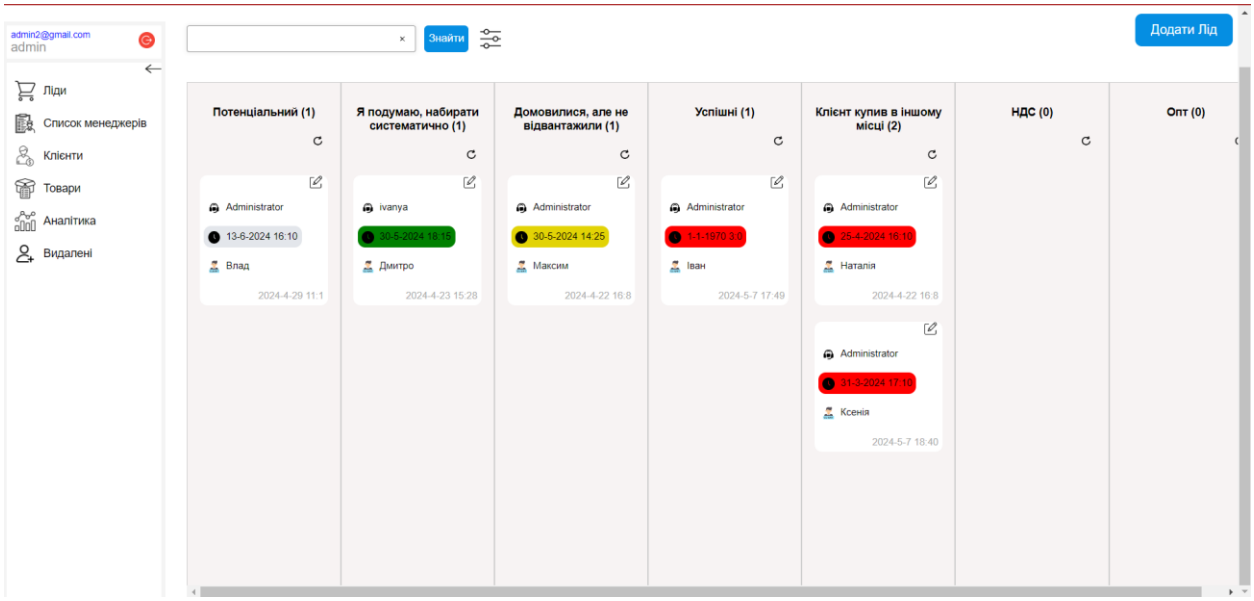


Рис. 2.13. Блок керування картками клієнтів

На ньому можна побачити поле для пошуку загальною базою за номером телефону (цілим або частиною) з кнопкою «Знайти», кнопку для відображення блоку фільтрів (рис. 2.14), кнопку для створення нового ліда та блок, який містить картки клієнтів, розміщені у колонки за статусом.


Рис. 2.14. Блок фільтрів

На блоці фільтрів можна побачити такі елементи: блок «Період» - містить два поля для встановлення початкової та кінцевої дати дзвінка (приклад – рис. 2.15), блок «Сортування» - містить кнопки для сортування за певними параметрами, блок «Відмінені» - містить перемикач, активувавши який відобразиться додаткова колонка зі статусом «Відмінені», яка за замовчуванням є прихованою (приклад – рис. 2.16). Нижче знаходиться кнопка «Менеджери», при наведенні на яку відображається список менеджерів із чекбоксами, для сортування карток по менеджерах (приклад – рис. 2.17). Нижче знаходиться

поле для сортування карток за ім’ям клієнта (приклад – рис. 2.18). Пошук відбувається за повним ім’ям та за частиною

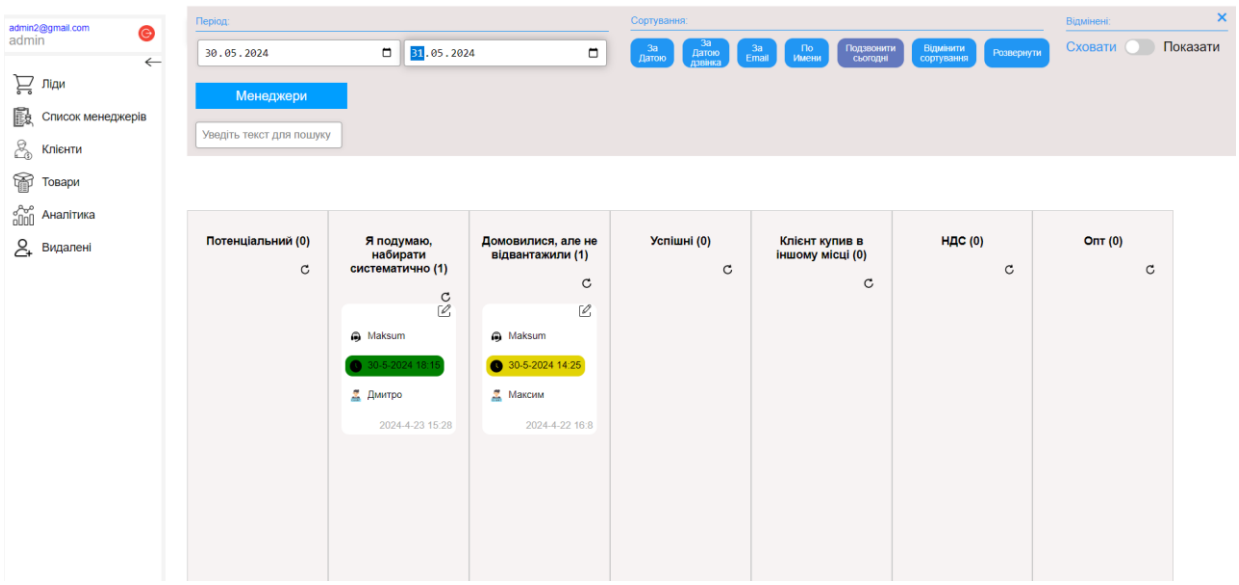


Рис. 2.15. Пошук за періодом

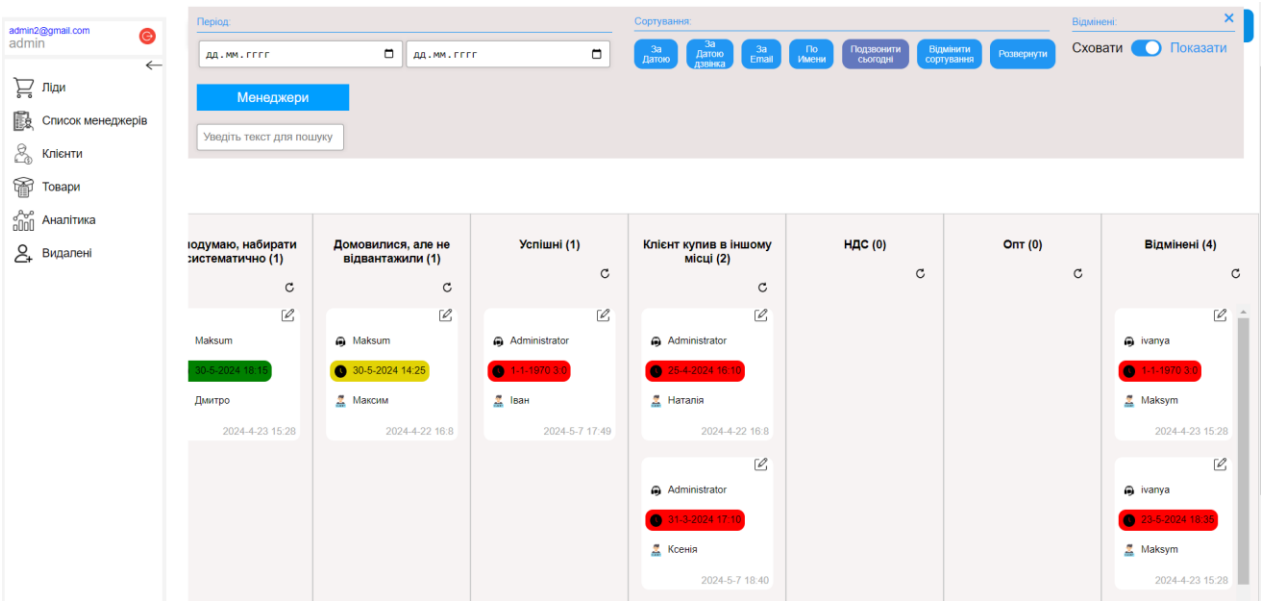


Рис. 2.16. Активований блок «Відмінені»

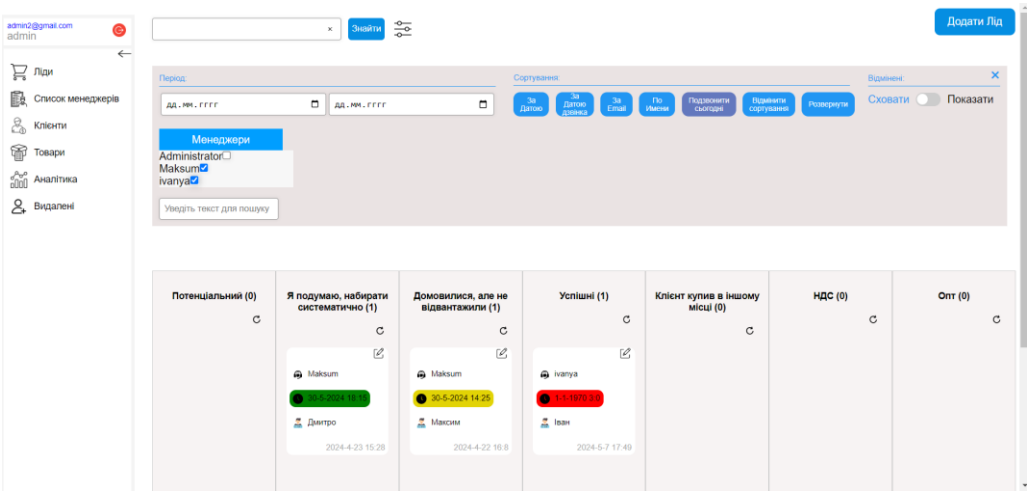


Рис. 2.17. Сортування за менеджерами

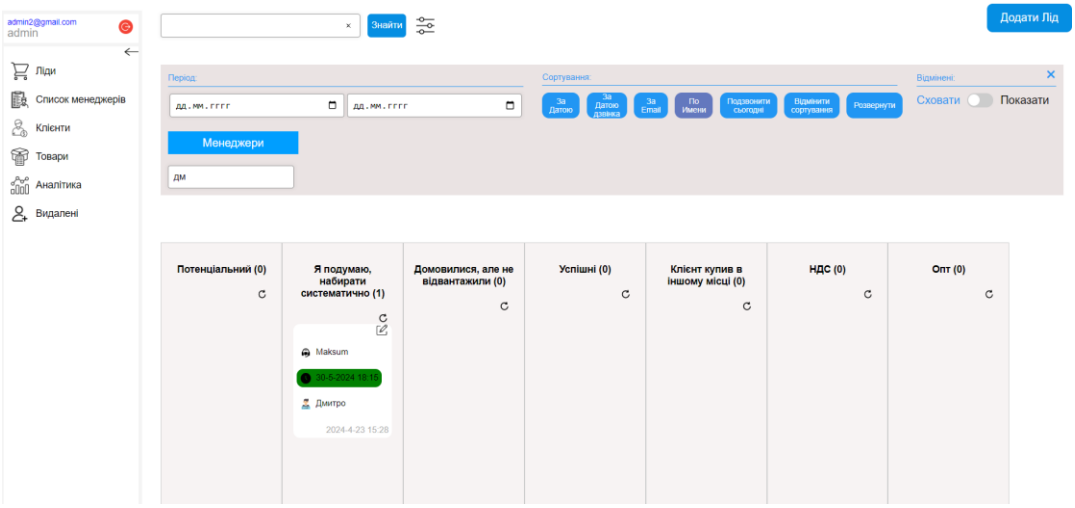


Рис. 2.18. Сортування за ім’ям

Картки клієнтів мають різне забарвлення частини з датою та часом дзвінка. Червоний колір – дата дзвінка сплила на день та більше. Жовтий колір – час дзвінка сплив більше ніч на час, дата «сьогоднішня». Зелений колір – час дзвінка не сплив або не більше ніж на час, дата «сьогоднішня». Сірий колір – дата дзвінка «наступного дня» або більше.

Приклад пошуку за номером телефону можна побачити на рис. 2.19.

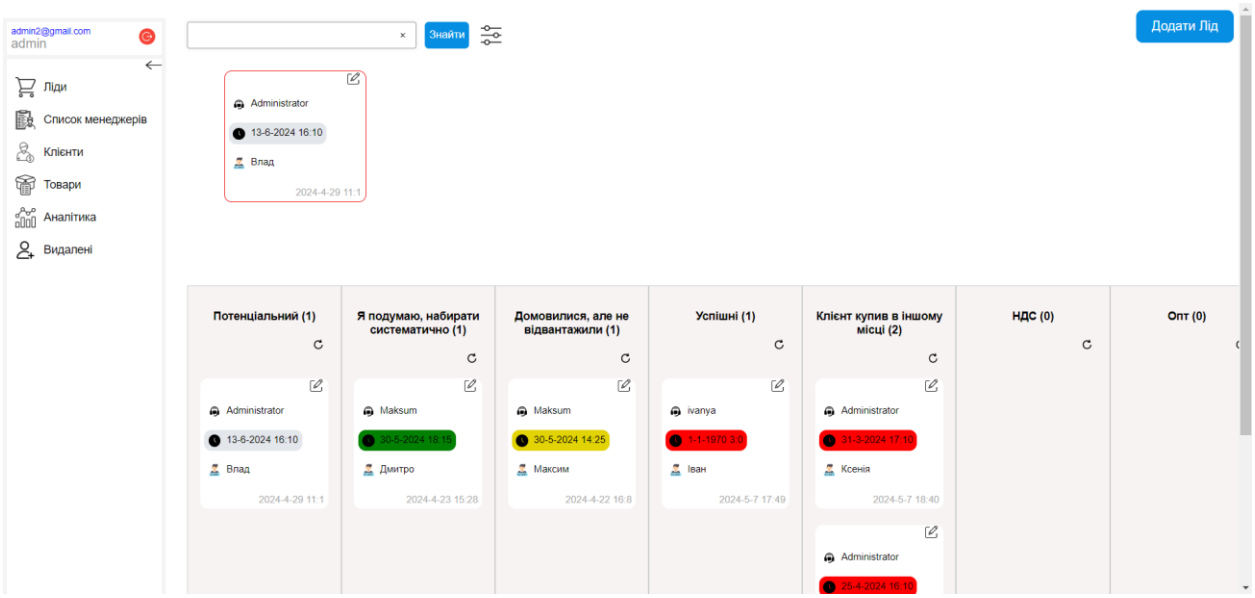


Рис. 2.19. Результат пошуку

Картки клієнтів можна перетягувати (drag and drop) з колонки в колонку з автоматичною зміною статусу (рис. 2.20).

