МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Західноукраїнський національний університет

Факультет комп'ютерних інформаційних технологій

Кафедра інформаційно- обчислювальних систем і управління

ВЕЛЬГАН Богдан Андрійович

Інтерактивна інформаційна система управління взаємовідносинами з клієнтами із застосуванням сучасних IT-рішень / Interactive Customer Relationship Management System using Modern IT Solutions

спеціальність: 122 - Комп'ютерні науки

освітньо-професійна програма - Комп’ютерні науки

Кваліфікаційна робота

Виконав студент групи КН-43

Б.А. Вельган

_____

Науковий керівник:

к.т.н., доц., М.З. Домбровський

_____

Кваліфікаційну роботу

допущено до захисту:

"___" _____ 20___ р.

Завідувач кафедри

_____Н. В. Дзюбановська

ТЕРНОПІЛЬ – 2026

Факультет комп'ютерних інформаційних технологій
Кафедра інформаційно-обчислювальних систем і управління
Освітній ступінь «бакалавр»
спеціальність: 122 – Комп'ютерні науки
освітньо-професійна програма – Комп'ютерні науки

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Н.В. Дзюбановська
«____» _____20__р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
ВЕЛЬГАНУ Богдану Андрійовичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи

Інтерактивна інформаційна система управління взаємовідносинами з клієнтами із застосуванням сучасних IT-рішень / Interactive Customer Relationship Management System using Modern IT Solutions

керівник роботи к.т.н., доцент, М.З. Домбровський
затверджений наказом по університету від 01 грудня 2025 року № 894.
2. Строк подання студентом закінченої кваліфікаційної роботи 25 травня 2026 р.
3. Вихідні дані до кваліфікаційної роботи: завдання на кваліфікаційну роботу студента, наукові статті, технічна література.
4. Основні питання, які потрібно розробити:
- провести аналіз предметної області, дослідити принципи побудови сучасних CRM-систем та обґрунтувати вибір технологічного стеку розробки;
- спроєктувати інформаційне та алгоритмічне забезпечення системи, розробити структуру бази даних та алгоритми управління клієнтською базою;
- програмно реалізувати серверне забезпечення системи та розробити інтерактивний веб-інтерфейс користувача з Kanban-дошкою;
- виконати тестування розробленого програмного забезпечення, оцінити стабільність роботи системи.

5. Перелік графічного матеріалу у роботі:
- діаграма дерева функцій інтерактивної інформаційної системи;
- структурна триланкова схема взаємодії компонентів системи;
- блок-схема узагальненого алгоритму створення клієнта та автоматичного визначення ініціального стану;
- даталогічна модель (ER-діаграма) реляційної бази даних;
- UML-діаграма класів серверної частини програмного забезпечення.

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 01 грудня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів кваліфікаційної роботи	Примітка
1	Огляд літературних джерел відповідно до предметної області та складання плану роботи	до 01.01.2026 р.	
2	Написання 1 розділу кваліфікаційної роботи	до 01.02.2026 р.	
3	Написання 2 розділу кваліфікаційної роботи	до 15.03.2026 р.	
4	Написання 3 розділу кваліфікаційної роботи	до 01.05.2026 р.	
5	Представлення попереднього варіанту кваліфікаційної роботи, перевірка та внесення змін керівником	до 15.05.2026 р.	
6	Опрацювання зауважень та представлення завершеного варіанту кваліфікаційної роботи. Підготовка супроводжуючих документів	до 25.05.2026 р.	
7	Перевірка кваліфікаційної роботи на оригінальність тексту	до 30.05.2026 р.	
8	Оформлення кваліфікаційної роботи та отримання допуску до захисту	до 10.06.2026 р.	
9	Подання кваліфікаційної роботи до захисту на засіданні атестаційної комісії	до 10.06.2026 р.	