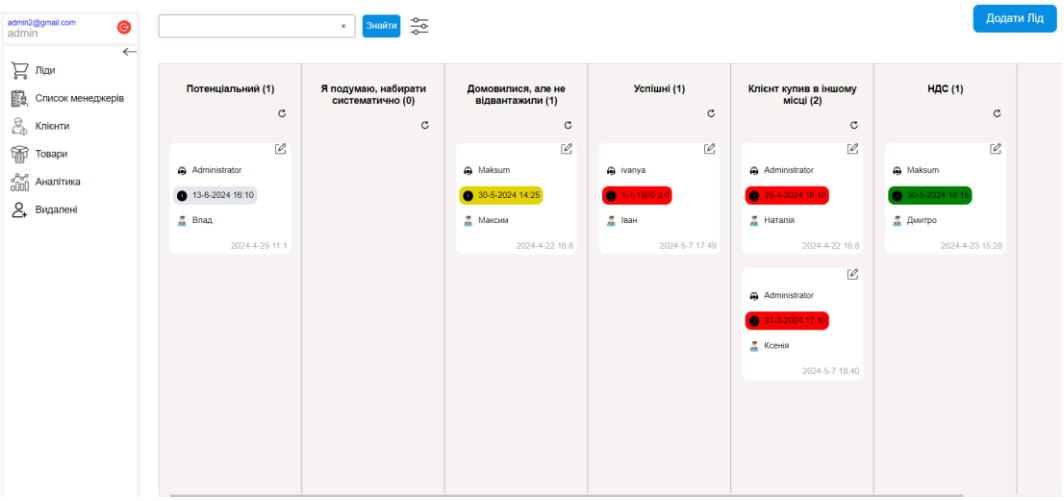


Рис. 2.20 – Результат перетягування

Натиснувши кнопку «Додати Лід», з’являється модальне вікно з формою створення картки клієнта (рис. 2.21).

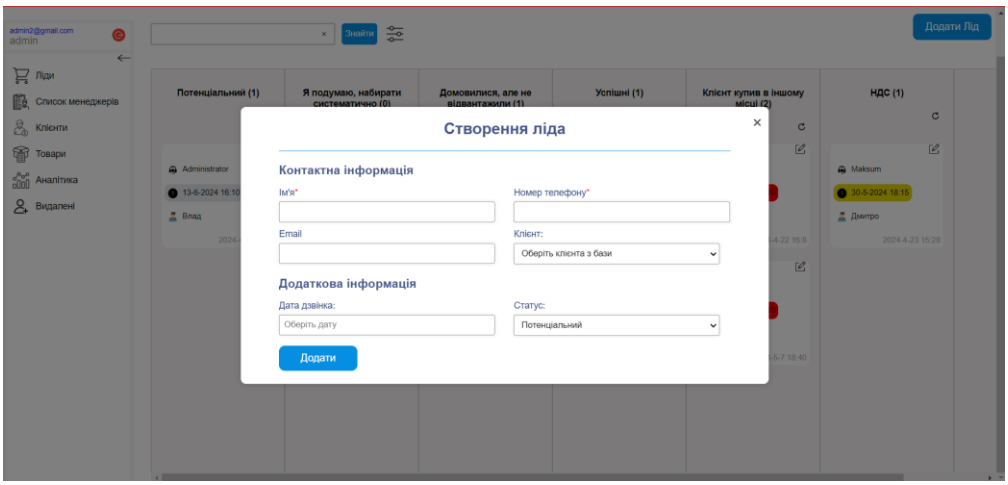


Рис. 2.21. Форма створення картки клієнта

Якщо при створенні картки вказати номер телефону, який є у базі, з’явиться повідомлення про це (рис. 2.22).

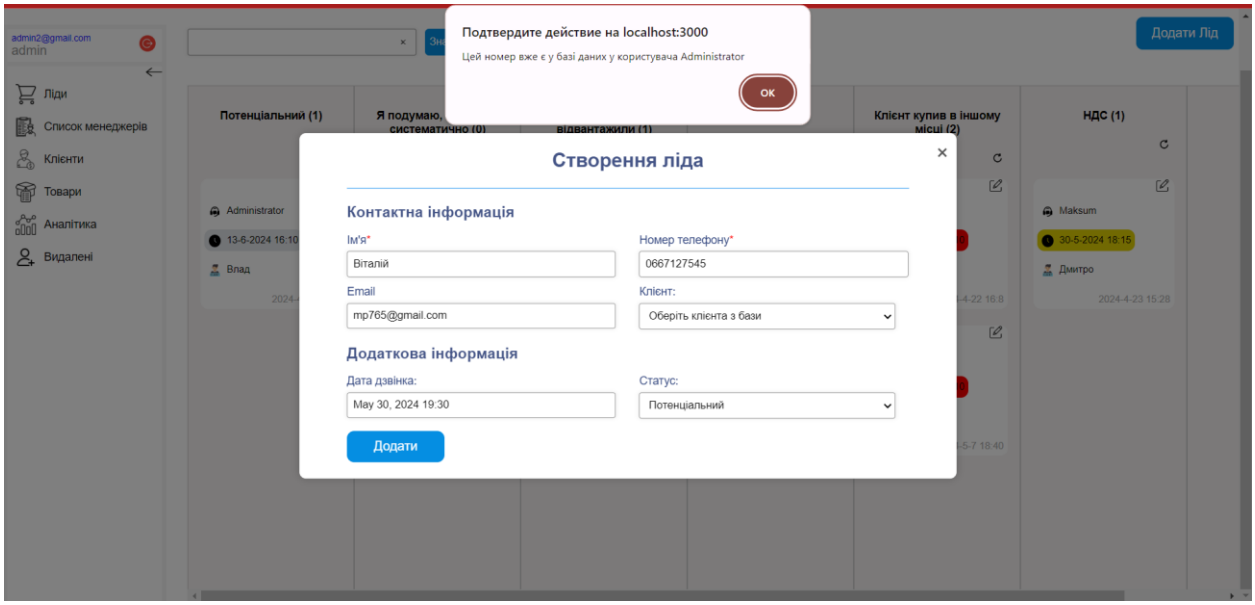


Рис. 2.22. Повідомлення про дубль номеру телефону

Заповнивши форму та натиснувши кнопку «Додати», створюється та розгортається картка (рис. 2.23 – рис. 2.24).

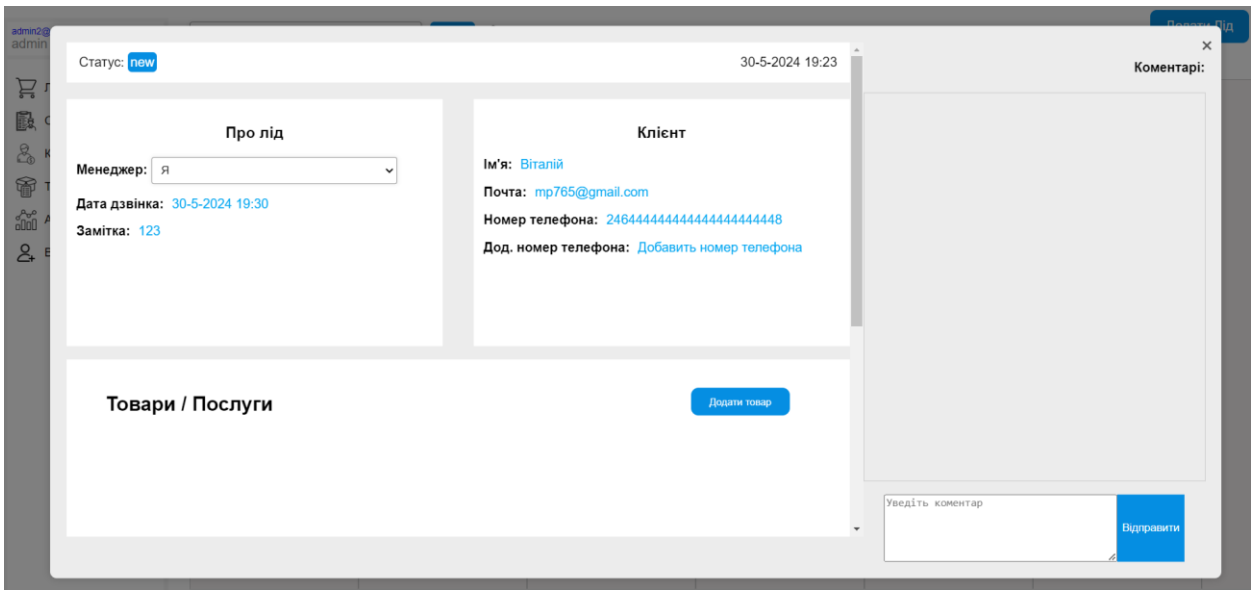


Рис. 2.23. Розгорнута картка (ч1)

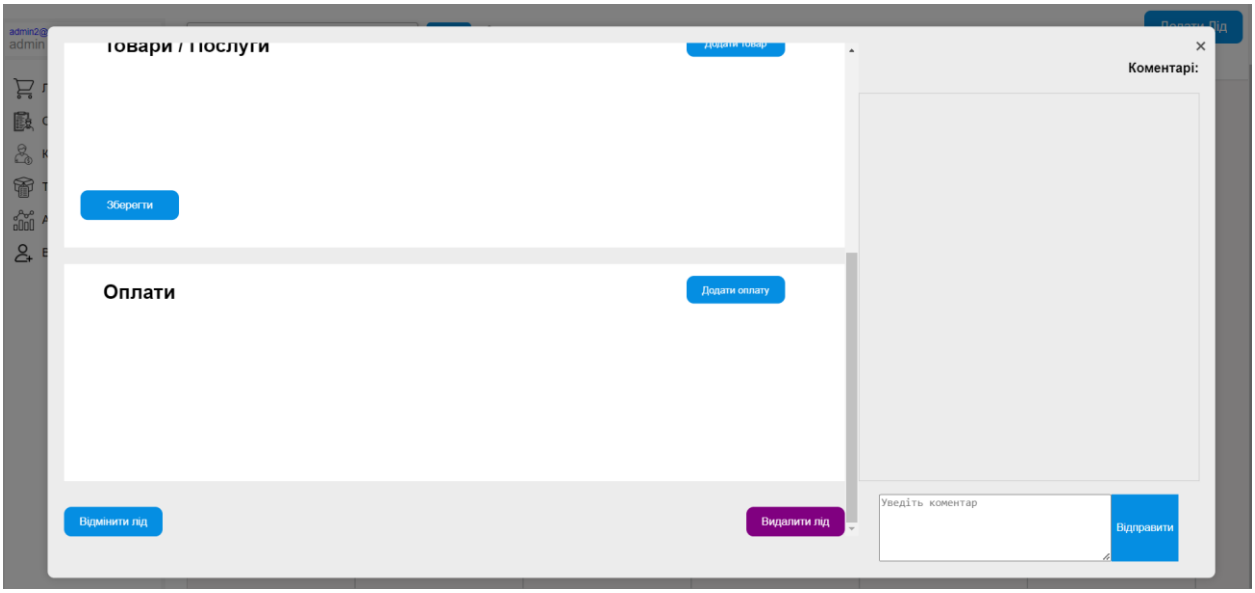


Рис. 2.24. Розгорнута картка (ч2)

У картці можна побачити інформацію про лід, про клієнта, статус, блок з коментарями, блок з товарами або послугами, блок з оплатами, кнопка «Відмінити лід» (зміна статусу на «Відмінені»), кнопка «Видалити лід». У розгорнутій картці можна редагувати дані (рис. 2.25 – рис. 2.26), додавати, видаляти та редагувати коментарі (рис. 2.27 – рис. 2.28). Також можна додавати, редагувати, видаляти товари та оплати.

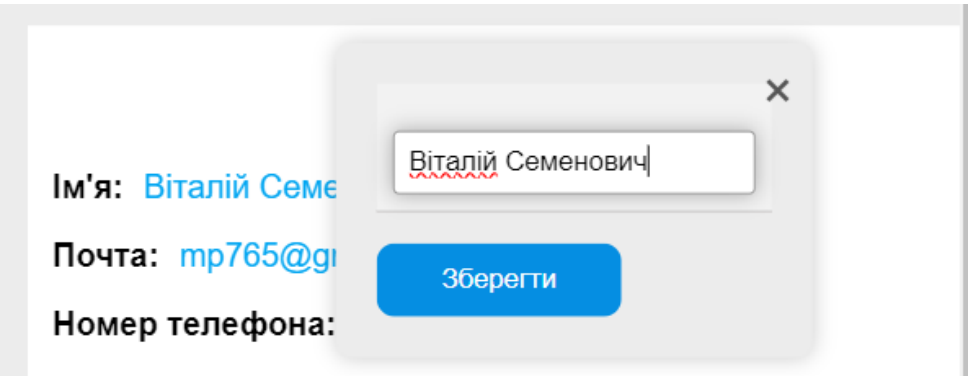


Рис. 2.25. Редагування ім’я

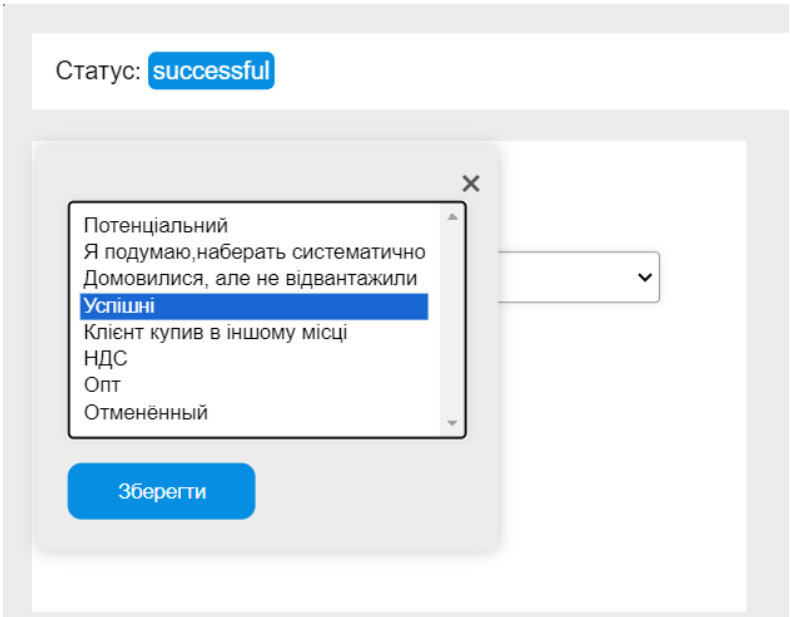


Рис. 2.26. Редагування статусу

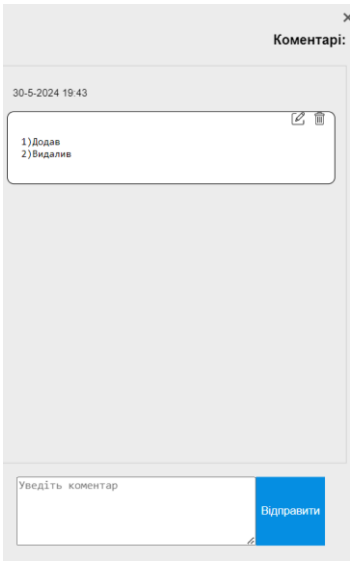


Рис. 2.27. Доданий коментар

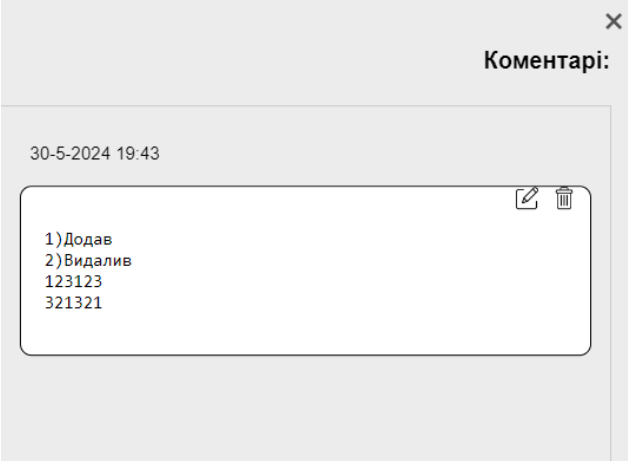


Рис. 2.28. Редагований коментар

Натиснувши кнопку «Додати товар», відкривається модальне вікно зі списком товарів (рис. 2.29), які попередньо додаються у вкладці «Товари»

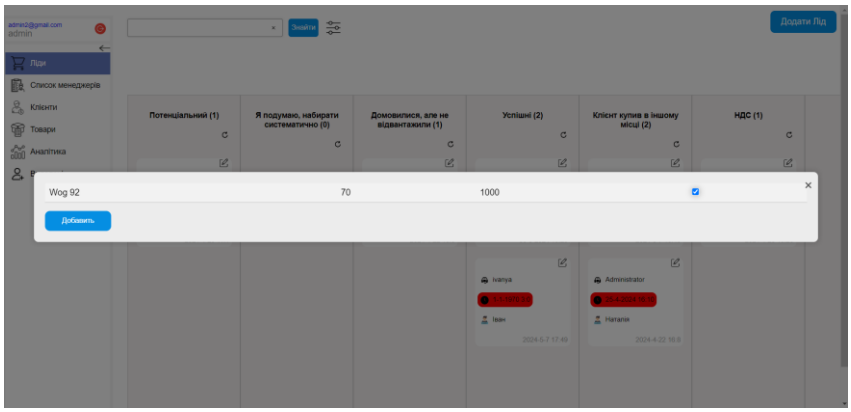


Рис. 2.29. Додавання товару

Додані товари відображаються у вкладці «Товари / Послуги» (рис. 2.30).

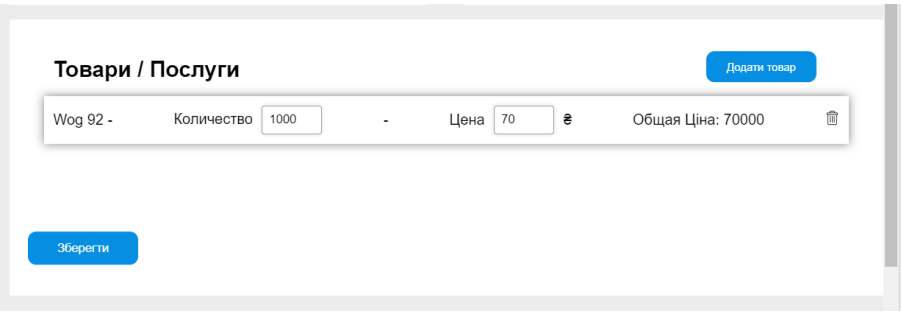


Рис. 2.30. Додані товари

Натиснувши кнопку «Додати оплату» відкривається меню для додавання оплати (рис. 2.31).

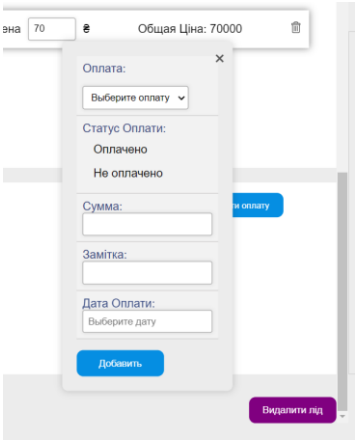


Рис. 2.31. Меню додавання оплати

Додані оплати відображаються у вкладці «Оплати» (рис. 2.32).

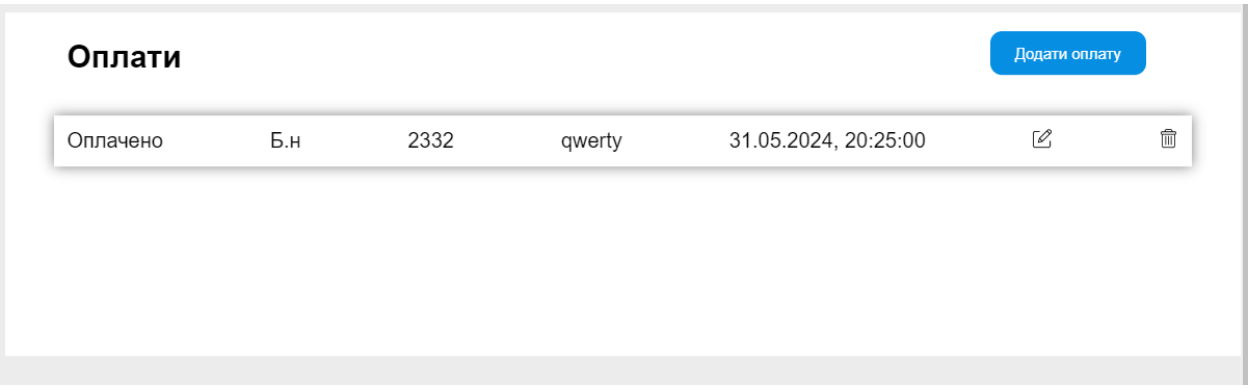


Рис. 2.32 . Додані оплати

Вкладка «Список менеджерів». Натиснувши кнопку «Список менеджерів» на боковому меню, відкривається блок з формою для додавання нових менеджерів та список вже доданих (рис. 2.33).

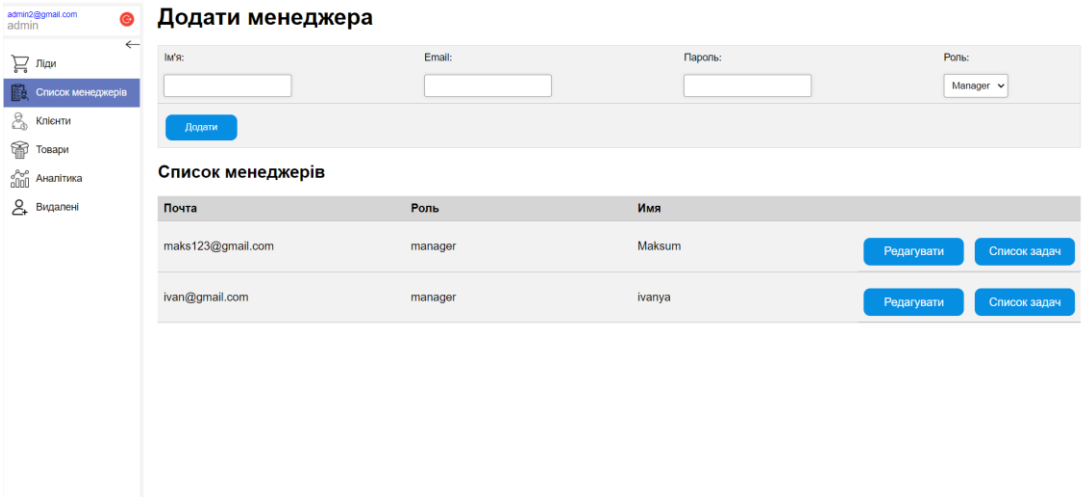


Рис. 2.33. Блок «Список менеджерів»

Натиснувши кнопку «Редагувати», відкривається модальне вікно, де можна редагувати дані відповідного менеджера та додати йому задачу (рис 2.34 – рис. 2.35).

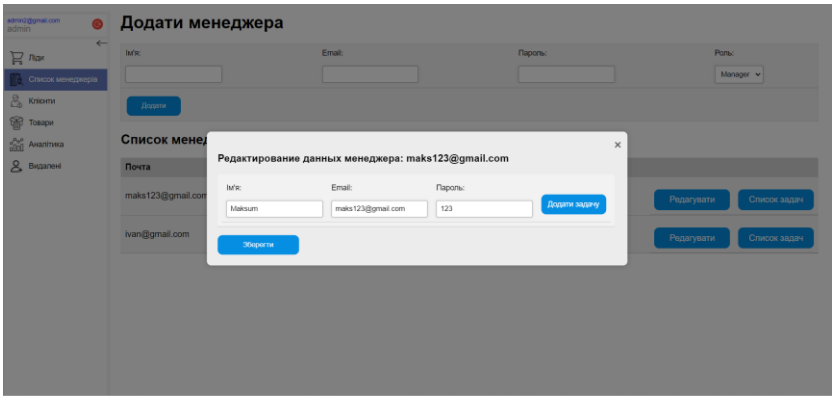


Рис. 2.34. Вікно редагування менеджера

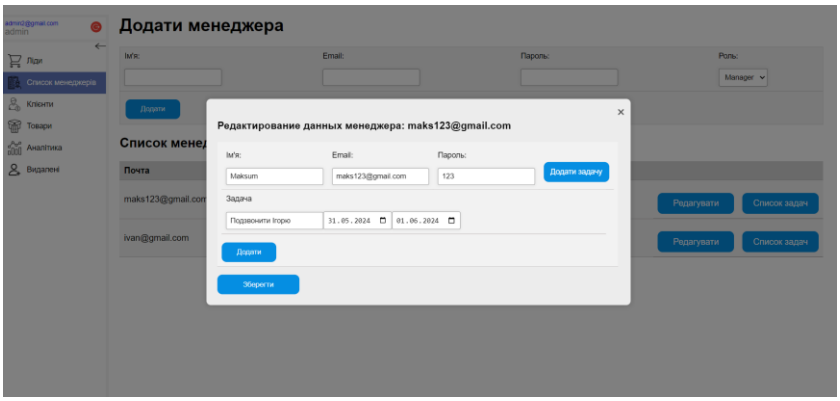


Рис. 2.35. Додавання задачі

Натиснувши кнопку «Список задач» відкривається вікно, де, обравши період, можна побачити список поставлених задач (рис. 2.36).

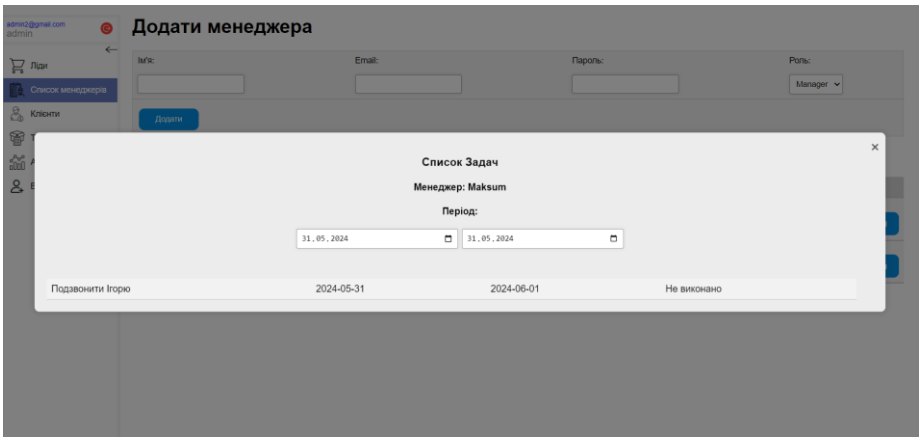


Рис. 2.36. Вікно зі списком задач

Вкладка «Клієнти». Натиснувши кнопку «Клієнти» на боковому меню, відкривається блок з формою для додавання нових клієнтів та список вже доданих (рис. 2.37).