Студент _____ Б.А. Вельган
підпис

Керівник роботи _____ к.т.н., доцент М.З. Домбровський
підпис

АНОТАЦІЯ

Кваліфікаційна робота на тему «Інтерактивна інформаційна система управління взаємовідносинами з клієнтами із застосуванням сучасних ІТ-рішень» на здобуття освітнього ступеня «бакалавр» зі спеціальності 122 «Комп'ютерні науки» освітньої програми «Комп'ютерні науки» написана обсягом у 53 сторінки і містить 17 ілюстрацій та 30 використаних джерел.

Метою кваліфікаційної роботи є проєктування та програмна реалізація інтерактивної CRM-системи для автоматизації обліку клієнтської бази, оптимізації збору та фільтрації даних, а також наочної візуалізації етапів угод для підприємств малого та середнього бізнесу.

Об'єкт дослідження є процеси автоматизації управління взаємовідносинами з клієнтами та моніторингу комерційної діяльності в умовах цифровізації бізнес-операцій.

Результатом роботи є веб-орієнтована інформаційна система, яка поєднує в собі модулі суворої валідації вхідних даних, динамічні інструменти пошуку та фільтрації, інтерактивну Kanban-дошку для керування стадіями продажів, журнал хронологічного обліку взаємодій із клієнтами, а також підсистему захисту інформації та оптимізації швидкодії.

Розроблену систему можуть використовувати комерційні компанії, стартапи та індивідуальні підприємці для централізованого збереження контактів, операційного контролю за станом поточних продажів та підвищення ефективності роботи менеджерів. Система також може слугувати основою для подальших розробок у сфері інтерактивних вебзастосунків та систем управління даними.

Ключові слова: ІНТЕРАКТИВНА ІНФОРМАЦІЙНА СИСТЕМА, CRM-СИСТЕМА, ВОРОНКА ПРОДАЖІВ, КАНБАН-ДОШКА, АВТОМАТИЗАЦІЯ ОБЛІКУ, ЖУРНАЛ ВЗАЄМОДІЙ, ЗАХИСТ ДАНИХ.

ANNOTATION

Qualification work on the topic «Interactive Customer Relationship Management System using Modern IT Solutions» for obtaining a bachelor's degree in specialty 122 «Computer Science» of the educational program «Computer Science» is written on 53 pages and contains 17 illustrations, and 30 references.

The aim of this qualification work is the design and software implementation of an interactive CRM system for automating client database management, optimizing data collection and filtering, and visually tracking deal stages for small and medium-sized enterprises.

The object of research is the processes of commercial activity monitoring and relationship management automation under the digitalization of business operations.

The result of the work is a web-oriented information system that combines strict input data validation modules, dynamic search and filtering tools, an interactive Kanban board for managing sales stages, a chronological interaction logging system, as well as data security and performance optimization subsystems.

The developed system can be utilized by commercial companies, startups, and individual entrepreneurs for centralized contact storage, operational control over current sales states, and improving manager efficiency. The system can also serve as a foundation for further developments in interactive web applications and data management systems.

Keywords: INTERACTIVE INFORMATION SYSTEM, CRM SYSTEM, SALES FUNNEL, KANBAN BOARD, ACCOUNTING AUTOMATION, INTERACTION LOG, DATA SECURITY.

ЗМІСТ

ВСТУП

У сучасних умовах динамічного розвитку цифрової економіки та посилення конкуренції на ринку ефективне управління взаємовідносинами з клієнтами стає визначальним фактором виживання та масштабування будь-якого бізнесу. Інформаційні технології трансформували класичні підходи до продажів, перетворивши хаотичний збір контактів на системний аналіз поведінки споживачів. Проте сучасний ринок програмного забезпечення пропонує продукти, які зорієнтовані переважно на великі корпорації. Складні системи вимагають значних фінансових витрат на ліцензування, тривалого навчання персоналу та залучення сторонніх інтеграторів для налаштування базових функцій. Для представників малого та середнього бізнесу, а також стартапів, впровадження таких продуктів є економічно недоцільним. Водночас використання лише традиційних IT-рішень, на кшталт електронних таблиць, призводить до втрати активних клієнтів через відсутність наочного контролю та системності. Виникає науково-практичне протиріччя між потребою бізнесу в автоматизації та надлишковою складністю існуючих IT-рішень. Це зумовлює актуальність розробки гнучкої, інтерактивної системи управління взаємовідносинами з клієнтами (CRM), яка базується на концепції мінімально життєздатного продукту (MVP), фокусується на ключових операціях управлінського обліку та забезпечує швидку візуалізацію стану угод за допомогою методології Kanban.