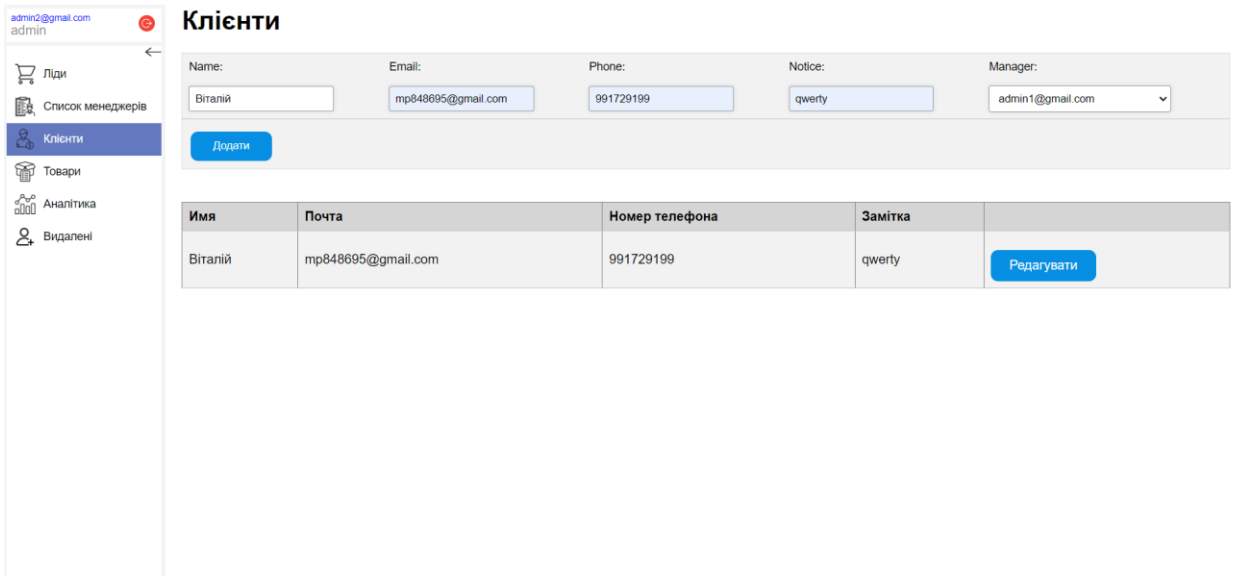


Рис. 2.37. Блок «Клієнти»

Натиснувши кнопку «Редагувати», відкривається модальне вікно, де можна редагувати дані відповідного клієнта (рис 2.38).

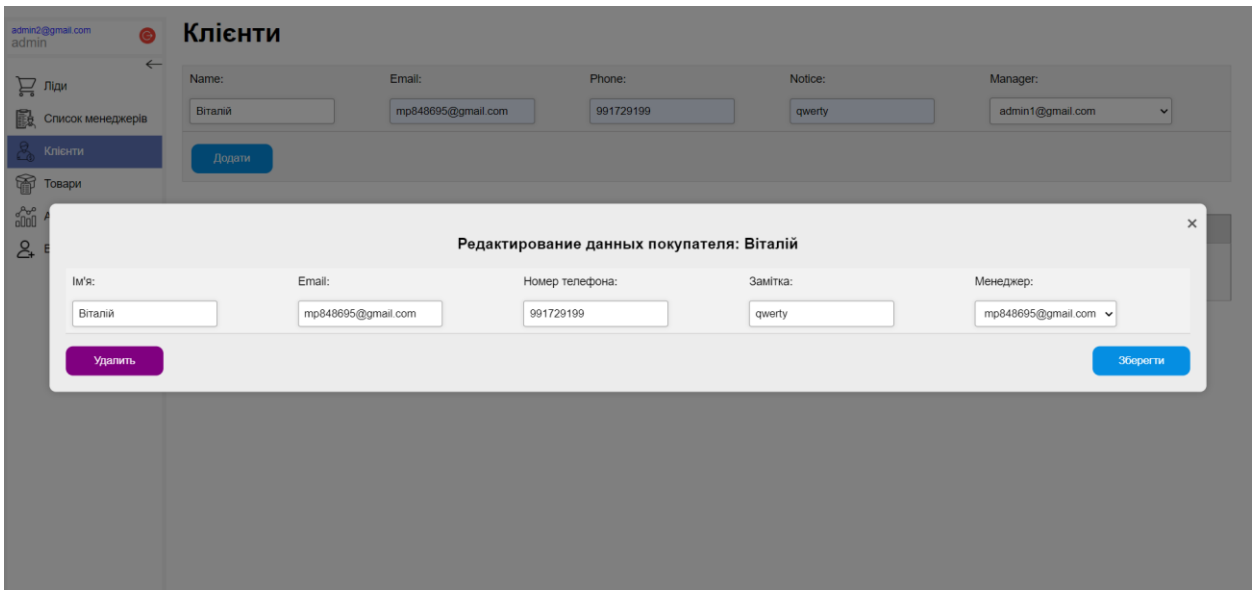


Рис. 2.38. Вікно редагування клієнта

Вкладка «Товари». Натиснувши кнопку «Товари» на боковому меню, відкривається блок з формою для додавання нових товарів та список вже доданих (рис. 2.39).

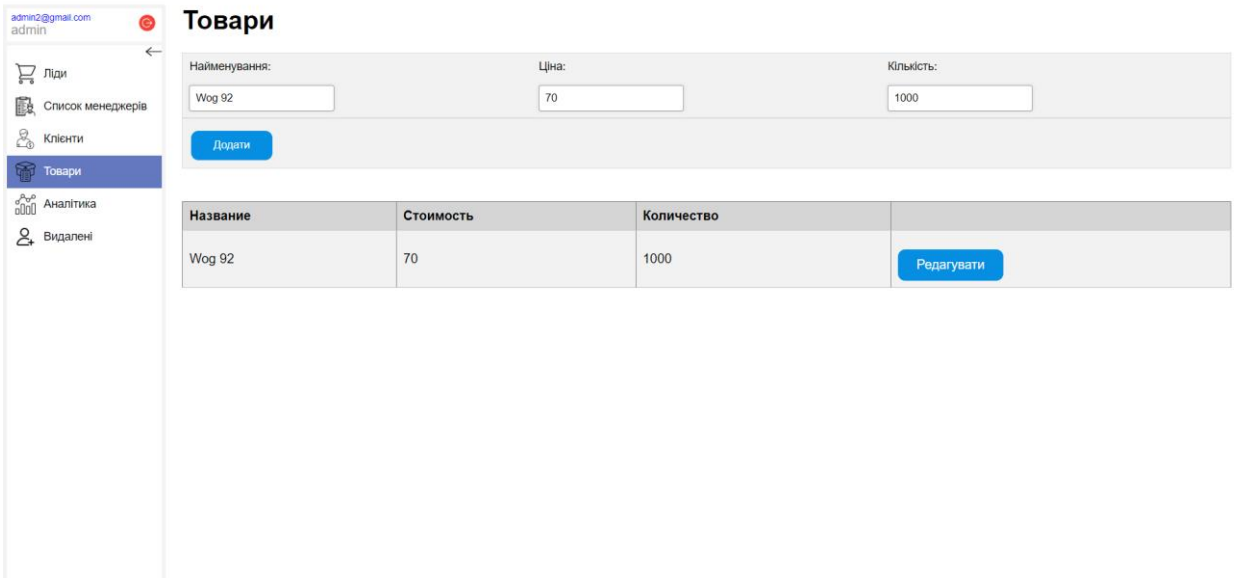


Рис. 2.39. Блок «Товари»

Натиснувши кнопку «Редагувати», відкривається модальне вікно, де можна редагувати дані відповідного товару (рис 2.40).

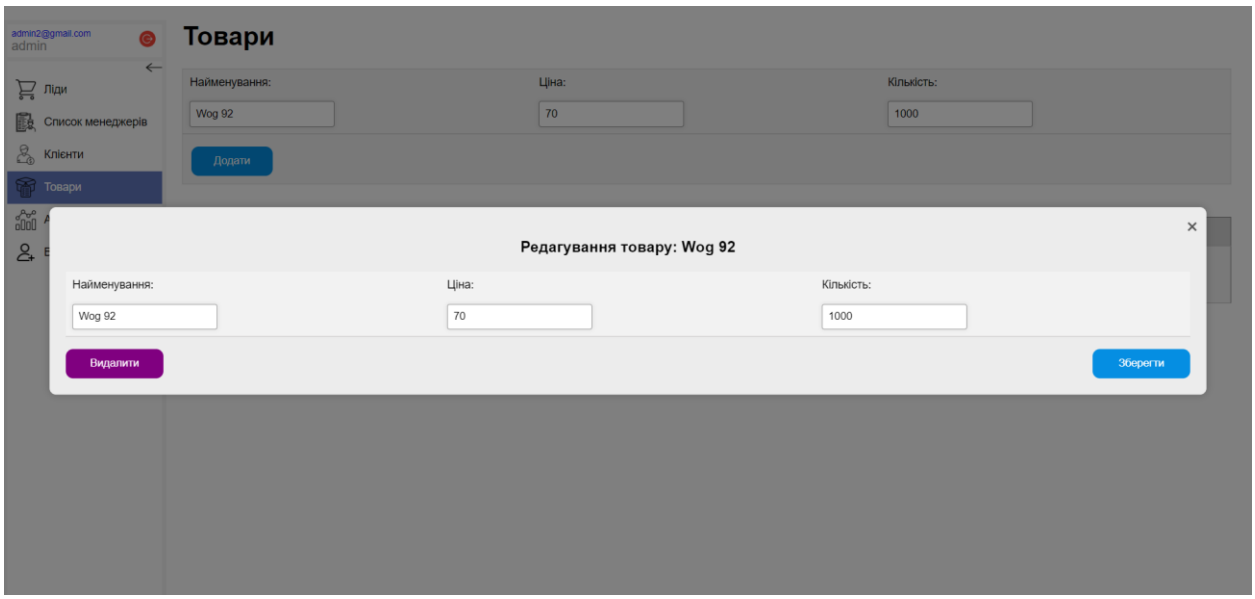


Рис. 2.40. Вікно редагуванн товару

Вкладка «Товари». Натиснувши кнопку «Аналітика» на боковому меню, відкривається блок з аналітичною діаграмою (рис. 2.41), яка відображає кількість видалених карток клієнтів за статусами

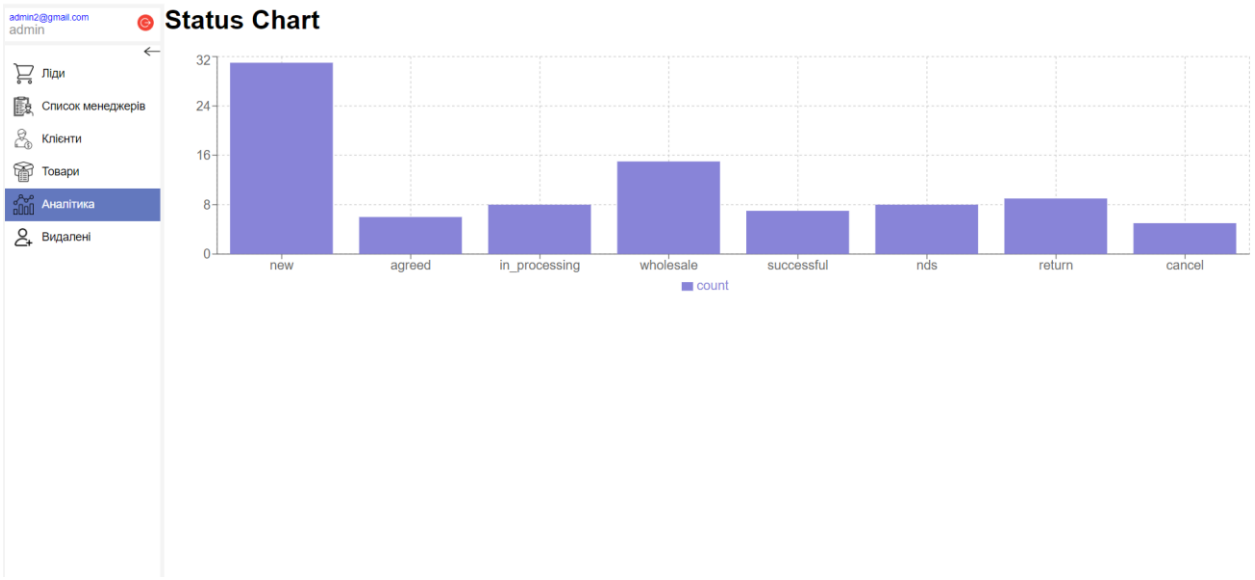


Рис. 2.41. Блок «Аналітика»

Вкладка «Видалені». Натиснувши кнопку «Видалені» на боковому меню, відкривається блок зі списком видалених карток клієнтів (рис. 2.42)

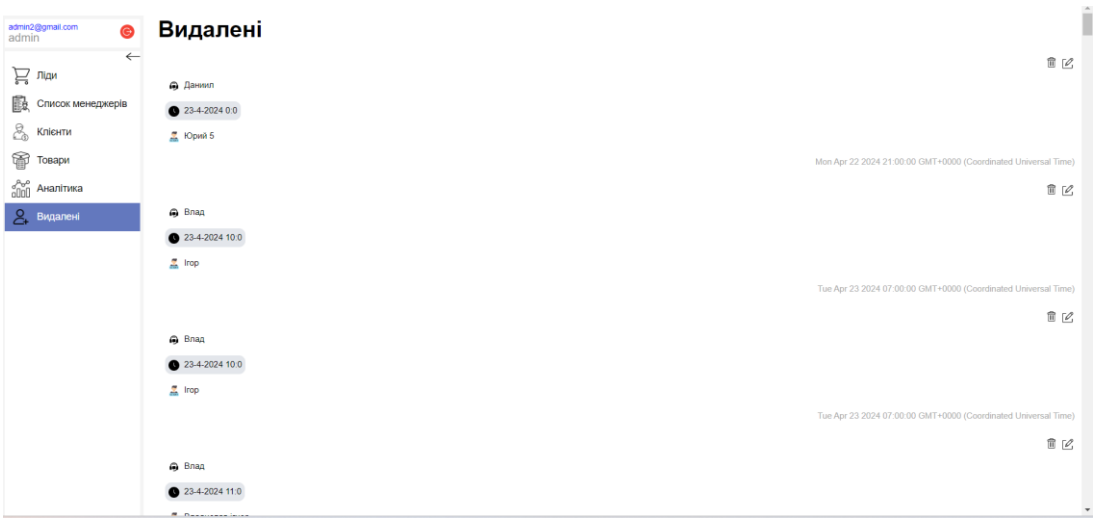


Рис. 2.42. Блок «Видалені»

Сторінка менеджера. Вигляд сторінки менеджера відображено на рис. 2.43.

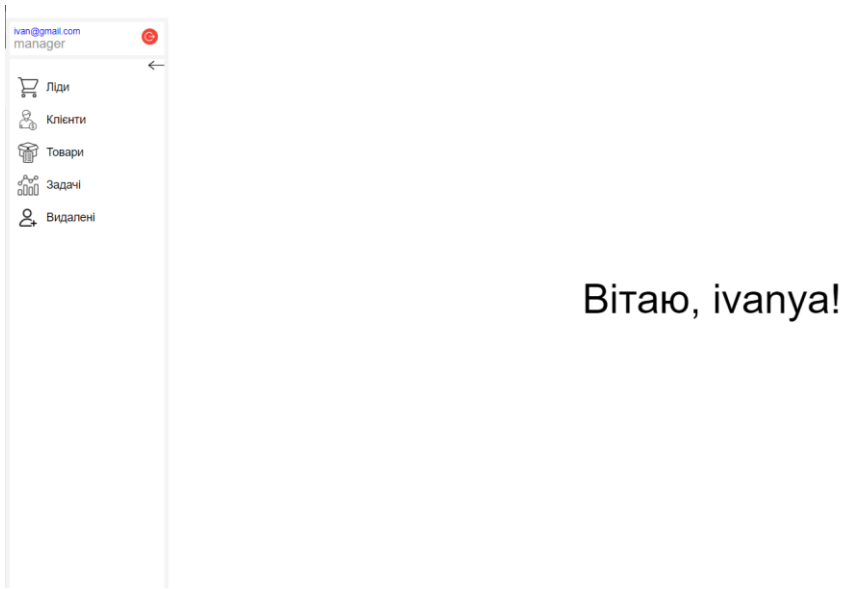


Рис. 2.43 – Головна сторінка менеджера

Як можна побачити на рис. 2.43, сторінка менеджера відрізняється відсутністю блоків «Список менеджерів» і «Аналітика» та з’являється блок «Задачі» (рис. 2.44)


Рис. 2.44. Блок «Задачі»

На ньому можна побачити встановлені задачі адміністратором. Також є можливість додати собі задачу самому.

# РОЗДІЛ 3
# ЕКОНОМІЧНИЙ РОЗДІЛ

## 3.1 Розрахунок трудомісткості та вартості розробки програмного продукту

Початкові дані:

1) Передбачуване число операторів програми: 2067;

2) Коефіцієнт складності програми: 1,5;

3) Коефіцієнт корекції програми в ході розробки: 0,4;

4) Годинна заробітна плата розробника – 229,9 ₴/год.

Згідно зі статистичними даними платформи «DOU» [17], середня заробітна плата Junior Software Engineer (SE) в Україні станом на грудень 2023 року становить 950 доларів США. На 16 червня 2024 року, згідно з офіційним курсом валют Національного банку України [18], 1 долар США дорівнює 40,59 гривень. Зазвичай графік роботи програмістів включає п'ятиденний робочий тиждень з восьмигодинним робочим днем.

Погодинну заробітну плату програміста можна розрахувати наступним чином: 950 доларів США ділимо на (21 робочий день * 8 годин), що дорівнює 5,65 доларів США на годину. В гривнях це становить 5,65 доларів США * 40,59 гривень за долар, що дорівнює 229,3 гривень на годину.

5)Коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі: 1,3;

6) Коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності: 1;

7) Вартість машино-години ЕОМ: 3,8 грн/год;

Станом на 15 червня 2024 року, вартість електроенергії становить 4,32 гривні за кіловат-годину [19]. Проект кваліфікаційної роботи розроблявся на портативному персональному комп'ютері (ноутбуці). У середньому, сучасні ноутбуки з 15-дюймовим екраном споживають близько 60 Ватт на годину, тому

витрати на електроенергію складають: 0,06 кВт * 4,32 грн/кВт·год = 0,26 грн/год.

Згідно з тарифним планом Київстар [20], витрати на інтернет становлять 350 гривень на місяць. Загальна вартість машино-години електронно-обчислювальної машини (ЕОМ) визначена як 4,5 грн/год.

Нормування праці в процесі створення ПЗ істотно ускладнено в силу творчого характеру праці програміста. Тому трудомісткість розробки ПЗ може бути розрахована на основі системи моделей з різною точністю оцінки.

Трудомісткість розробки ПЗ можна розрахувати за формулою:

$$t = t_o + t_u + t_a + t_n + t_{отл} + t_д \text{, людино-годин,} \qquad (3.1)$$

де $t_o$- витрати праці на підготовку й опис поставленої задачі (приймається 50 людино-годин);

$t_u$ - витрати праці на дослідження алгоритму рішення задачі;

$t_a$- витрати праці на розробку блок-схеми алгоритму;

$t_n$-витрати праці на програмування по готовій блок-схемі;

$t_{отл}$-витрати праці на налагодження програми на ЕОМ;

$t_д$ - витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у програмному забезпеченні, яке розробляється.

Умовне число операторів (підпрограм):

$$Q = q \cdot C \cdot (1 + p) \qquad (3.2)$$

де $q$ - передбачуване число операторів;

$C$ - коефіцієнт складності програми;

$p$ - коефіцієнт корекції програми в ході її розробки.

Визначимо значення умовного числа операторів в програмі:

$$Q = 2067 * 1,5 * (1 + 0,4) = 4340,7$$

Витрати праці на вивчення опису задачі $t_u$ визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{Q \cdot B}{(75..85) \cdot k} \text{ , людино-годин} \qquad (3.3)$$

де $B$ - коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі;

$k$ - коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності.

Визначимо значення витрат праці на вивчення опису задачі $t_u$:

$$t_u = (4340,7 * 1,2) / 85 = 61,2 \text{ людино-годин}$$

Витрати праці на розробку алгоритму рішення задачі визначаються так:

$$t_a = \frac{Q}{(20\ldots25) \cdot k} \text{ , людино-годин} \qquad (3.4)$$

де $Q$ – умовне число операторів програми;

$k$ – коефіцієнт кваліфікації програміста.

Визначимо значення витрат праці на розробку алгоритму рішення задачі:

$$t_a = 4340,7 / (25 * 1) = 173,6 \text{ людино-годин.}$$

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20\ldots25)\cdot k}\ \text{, людино-годин} \qquad (3.5)$$

Визначимо значення витрат на складання програми по готовій блок-схемі:

$t_n$ = 4340,7 / (25 * 1) = 173,6 людино-годин.

Витрати праці на налагодження програми на ЕОМ:

за умови автономного налагодження одного завдання:

$$t_{отл} = \frac{Q}{(4..5)\cdot k}\ \text{, людино-годин} \qquad (3.6)$$

Визначимо значення витрат праці на налагодження програми на ЕОМ за умови автономного налагодження одного завдання

$t_{отл}$ = 4340,7 / (5*1) = 868,14 людино-годин.

за умови комплексного налагодження завдання:

$$t^k_{отл} = 1{,}5\cdot t_{отл}\ \text{, людино-годин} \qquad (3.7)$$

Визначимо значення витрат праці на налагодження програми на ЕОМ за умови комплексного налагодження завдання

$t^k_{отл}$= 1,5 · 868,14 = 1302,21 людино-годин.

Витрати праці на підготовку документації визначаються за формулою:

$$t_д = t_{др} + t_{до}\ \text{, людино-годин} \qquad (3.8)$$

де $t_{др}$-трудомісткість підготовки матеріалів і рукопису:

$$t_{др} = \frac{Q}{(15..20) \cdot k}\ \text{, людино-годин} \qquad (3.9)$$

$t_{до}$ - трудомісткість редагування, печатки й оформлення документації:

$$t_{до} = 0{,}75 \cdot t_{др}\ \text{, людино-годин.} \qquad (3.10)$$

Визначимо наступні значення:

$t_{др}$ = 4340,7 / (20 * 1) = 217,03 людино-годин.

$t_{до}$ = 0,75 · 217,03 = 162,77 людино-годин.

Визначимо значення витрат праці на підготовку документації

$t_д$ = 217,03 + 162,77 = 379,8 людино-годин.

Повертаючись до формули (3.1), визначимо значення повної оцінки трудомісткості розробки програмного забезпечення:

$t$ = 50 + 61,2 + 173,6 + 173,6 + 868,14 + 379,8 = 1706,34 людино-години.

### 3.2. Розрахунок витрат на створення програмного забезпечення

Витрати на створення ПЗ $К_{ПО}$ включають витрати на заробітну плату виконавця програми $З_{ЗП}$ і витрат машинного часу, необхідного на налагодження програми на ЕОМ:

$$К_{ПО} = З_{ЗП} + З_{МВ}, \text{ грн} \quad (3.11)$$

Заробітна плата виконавців $З_{ЗП}$ визначається за формулою:

$$З_{ЗП} = t \cdot С_{ПР}, \text{ грн} \quad (3.12)$$

де: $t$ - загальна трудомісткість, людино-годин;

$С_{ПР}$ - середня годинна заробітна плата програміста, грн/година

Розрахуємо заробітну плату виконавців:

$З_{ЗП}$ = 1706,34 * 229,9 = 392 287 грн.

Вартість машинного часу, необхідного для налагодження програми на ЕОМ, визначається за формулою:

$$З_{мв} = t_{отл} \cdot С_{мч}, \text{ грн}, \quad (3.13)$$

де $t_{отл}$ - трудомісткість налагодження програми на ЕОМ, год;

$С_{мч}$ - вартість машино-години ЕОМ, грн/год.

Визначимо значення вартості машинного часу, необхідного для налагодження програми на ЕОМ:

$З_{мв}$ = 853,71 * 3,8 = 3 244,1 грн.

Визначимо значення витрат на створення ПЗ:

$$K_{ПО} = 387\,599 + 3\,244{,}1 = 390\,843{,}1 \text{ грн.}$$

Очікуваний період створення ПЗ:

$$T = \frac{t}{B_k \cdot F_p}, \text{ міс.} \quad (3.14)$$

де $B_k$ - число виконавців;

$F_p$ - місячний фонд робочого часу (176 годин).

Очікуваний період створення ПЗ:

$$T = 1706{,}34 / (1 * 176) \approx 9{,}69 \text{ міс.}$$

Висновки за розділом. Для розробки CRM-системи для керування клієнтською та продуктовою базою компанії за допомогою React та Node.js загальна трудомісткість становить 1706,34 людино-години. При стандартному робочому графіку, ща становить 40 годин на тиждень, очікуваний період розробки ПЗ складає приблизно 9,69 місяців. Загальні витрати на створення ПЗ складають 392287 грн.

# ВИСНОВКИ

Метою кваліфікаційної роботи є розробка CRM-системи для керування клієнтської та продуктової бази компанії з використанням React та Node.js , що дозволить компанії управляти взаємодією з клієнтами, збирати та аналізувати інформацію про них.

Актуальність теми впровадження CRM-систем обумовлюється зростаючою потребою підприємств у підвищенні ефективності взаємодії з клієнтами. У сучасних умовах глобалізації та діджиталізації бізнесу, компанії змушені шукати нові шляхи для утримання клієнтів та залучення нових.

Розроблена інформаційна система являє собою CRM-систему, яка дозволяє керувати клієнтською та продуктовою базою, надає можливість автоматизувати рутинні задачі та аналізувати дані.

У процесі розробки проекту кваліфікаційної роботи були виконані наступні задачі:

1) Проаналізовано існуючий приклад подібної системи;
2) Розроблена архітектура та алгоритми функціонування програми;
3) Обгрунтовано обрані мови програмування та технології;
4) Створено базу даних системи;
5) Розроблено інтерфейс CRM-системи

У процесі розробки були використані наступні технології та програмні засоби: інтерфейс було розроблено за допомогою бібліотеки React, HTML, CSS; серверна частина написана за допомогою фреймворку web-додатків для Node.js – Express.js; у якості сховища даних була обрана база даних MongoDB.

Провівши небхідні розрахунки було визначено, що час для створення CRM-системи, розробленої у рамках даної кваліфікаційної роботи, при стандартному графіку роботи розробника в Україні дорівнює 9,69 міс. Витрати на створення програмного забезпечення становлять 390 843,1 грн.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Kyle Simpson. You Don't Know JS: Up & Going. — O'Reilly Media, 2015. — 102 с. (дата звернення: 07.06.2024).
2. Рейтинг мов програмування. URL: https://dev.ua/ru/news/reitynh-mov-prohramuvannia-2023-1676881424 (дата звернення: 07.06.2024).
3. Що таке БД NoSQL. URL: https://freehost.com.ua/ukr/faq/wiki/chto-takoe-bazi-dannih-nosql/ (дата звернення: 08.06.2024).
4. ReactJS Tutorial. URL: https://legacy.reactjs.org/tutorial/tutorial.html (дата звернення: 08.06.2024).
5. What is MongoDB?. URL: https://www.mongodb.com/docs/manual/ (дата звернення: 08.06.2024).
6. Express - fast, unopinionated, minimalist web framework for Node.js. URL: https://expressjs.com/ (дата звернення: 08.06.2024).
7. Mongoose middleware. URL: https://mongoosejs.com/docs/middleware.html (дата звернення: 08.06.2024).
8. MongooseJS guide. URL: https://mongoosejs.com/docs/guide.html (дата звернення: 09.06.2024).
9. HTTP методи запиту. URL: https://w3schoolsua.github.io/tags/ref_httpmethods.html#gsc.tab=0 (дата звернення: 09.06.2024).
10. Using the Fetch API. URL: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch (дата звернення: 09.06.2024).
11. MVC: Model, View, Controller. URL: https://www.codecademy.com/article/mvc (дата звернення: 10.06.2024).
12. WebSocket server. URL: https://developer.mozilla.org/ru/docs/Web/API/WebSocket (дата звернення: 10.06.2024).
13. Visual Studio Code. URL: https://code.visualstudio.com/docs (дата звернення: 10.06.2024).

14. What is NodeJS?. URL: https://nodejs.tutorial.in.ua/what-is-nodejs/ (дата звернення: 10.06.2024).
15. MongoDB Atlas. URL: https://www.mongodb.com/docs/atlas/ (дата звернення: 10.06.2024).
16. MongoDB Tools. URL: https://www.mongodb.com/docs/database-tools/ (дата звернення: 11.06.2024).
17. Зарплати українських розробників — зима 2024. URL: https://dou.ua/lenta/articles/salary-report-devs-winter-2024/. (дата звернення: 14.06.2024).
18. Офіційний курс гривні щодо іноземних валют. URL: https://bank.gov.ua/ua/markets/exchangerates (дата звернення: 14.06.2024).
19. Тарифи на електроенергію для побутових споживачів з 01.06.2024 р. URL: https://yasno.com.ua/news/all_news/electricity-tariffs-for-household-consumers-from-01-06-2024 Стаж коеф 0,8 (дата звернення: 14.06.2024).
20. Київстар. URL: https://kyivstar.ua/home-internet (дата звернення: 14.06.2024).

# ДОДАТОК А

## Код програми

```
adminPage.js // Сторінка адміністратора
// Імпорт необхідних бібліотек та іншиш елементів
import { useState, useEffect } from "react";
import BuyersBlock from "./BuyerBlock";
import ProductBlock from "./ProductBlock";
import ClientsBlock from "./ClientsBlock";
import DeletedClientsList from "./DeletedClientsListBlock";
import EditManagerForm from "../../../forms/EditManagerForm";
import { fetchDeleteLeed, fetchDeleteNotice, fetchDataById, fetchDataByIdNotice, handleEditData, registerManager, registerClient, registerBuyer, registerProduct, formatDateTime, formatDateTimeTomorrow, formatDateTimeYet, handleDeleteData, registerNotice, registerTask, fetchDataByIdTask } from "../../../common/EditData";
import LeftMenu from "../../../common/LeftMenu";
import handleClientClicks from '../../../common/Click'
import "react-datepicker/dist/react-datepicker.css";
import { logout, handleEditClient } from "../../../common/methods";
import handleSearch from '../../../common/Search'
import Overlay from "./Overlay";
import Spinner from "../../../../services/Spinner";
import AnalyticsImg from "../../../../img/analytics.png"
import AddUserImg from "../../../../img/addUser.png"
import BuyerImg from "../../../../img/buyer.png"
import LeedImg from "../../../../img/leed.png"
import ListImg from "../../../../img/list.png"
import ProductListImg from "../../../../img/productList.png"
import LeaveImg from "../../../../img/leave.png"
import ClockImg from '../../../../img/clock.png'
import ClientImg from '../../../../img/client.png'
import ManagerImg from '../../../../img/manager.png'
import EditImg from '../../../../img/edit.png'
import HideImg from '../../../../img/hide.png'
import './adminPageStyle.css'
import {  getDataNew, getManagers, getBuyers, getProducts, getNotices, getCloseClients, getTasks } from "../../../../services/gettingData";
import StatusChart from './Diagram';
import { v4 as uuidv4 } from 'uuid';
export const wsConnection = new WebSocket("ws://localhost:5000");
// Ініціалізація серверу WebSocket
wsConnection.onopen = function() {
  alert("Соединение установлено.");
};

wsConnection.onclose = function(event) {
  if (event.wasClean) {
    alert('Соединение закрыто чисто');
  } else {
    alert('Обрыв соединения'); // например, "убит" процесс сервера
  }
  alert('Код: ' + event.code + ' причина: ' + event.reason);
};

wsConnection.onerror = function(error) {
  alert("Ошибка " + error.message);
};

export const wsSend = function(data) {
// readyState - true, если есть подключение
  if(!wsConnection.readyState){
```

```
        setTimeout(function (){
            wsSend(data);
        },100);
    } else {
        wsConnection.send(data);
    }
};

const AdminPage = () => {

  const overlays = [

  ];
// Витягування даних з localStorage
const storedAdminKey = localStorage.getItem('adminKey');
  const storedUserName = localStorage.getItem('userName')
  const storedUserRole = localStorage.getItem('userRole')
  const storedUser = localStorage.getItem('user')
  const currentDate = new Date();
  const formattedDateTime = formatDateTime(currentDate);
  const formatDateTomorrow = formatDateTimeTomorrow(currentDate)
  const formatDateYet = formatDateTimeYet(currentDate)
// Створення необхідних станів програми
  const [loadedItems, setLoadedItems] = useState(100000);
  const [loadedItemsAgreed, setLoadedItemsAgreed] = useState(100000);
  const [loadedItemsInProcessing, setLoadedItemsInProcessing] = useState(100000);
  const [loadedItemsCancel, setLoadedItemsCancel] = useState(100000);
  const [loadedItemsSuccesful, setLoadedItemsSuccesful] = useState(100000);
  const [loadedItemsReturn, setLoadedItemsReturn] = useState(100000);
  const [loadedItemsNds, setLoadedItemsNds] = useState(100000);
  const [loadedItemsWholesale, setLoadedItemsWholesale] = useState(100000);
  const [dataLoaded, setDataLoaded] = useState(false);
  const [statusFilter, setStatusFilter] = useState('new');
  const [statusFilterAgreed, setStatusFilterAgreed] = useState('agreed');
  const [statusFilterInProcessing, setStatusFilterInProcessing] = useState('in_processing')
  const [statusFilterCancel, setStatusFilterCancel] = useState('cancel')
  const [statusFilterSuccesful, setStatusFilterSuccesful] = useState('successful')
  const [statusFilterReturn, setStatusFilterReturn] = useState('return')
  const [statusFilterNds, setStatusFilterNds] = useState('nds')
  const [statusFilterWholesale, setStatusFilterWholesale] = useState('wholesale')
  const [checkedProducts, setCheckedProducts] = useState([]);
  const [products, setProducts] = useState([])
  const [activeMenuItem, setActiveMenuItem] = useState(null);
  const [hideLeftMenu, setHideLeftMenu] = useState(false)
  const [adminKey, setAdminKey] = useState(null);
  const [userName, setUserName] = useState(null)
  const [userRole, setUserRole] = useState(null)
  const [users, setUsers] = useState([]);
  const [ManagerStatusOptions, setManagerStatusOptions] = useState([])
  const [managerIDOptions, setManagerIDOptions] = useState([]);
  const [paymentArray, setPaymentArray] = useState([])
  const [filterBlock, setFilterBlock] = useState(false)
  const [data, setData] = useState(null);
  const [user, setUser] = useState(null)
  const [dataManagers, setDataManagers] = useState(null)
  const [dataBuyers, setDataBuyers] = useState(null)
  const [dataProducts, setDataProducts] = useState(null)
  const [dataNotices, setDataNotices] = useState(null)
  const [closeClients, setCloseClients] = useState(null)
  const [dataTasks, setDataTasks] = useState(null)
  const [selectTask, setSelectTask] = useState([])
  const [notFind, setNotFind] = useState(false)
  const [leedNotice, setLeedNotice] = useState([])
```

```
const [sortActivated, setSortActivated] = useState(false);
const [sortedData, setSortedData] = useState([]);
const [startDate, setStartDate] = useState(null);
const [endDate, setEndDate] = useState(null);
const [notMyClient, setNotMyClient] = useState([]);
const [myClient, setMyClient] = useState([]);
const [numberPhone, setNumber] = useState("");
const [availability, setAvailability] = useState();
const [showClients, setShowClients] = useState(true);
const [selectedManager, setSelectedManager] = useState(null);
const [selectedClient, setSelectedClient] = useState(null);
const [selectedBuyer, setSelectedBuyer] = useState(null);
const [selectedProduct, setSelectedProduct] = useState(null);
const [selectedNotice, setSelectedNotice] = useState(null);
const [startDateFilter, setStartDateFilter] = useState(null);
const [startDateFilterLoad, setStartDateFilterLoad] = useState(null);
const [endDateFilter, setEndDateFilter] = useState(null);
const [startBlock, setStartBlock] = useState(true)
const [myDataLeed, setMyDataLeed] = useState([])
const [editMode, setEditMode] = useState(false);
const [editClientMode, setEditClientMode] = useState(false);
const [editClientViewMode, setEditClienViewtMode] = useState(false);
const [addLeedMode, setAddLeedMode] = useState(false)
const [editTaskMode, setEditTaskMode] = useState(false)
const [editBuyerMode, setEditBuyerMode] = useState(false);
const [editProductMode, setEditProductMode] = useState(false);
const [editNoticeMode, setEditNoticeMode] = useState(false);
const [taskToday, setTaskToday] = useState(false);
const [managersList, setManagersList] = useState(false)
const [showBuyerBlock, setShowBuyerBlock] = useState(false)
const [showProductBlock, setShowProductBlock] = useState(false)
const [showAnalyticBlock, setShowAnalyticBlock] = useState(false)
const [showDeletedBlock, setShowDeletedBlock] = useState(false)
const [editClientModeMobile, setEditClientModeMobile] = useState(false)
const [editClientModeEmail, setEditClientModeEmail] = useState(false)
const [editClientModeName, setEditClientModeName] = useState(false)
const [editClientModeDate, setEditClientModeDate] = useState(false)
const [editClientModeStatus, setEditClientModeStatus] = useState(false)
const [editClientModeManager, setEditClientModeManager] = useState(false)
const [editClientModeProduct, setEditClientModeProduct] = useState(false)
const [editClientModePayment, setEditClientModePayment] = useState(false)
const [editClientModePaymentCost, setEditClientModePaymentCost] = useState(false)
const [editClientModeNotice, setEditClientModeNotice] = useState(false)
const [editClientModeSecondPhone, setEditClientModeSecondPhone] = useState(false)

const [registrationData, setRegistrationData] = useState({
  nameManager: '',
  email: '',
  password: '',
  role: 'manager',
  managerKey: storedAdminKey,

});
const [registrationDataClient, setRegistrationDataClient] = useState({
  _id: '',
  email: '',
  phone: '',
  secondPhone: '',
  role: 'client',
  managerID: storedAdminKey,
  managerKey: storedAdminKey,
  status: 'new',
  product: [],
```

```
    payment: [],
    notice: '123',
    selectedDate: '',
    dateOfCreated: `${formattedDateTime}`,
    clientName: ''
  });
  const [registrationDataBuyer, setRegistrationDataBuyer] = useState({
    email: '',
    phone: '',
    name: '',
    notice: '',
    role: 'buyer',
    managerID: '',
    managerKey: storedAdminKey,
  });
  const [registrationDataProduct, setRegistrationDataProduct] = useState({
    name: '',
    cost: '',
    count: '',
  });
  const [registrationDataNotice, setRegistrationDataNotice] = useState({
    noticeID: '',
    content: '',
    noticeDate: '',
  });

  const [registrationDataTask, setRegistrationDataTask] = useState({
    startDate: '',
    endDate: '',
    createdDate: `${formattedDateTime}`,
    managerID: '',
    managerKey: storedAdminKey,
    taskLine: '',
    taskStatus: 'false'
  });

 // Завантаження початкових даних
 useEffect(() => {
    setAdminKey(storedAdminKey);
    setUserName(storedUserName)
    setUserRole(storedUserRole)
    setUser(storedUser)
  }, [loadedItems, statusFilter, statusFilterAgreed, loadedItemsAgreed, statusFilterInProcessing, loadedItemsInProcessing, statusFilterCancel, loadedItemsCancel, statusFilterSuccesful, loadedItemsSuccesful, statusFilterReturn, loadedItemsReturn, statusFilterNds, loadedItemsNds, statusFilterWholesale, loadedItemsWholesale]);


  useEffect(() => {
    fetchData();
    const currenttoDate = new Date().toISOString().split('T')[0];
    setStartDateFilter(currenttoDate);
    setStartDateFilterLoad(currentDate)
    setEndDateFilter(currenttoDate);
  }, []);

  useEffect(() => {
    let wsConnection = new WebSocket("ws://localhost:5000");

    wsConnection.onmessage = function (event) {
      const message = JSON.parse(event.data);
      if (message.type === 'client_added') {
        fetchData();
```

```
    }
  };

  wsConnection.onclose = function (event) {
    // При обнаружении закрытия соединения создаем новое подключение
    wsConnection = new WebSocket("ws://localhost:5000");

    wsConnection.onmessage = function (event) {
      const message = JSON.parse(event.data);
      if (message.type === 'client_added') {
        fetchData();
      }
    };

    wsConnection.onerror = function (error) {
      console.log("Ошибка " + error.message);
    };
  };

  return () => {
    wsConnection.close();
  };
}, []);
useEffect(() => {
  const fetchData = async () => {
    try {
      const adminKey = localStorage.getItem('adminKey');
      const response = await fetch(`/api/usersByAdminKey?adminKey=${adminKey}`);

      if (response.ok) {
        const data = await response.json();
        setUsers(data);
      } else {
        console.error('Failed to fetch users:', response.statusText);
      }
    } catch (error) {
      console.error('Error fetching users:', error);
    }
  };

  fetchData();
}, []);

useEffect(() => {
  const managerIDArray = users.map((user) => user);
  setManagerIDOptions(managerIDArray);
}, [users]);

useEffect(() => {
  const managerStatusArray = ['new', 'in_processing', 'agreed', 'successful', 'return', 'nds', 'wholesale', 'cancel']
  const paymentArray = ['Б.н', 'Наличными', 'На карту']
  setPaymentArray(paymentArray)
  setManagerStatusOptions(managerStatusArray)
}, [])

const showBlock = (blockName) => {
  setActiveMenuItem(blockName);
  setShowClients(blockName === 'Ліди');
  setManagersList(blockName === 'Список менеджерів');
  setShowBuyerBlock(blockName === 'Клієнти');
  setShowProductBlock(blockName === 'Товари');
  setShowAnalyticBlock(blockName === 'Аналітика')
  setShowDeletedBlock(blockName === 'Видалені')
```

```
            setStartBlock(false)
          };
          const updateFetch = async () => {
            try {
              const responseData1 = await getDataNew(statusFilter);
              const responseDataAgreed = await getDataNew(statusFilterAgreed);
              const responseDataInProcessing = await getDataNew(statusFilterInProcessing);
              const responseDataCancel = await getDataNew(statusFilterCancel);
              const responseDataSuccesful = await getDataNew(statusFilterSuccesful);
              const responseReturn = await getDataNew(statusFilterReturn);
              const responseDataNds = await getDataNew(statusFilterNds);
              const responseWholesale = await getDataNew(statusFilterWholesale);

              if (
                Array.isArray(responseData1)


              ) {

                const responseData = [...responseData1, ...responseDataAgreed, ...responseDataInProcessing, ...responseDataCancel, ...responseDataSuccesful, ...responseReturn, ...responseDataNds, ...responseWholesale]


                setMyDataLeed(responseData.filter((person) => storedAdminKey === person.managerKey || storedAdminKey === person.managerID))
                setDataLoaded(true);
              } else {
                console.error('Some data failed to load');
              }
            } catch (error) {
              console.error('Error fetching data:', error);
            }
          };
        // Отримання даних з БД
        const fetchData = async () => {
            try {
              const responseData1 = await getDataNew(statusFilter);
              const responseDataAgreed = await getDataNew(statusFilterAgreed);
              const responseDataInProcessing = await getDataNew(statusFilterInProcessing);
              const responseDataCancel = await getDataNew(statusFilterCancel);
              const responseDataSuccesful = await getDataNew(statusFilterSuccesful);
              const responseReturn = await getDataNew(statusFilterReturn);
              const responseDataNds = await getDataNew(statusFilterNds);
              const responseWholesale = await getDataNew(statusFilterWholesale);
              const responseManagers = await getManagers();
              const responseBuyers = await getBuyers();
              const responseProducts = await getProducts();
              const responseNotices = await getNotices();
              const responseCloseClients = await getCloseClients();
              const responseTasks = await getTasks();

              if (
                Array.isArray(responseData1) &&
                // responsAllData &&
                responseManagers &&
                responseBuyers &&
                responseProducts &&
                responseNotices &&
                responseCloseClients &&
                responseTasks
              ) {
                setData((prevData) => {
```

```
            const responseData = [...responseData1, ...responseDataAgreed, ...responseDataInProcessing, ...responseDataCancel, ...responseDataSuccesful, ...responseReturn, ...responseDataNds, ...responseWholesale]

            setAllData(responsAllData)

            const existingData = Array.isArray(prevData) ? prevData : [];
            const uniqueData = responseData.filter((newItem) => !existingData.some((item) => item._id === newItem._id));

            const combinedData = [...existingData, ...uniqueData];

            const uncompletedTaskToday = combinedData.filter((task) => {
              const taskDate = new Date(task.selectedDate);
              const formattedDate = new Date(formattedDateTime);
              return taskDate < formattedDate;
            });

            const taskToday = combinedData.filter((task) => {
              const taskDate = new Date(task.selectedDate);
              const formattedDateYes = new Date(formattedDateTime);

              const formattedDate = new Date(formatDateTomorrow);
              const formattedDateYet = new Date(formatDateYet);
              return formattedDateYet < taskDate && taskDate < formattedDate;
            });

            setTaskToday(taskToday)

            console.log(uncompletedTaskToday)
            return combinedData;
          });
          const responseData = [...responseData1, ...responseDataAgreed, ...responseDataInProcessing, ...responseDataCancel, ...responseDataSuccesful, ...responseReturn, ...responseDataNds, ...responseWholesale]

          setDataManagers(responseManagers);
          setDataBuyers(responseBuyers);
          setDataProducts(responseProducts);
          setDataNotices(responseNotices);
          setCloseClients(responseCloseClients);
          setDataTasks(responseTasks)
          setMyDataLeed(responseData.filter((person) => storedAdminKey === person.managerKey || storedAdminKey === person.managerID))
          setDataLoaded(true);
        } else {
          console.error('Some data failed to load');
        }
      } catch (error) {
        console.error('Error fetching data:', error);
      }
    };

    const fetchDataReload = async (status) => {
      try {
        const response = await getDataNew(status);

        if (Array.isArray(response)) {
          setMyDataLeed(prevState => [...prevState, response[response.length - 1]]);
          setDataLoaded(true);
        } else {
          console.error('Some data failed to load');
        }
      } catch (error) {
        console.error('Error fetching data:', error);
```

```
    }
  };

// Методи видалення елементів
  const handleDeleteClient = async () => {
    const id = {
      _id: selectedClient._id
    }
    await handleDeleteData('/api/deleteClient', id, fetchData);
  };

  const handleDeleteBuyer = async () => {
    const id = {
      _id: selectedBuyer._id
    }
    await handleDeleteData('/api/deleteBuyer', id, fetchData);
  };

  const handleDeleteProduct = async () => {

    const id = {
      _id: selectedProduct._id
    }
    await handleDeleteData('/api/deleteProduct', id, fetchData);
  };
  //commit
  const handleDeleteNotice = async (id) => {

    fetchDeleteNotice(id, '/api/getNoticeData', setSelectedNotice, fetchData);

  };
  //commit
  const handleDeleteClientFully = async (id) => {

    fetchDeleteLeed(id, '/api/getCloseClientsData', setSelectedClient, fetchData);
    console.log(id)
  };

  const handleInputChange = (event) => {
    const newValue = event.target.value.trimStart();
    setNumber(newValue);
  };
  //commit
// Пошук клієнтів
const handleSearchWrapper = (event) => {
    clearing()
    if (numberPhone.trim() !== '') {
      handleSearch(event, numberPhone, adminKey, setMyClient, setNotMyClient, setNotFind);
    }

  };
 // Функція для обробки змін у формі реєстрації менеджера
const handleRegistrationChange = (e) => {
  setRegistrationData({
    ...registrationData,
    [e.target.name]: e.target.value
  });
};

// Функція для обробки змін у формі реєстрації клієнта
const handleRegistrationClientChange = (e) => {
  setRegistrationDataClient({
    ...registrationDataClient,
```

```
    [e.target.name]: e.target.value,
  });
};

// Функція для обробки змін у формі реєстрації покупця
const handleRegistrationBuyerChange = (e) => {
  setRegistrationDataBuyer({
    ...registrationDataBuyer,
    [e.target.name]: e.target.value,
  });
};

// Функція для обробки змін у формі реєстрації продукту
const handleRegistrationProductChange = (e) => {
  setRegistrationDataProduct({
    ...registrationDataProduct,
    [e.target.name]: e.target.value,
  });
};

// Функція для обробки змін у формі реєстрації повідомлення
const handleRegistrationNoticeChange = (e) => {
  setRegistrationDataNotice({
    ...registrationDataNotice,
    [e.target.name]: e.target.value,
  });
};

// Функція для обробки змін у формі реєстрації завдання
const handleRegistrationTaskChange = (e) => {
  setRegistrationDataTask({
    ...registrationDataTask,
    [e.target.name]: e.target.value,
  });
};

// Функція для реєстрації менеджера
const handleRegistration = async () => {
  await registerManager(registrationData, fetchData);
};

// Функція для реєстрації клієнта
const handleRegistrationClient = async () => {
  await registerClient({
    ...registrationDataClient,
    _id: uuidv4(),
    dateOfCreated: `${formattedDateTime}`,
    managerID: storedAdminKey,
    managerKey: storedAdminKey,
  },
    fetchDataReload, cancelSort, handleClientDoubleClick, setRegistrationDataClient, setAddLeedMode);
};

// Функція для реєстрації покупця
const handleRegistrationBuyer = async () => {
  await registerBuyer(registrationDataBuyer, fetchData);
};

// Функція для реєстрації продукту
const handleRegistrationProduct = async () => {
  await registerProduct(registrationDataProduct, fetchData);
};
```

```
// Функція для реєстрації завдання
const handleRegistrationTask = async () => {
  await registerTask(registrationDataTask, fetchData);
};

 // Функція для обробки реєстрації повідомлення
const handleRegistrationNotice = async () => {
  await registerNotice(registrationDataNotice, fetchData);
  setRegistrationDataNotice(prevData => ({
    ...prevData,
    content: ''
  }));
};

// Функція для обробки кліку по менеджеру
const handleManagerClick = async (id) => {
  fetchDataById(id, '/api/getManagerData', setSelectedManager, setEditMode, setRegistrationDataNotice);
};

// Функція для перегляду завдання
const handleViewTask = async (id) => {
  try {
    await fetchDataByIdTask(id, '/api/getManagerData', setSelectedManager, toggleTaskMode, dataTasks, setSelectTask);
  } catch (error) {
    console.error('Произошла ошибка при выполнении fetchDataById:', error);
  }
};

// Функція для обробки кліку по даним (клієнт)
const handleDataClick = async (id, setMode) => {
  fetchDataById(id, '/api/getClientData', setSelectedClient, setMode, setRegistrationDataNotice, setCheckedProducts);
}

// Функція для обробки кліку по продукту
const handleDataProductClick = async (id, setMode, setModeLeed) => {
  await fetchDataById(id, '/api/getClientData', setSelectedClient, setMode, setRegistrationDataNotice, setCheckedProducts);
}

// Функція для обробки кліку по клієнту
const handleClientClick = async (id) => {
  fetchDataById(id, '/api/getClientData', setSelectedClient, setEditClientModeMobile, setRegistrationDataNotice, setCheckedProducts);
};

// Функція для обробки подвійного кліку по клієнту
const handleClientDoubleClick = async (id) => {
  fetchDataById(id, '/api/getClientData', setSelectedClient, setEditClienViewtMode, setRegistrationDataNotice, setCheckedProducts);
};

// Функція для обробки подвійного кліку для видалення клієнта
const handleDeleteClientDoubleClick = async (id) => {
  fetchDataById(id, '/api/getCloseClientsData', setSelectedClient, setEditClienViewtMode, setRegistrationDataNotice);
};

// Функція для обробки кліку по покупцю
const handleBuyerClick = async (id) => {
  fetchDataById(id, '/api/getBuyerData', setSelectedBuyer, setEditBuyerMode, setRegistrationDataNotice);
};
```

```
// Функція для обробки кліку по продукту
const handleProductClick = async (id) => {
  fetchDataById(id, '/api/getProductData', setSelectedProduct, setEditProductMode, setRegistrationDataNotice);
};

// Функція для обробки кліку по повідомленню
const handleNoticeClick = async (id) => {
  fetchDataByIdNotice(id, '/api/getNoticeData', setSelectedNotice, setEditNoticeMode,
setRegistrationDataNotice);
};

// Функція для редагування менеджера
const handleEditManager = async () => {
  const requestData = {
    _id: selectedManager._id,
    email: selectedManager.email,
    password: selectedManager.password,
    nameManager: selectedManager.nameManager
  };

  await handleEditData('/api/updateManagerData', requestData, setEditMode, fetchData);
};

// Функція для очищення даних
const clearing = () => {
  setNotMyClient([]);
  setMyClient([]);
  setAvailability("");
  setNumber("");
  setNotFind(false)
};

// Функція для обробки змін дати
const handleDateChange = (date) => {
  if (typeof date === 'string') {
    date = new Date(date);
  }

  if (date instanceof Date && !isNaN(date)) {
    const offset = new Date().getTimezoneOffset();
    const correctedDate = new Date(date.getTime());

    setRegistrationDataClient((prevData) => ({
      ...prevData,
      selectedDate: correctedDate,
    }));
  } else {
    console.error("Invalid date object:", date);
  }
};

// Функція для переключення режиму завдання
const toggleTaskMode = () => {
  setEditTaskMode(true);
};

// Функція для закриття всіх режимів редагування
const close = () => {
  setEditClientMode(false);
  setEditBuyerMode(false);
  setEditProductMode(false);
  setEditClienViewtMode(false);
```

```
  setAddLeedMode(false);
  setEditMode(false);
  setEditTaskMode(false);
  setNotFind(false);
  setCheckedProducts([]);
  setProducts([]);
};

// Функція для додавання нового ліда
const addLeed = () => {
  setAddLeedMode(true);
};

// Функція для обробки змін початкової дати фільтру
const handleStartDateChange = (date) => {
  setStartDateFilter(date);
};

// Функція для обробки змін кінцевої дати фільтру
const handleEndDateChange = (date) => {
  setEndDateFilter(date);
};

// Функція для сортування даних за датою
const sortByDate = (data) => {
  return data.sort((a, b) => new Date(a.startDate) - new Date(b.startDate));
};

// Функція для відкриття блоку фільтрів
const openFilter = () => {
  setFilterBlock(true);
};

// Функція для закриття блоку фільтрів
const closeFilter = () => {
  setFilterBlock(false);
};

// Функція для приховування меню
const hideMenu = () => {
  setHideLeftMenu(prevState => !prevState);
};

// Функція для додавання платежу (поки що пуста)
const addPayment = () => {
  // Код для додавання платежу
};

// Функція для скасування сортування
const cancelSort = () => {
  setSortActivated(false);
  setSortedData([]);
  setStartDate(null);
  setEndDate(null);
  setActiveMenuItem(false);
  setFilterBlock(false);
};
  //commit
  const handleFormSubmit = (event) => {
    event.preventDefault();
    handleSearchWrapper();
  };
```

```
        return (
          <div className="main">
            {dataLoaded ? (
              <>
                {overlays.map((mode, index) => (
                  <Overlay key={index} mode={eval(mode)} close={close} />
                ))}
                <LeftMenu hideLeftMenu={hideLeftMenu} userName={userName} userRole={userRole}
logout={logout} LeaveImg={LeaveImg} hideMenu={hideMenu} HideImg={HideImg}
activeMenuItem={activeMenuItem} showBlock={showBlock} LeedImg={LeedImg} ListImg={ListImg}
BuyerImg={BuyerImg} ProductListImg={ProductListImg} AnalyticsImg={AnalyticsImg}
AddUserImg={AddUserImg} />

                <div className="right_content">
                  {startBlock && <>
                    <div className="hello">Вітаю, {user}!</div>
                  </>}
                  <div>

                    <div>
                      {showClients &&
                        <ClientsBlock
                          setFilterBlock={setFilterBlock}
                          fetchDataReload={updateFetch}
                          fetchData={fetchData}
                          showClients={showClients}
                          addLeedMode={addLeedMode}
                          dataBuyers={dataBuyers}
                          handleRegistrationClient={handleRegistrationClient}
                          setRegistrationDataClient={setRegistrationDataClient}
                          availability={availability}
                          notFind={notFind}
                          addLeed={addLeed}
                          openFilter={openFilter}
                          handleSearchWrapper={handleSearchWrapper}
                          clearing={clearing}
                          myClient={myClient}
                          notMyClient={notMyClient}
                          dataManagers={dataManagers}
                          // hasUncompletedTaskToday={hasUncompletedTaskToday}
                          formatDateTime={formatDateTime}
                          ManagerImg={ManagerImg}
                          ClockImg={ClockImg}
                          ClientImg={ClientImg}
                          EditImg={EditImg}
                          handleClientClicks={handleClientClicks}
                          handleFormSubmit={handleFormSubmit}
                          numberPhone={numberPhone}
                          handleInputChange={handleInputChange}
                          handleDataProductClick={handleDataProductClick}
                          setEditClienViewtMode={setEditClienViewtMode}
                          setEditClientModeNotice={setEditClientModeNotice}
                          editClientModeNotice={editClientModeNotice}
                          setEditClientModeSecondPhone={setEditClientModeSecondPhone}
                          editClientModeSecondPhone={editClientModeSecondPhone}
                          editClientModePaymentCost={editClientModePaymentCost}
                          setEditClientModePaymentCost={setEditClientModePaymentCost}
                          addPayment={addPayment}
                          setEditClientModePayment={setEditClientModePayment}
                          editClientModePayment={editClientModePayment}
                          setProducts={setProducts}
                          products={products}
                          setCheckedProducts={setCheckedProducts}
```

checkedProducts={checkedProducts}
setEditClientModeProduct={setEditClientModeProduct}
editClientModeProduct={editClientModeProduct}
handleDataClick={handleDataClick}
setEditMode={setEditMode}
setEditNoticeMode={setEditNoticeMode}
setEditClientModeEmail={setEditClientModeEmail}
setEditClientModeName={setEditClientModeName}
setEditClientModeDate={setEditClientModeDate}
setEditClientModeStatus={setEditClientModeStatus}
setEditClientModeManager={setEditClientModeManager}
setEditClientModeMobile={setEditClientModeMobile}
editClientModeManager={editClientModeManager}
editClientModeEmail={editClientModeEmail}
editClientModeName={editClientModeName}
editClientModeMobile={editClientModeMobile}
editClientModeDate={editClientModeDate}
editClientModeStatus={editClientModeStatus}
startDate={startDate}
setStartDate={setStartDate}
endDate={endDate}
setEndDate={setEndDate}
sortActivated={sortActivated}
setSortActivated={setSortActivated}
sortedData={sortedData}
setSortedData={setSortedData}
cancelSort={cancelSort}
setShowClients={setShowClients}
setMyDataLeed={setMyDataLeed}
myDataLeed={myDataLeed}
handleDeleteNotice={handleDeleteNotice}
leedNotice={leedNotice}
editNoticeMode={editNoticeMode}
setSelectedNotice={setSelectedNotice}
handleNoticeClick={handleNoticeClick}
selectedNotice={selectedNotice}
taskToday={taskToday}
setLoadedItemsCancel={setLoadedItemsCancel}
setStatusFilterCancel={setStatusFilterCancel}
setLoadedItemsWholesale={setLoadedItemsWholesale}
setStatusFilterWholesale={setStatusFilterWholesale}
setLoadedItemsNds={setLoadedItemsNds}
setStatusFilterNds={setStatusFilterNds}
setStatusFilterReturn={setStatusFilterReturn}
setLoadedItemsReturn={setLoadedItemsReturn}
setStatusFilterSuccesful={setStatusFilterSuccesful}
setLoadedItemsSuccesful={setLoadedItemsSuccesful}
setStatusFilterAgreed={setStatusFilterAgreed}
setLoadedItemsAgreed={setLoadedItemsAgreed}
setStatusFilterInProcessing={setStatusFilterInProcessing}
setLoadedItemsInProcessing={setLoadedItemsInProcessing}
setStatusFilter={setStatusFilter}
setLoadedItems={setLoadedItems}
closeFilter={closeFilter}
filterBlock={filterBlock}
handleDateChange={handleDateChange}
paymentArray={paymentArray}
data={data}
dataNotices={dataNotices}
handleRegistrationNoticeChange={handleRegistrationNoticeChange}
registrationDataNotice={registrationDataNotice}
handleRegistrationNotice={handleRegistrationNotice}
handleDeleteClient={handleDeleteClient}

```
        handleClientDoubleClick={handleClientDoubleClick}
        handleClientClick={handleClientClick}
        editClientViewMode={editClientViewMode}
        editClientMode={editClientMode}
        handleEditClient={handleEditClient}
        setSelectedClient={setSelectedClient}
        selectedClient={selectedClient}
        keyManage={adminKey}
        managerIDOptions={managerIDOptions}
        ManagerStatusOptions={ManagerStatusOptions}
        registrationDataClient={registrationDataClient}
        handleRegistrationClientChange={handleRegistrationClientChange}
        dataProducts={dataProducts}
        close={close}
       />
      }
      {showAnalyticBlock && <div >
       <h1>Status Chart</h1>
       <StatusChart data={closeClients} />
      </div>
      }

     </div>

    </div>
    <div>
     {showDeletedBlock &&
      <DeletedClientsList
       handleDeleteClientFully={handleDeleteClientFully}
       showDeletedBlock={showDeletedBlock}
       closeClients={closeClients}
       handleClientClicks={handleClientClicks}
       dataManagers={dataManagers}
       formatDateTime={formatDateTime}
       ManagerImg={ManagerImg}
       ClockImg={ClockImg}
       ClientImg={ClientImg}
       EditImg={EditImg}
       handleDeleteClient={handleDeleteClient}
       setProducts={setProducts}
       products={products}
       setCheckedProducts={setCheckedProducts}
       checkedProducts={checkedProducts}
       setEditClientModeProduct={setEditClientModeProduct}
       setEditClientModeManager={setEditClientModeManager}
       setEditClientModeStatus={setEditClientModeStatus}
       setEditClientModeDate={setEditClientModeDate}
       editClientModeProduct={editClientModeProduct}
       editClientModeManager={editClientModeManager}
       editClientModeStatus={editClientModeStatus}
       editClientModeDate={editClientModeDate}
       setEditClientModeName={setEditClientModeName}
       editClientModeName={editClientModeName}
       setEditClientModeEmail={setEditClientModeEmail}
       handleDataClick={handleDataClick}
       editClientModeEmail={editClientModeEmail}
       editClientModeMobile={editClientModeMobile}
       handleDeleteNotice={handleDeleteNotice}
       leedNotice={leedNotice}
       editNoticeMode={editNoticeMode}
       setSelectedNotice={setSelectedNotice}
       handleNoticeClick={handleNoticeClick}
       selectedNotice={selectedNotice}
```

```
                dataNotices={dataNotices}
                handleRegistrationNoticeChange={handleRegistrationNoticeChange}
                registrationDataNotice={registrationDataNotice}
                handleRegistrationNotice={handleRegistrationNotice}
                handleClientClick={handleClientClick}
                registrationDataClient={registrationDataClient}
                handleRegistrationClientChange={handleRegistrationClientChange}
                dataProducts={dataProducts}
                storedAdminKey={storedAdminKey}
                editClientViewMode={editClientViewMode}
                setSelectedClient={setSelectedClient}
                selectedClient={selectedClient}
                handleEditClient={handleEditClient}
                managerIDOptions={managerIDOptions}
                ManagerStatusOptions={ManagerStatusOptions}
                close={close}
                handleDeleteClientDoubleClick={handleDeleteClientDoubleClick}
              // hasUncompletedTaskToday={hasUncompletedTaskToday}
              />
            }
          </div>
          <div>
            {showBuyerBlock &&
              <div>
                <BuyersBlock
                  showBuyerBlock={showBuyerBlock}
                  registrationDataBuyer={registrationDataBuyer}
                  handleRegistrationBuyerChange={handleRegistrationBuyerChange}
                  dataManagers={dataManagers}
                  handleRegistrationBuyer={handleRegistrationBuyer}
                  dataBuyers={dataBuyers}
                  handleBuyerClick={handleBuyerClick}
                  editBuyerMode={editBuyerMode}
                  setSelectedBuyer={setSelectedBuyer}
                  ManagerStatusOptions={ManagerStatusOptions}
                  selectedBuyer={selectedBuyer}
                  handleDeleteBuyer={handleDeleteBuyer}
                  setEditMode={setEditMode}
                  fetchData={fetchData}
                  close={close}
                />
              </div>
            }
          </div>
          {managersList && <div style={{ marginLeft: "1.5%" }}>
            {/* //commit2 */}
            <h1>Додати менеджера</h1>

            <form enctype="multipart/form-data">
              <table className="reg_table computer">
                <tbody>
                  <tr>
                    <td  className="input_td"><div>Ім'я:</div><input  type="text"  name="nameManager"
value={registrationData.nameManager} onChange={handleRegistrationChange} required /></td>
                    <td  className="input_td"><div>Email:</div><input  type="email"  name="email"
value={registrationData.email} onChange={handleRegistrationChange} required /></td>
                    <td  className="input_td"><div>Пароль:</div><input  type="password"  name="password"
value={registrationData.password} onChange={handleRegistrationChange} required /></td>
                    <td  className="input_td"><div>Роль:</div><select  name="role"  value={registrationData.role}
onChange={handleRegistrationChange}>
                      <option value="manager">Manager</option>
                      <option value="admin">Admin</option>
                    </select>
```

```
                    </td>
                   </tr>
                  </tbody>
                  <button                 className="register                 btn_buyer"                 type="button"
onClick={handleRegistration}>Додати</button>
                 </table>

                 <table className="reg_table mobile">
                  <tbody>
                   <tr><td    className="input_td"><div>Name:</div><input    type="text"    name="nameManager"
value={registrationData.nameManager} onChange={handleRegistrationChange} required /></td></tr>
                   <tr><td       className="input_td"><div>Email:</div><input       type="email"       name="email"
value={registrationData.email} onChange={handleRegistrationChange} required /></td></tr>
                   <tr><td  className="input_td"><div>Password:</div><input  type="password"  name="password"
value={registrationData.password} onChange={handleRegistrationChange} required /></td></tr>
                   <tr><td                 className="input_td"><div>Role:</div><select                 name="role"
value={registrationData.role} onChange={handleRegistrationChange}>
                    <option value="manager">Manager</option>
                    <option value="admin">Admin</option>
                   </select>
                   </td>
                   </tr>
                  </tbody>
                  <button                 className="register                 btn_buyer"                 type="button"
onClick={handleRegistration}>Додати</button>
                 </table>


                </form>
                <h2>Список менеджерів</h2>

                <div className="table_block">
                 <table style={{ cursor: "pointer" }}>
                  <thead>
                   <tr>
                    <th>Почта</th>
                    <th>Роль</th>
                    <th>Имя</th>
                    <th></th>
                   </tr>
                  </thead>
                  <tbody>
                   {users.map((option) => (
                    <tr className="manager_el" key={option._id} >
                     <td>{option.email}</td>
                     <td>{option.role}</td>
                     <td>{option.nameManager}</td>
                     <td className="manager_el_btns">
                      <div              className="register              edit_man_btn"              onClick={()              =>
handleManagerClick(option._id)}>Редагувати</div>
                      <div className="register list_task_btn" onClick={() => handleViewTask(option._id)}>Список
задач</div>
                     </td>
                    </tr>

                   ))}
                  </tbody>
                 </table>
                </div>


                <EditManagerForm close={close} setSelectedManager={setSelectedManager} editMode={editMode}
selectedManager={selectedManager}                                                handleEditManager={handleEditManager}
```

```
registrationDataTask={registrationDataTask}  handleRegistrationTaskChange={handleRegistrationTaskChange}
handleRegistrationTask={handleRegistrationTask} />

                <table style={{ marginTop: "40px" }}>
                 <tbody>

                  {editTaskMode && <>
                   <div className={`overlay ${editTaskMode ? "active" : ""}`} onClick={close}></div>

                   <div style={{ display: "block" }} className={`user_modal ${editTaskMode ? "show" : ""}`}>
                    <div className="close" onClick={close}>
                     &times;
                    </div>

                    <h3 className="list_task_head">Список Задач</h3>
                    <h4 className="list_task_head">Менеджер: {selectedManager.nameManager}</h4>
                    <div className="period">Період:</div>
                    <div style={{ marginBottom: "50px" }} className="period_inputs period_inputs_date">
                     <input  type="date"  value={startDateFilter}  onChange={(e)  =>
handleStartDateChange(e.target.value)} />
                     <input  type="date"  value={endDateFilter}  onChange={(e)  =>
handleEndDateChange(e.target.value)} />
                    </div>
                    <table>
                     {sortByDate(selectTask)
                      .filter(el => {
                       if (startDateFilter && endDateFilter) {
                        const taskDate = new Date(el.startDate);
                        return taskDate >= new Date(startDateFilter) && taskDate <= new Date(endDateFilter);
                       }
                       return true;
                      })
                      .map(el => (
                       <tr key={el.id}>
                        <td>{el.taskLine}</td>
                        <td>{el.startDate}</td>
                        <td>{el.endDate}</td>
                        <td>{el.taskStatus === 'true' ? 'Выполнено' : 'Не виконано'}</td>
                       </tr>
                      ))}
                    </table>

                   </div>
                  </>}

                 </tbody>
                </table>
               </div>
               }
               {showProductBlock &&
                <ProductBlock
                 registrationDataProduct={registrationDataProduct}
                 handleRegistrationProductChange={handleRegistrationProductChange}
                 handleRegistrationProduct={handleRegistrationProduct}
                 dataProducts={dataProducts}
                 handleProductClick={handleProductClick}
                 handleDeleteProduct={handleDeleteProduct}
                 editProductMode={editProductMode}
                 setSelectedProduct={setSelectedProduct}
                 selectedProduct={selectedProduct}
                 setEditMode={setEditMode}
                 fetchData={fetchData}
                 close={close}
```

```
            />
          }

        </div>
      </>
    ) : (
      <Spinner />
    )}
  </div>
 );
}

export default AdminPage
```

**ДОДАТОК Б**

**ВІДГУК**
**керівника економічного розділу**
**на кваліфікаційну роботу бакалавра**
**на тему:**
**"Розробка CRM-системи для керування клієнтської**
**та продуктової бази компанії з використанням React та Node.js"**

**студента групи 121-20-1 Пучка Владислава Сергійовича**

**Керівник економічного розділу**
**доцент каф. ПЕП та ПУ, к.е.н** **Л. В. Касьяненко**

**ДОДАТОК В**

**Перелік файлів на диску**

## ПЕРЕЛІК ДОКУМЕНТІВ НА МАГНІТНОМУ НОСІЇ

| **Ім’я файлу** | **Опис** |
|---|---|
| Пояснювальні документи | |
| Кваліфікаційна робота.docx | Пояснювальна записка до кваліфікаційної роботи. Документ Word. |
| Кваліфікаційна робота.pdf | Пояснювальна записка до кваліфікаційної роботи в форматі PDF |
| Програма | |
| програма.rar | Архів. Містить коди програми і скомпільовану програму |
| Презентація | |
| презентація.ppt | Презентація кваліфікаційної роботи |