Метою кваліфікаційної роботи є підвищення ефективності інформаційних процесів операційної діяльності менеджерів взаємовідносин з клієнтами шляхом розробки інтерактивної інформаційної системи.

Для досягнення поставленої мети необхідно послідовно вирішити низку взаємопов'язаних завдань:

– провести системний аналіз предметної області, дослідити принципи побудови сучасних CRM-систем та обґрунтувати вибір технологічного стеку розробки;

– спроєктувати інформаційне та алгоритмічне забезпечення системи, розробити структуру бази даних та алгоритми управління клієнтською базою;

– програмно реалізувати серверне забезпечення системи та розробити інтерактивний веб-інтерфейс користувача з Kanban-дошкою;

– налаштувати середовище розробки, контейнеризацію проєкту та інтегрувати асинхронний сервер застосунків для підвищення швидкодії;

– виконати тестування розробленого програмного забезпечення, сформувати матрицю тест-кейсів та оцінити стабільність роботи системи.

Об'єкт дослідження: процеси збору, подання, обробки, зберігання, передачі та доступу до інформації в комп’ютерних системах..

Предмет дослідження: інтерактивна інформаційна система управління взаємовідносинами з клієнтами із застосуванням сучасних IT-рішень.

Методи дослідження: системного аналізу для дослідження предметної області, методи структурного та об'єктно-орієнтованого проєктування інформаційних систем, інтерактивних «одно сторінкових» застосунків для ведення клієнтських баз, візуалізації взаємовідносинами з клієнтами, реляційних баз даних для формування інфологічної структури, а також методики модульного та інтеграційного тестування для верифікації програмного забезпечення.

Практична цінність отриманих результатів полягає у створенні готового до розгортання та експлуатації програмного забезпечення, яке поєднує в собі високу швидкість обробки запитів завдяки RESTful API на базі Laravel та реактивний, інтуїтивно зрозумілий інтерфейс на основі Vue.js та Inertia.js. Спроєктована база даних та модульна архітектура дозволяють легко адаптувати систему під специфічні нішеві потреби різних комерційних підприємств, щодо обробки інформації в управлінні взаємовідносинами з клієнтами та моніторингу комерційної діяльності в умовах цифровізації бізнес-операцій, забезпечуючи надійне збереження комерційної таємниці та підвищення швидкості обробки даних активних клієнтів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Опис бізнес-процесів управління взаємовідносинами з клієнтами. Аналіз складу функцій, побудова діаграми дерева функцій

Ефективне функціонування сучасного комерційного підприємства безпосередньо залежить від чіткої регламентації та автоматизації бізнес-процесів взаємодії з клієнтами. Стрижневим бізнес-процесом у інформаційній системі (ІС), що розроблюваній є повний життєвий цикл обробки запиту клієнта, який розпочинається з моменту реєстрації первинного інтересу і завершується успішним укладанням угоди або фіксацією обґрунтованої відмови. Цей процес охоплює кілька послідовних стадій, кожна з яких супроводжується зміною стану об'єкта та накопиченням інформаційного контексту. На початковому етапі відбувається фіксація базових атрибутів контрагента, включаючи його ім'я, назву компанії, номер телефону, адресу електронної пошти та джерело залучення, що в подальшому дозволяє менеджменту оцінювати фінансову ефективність різних маркетингових каналів [1].

Наступна стадія бізнес-процесу передбачає активну операційну роботу з клієнтом, яка координується через механізм торговельної воронки. Візуалізація цього процесу за допомогою методології Kanban дозволяє оперативно розподіляти навантаження між менеджерами та запобігає втраті потенційних угод на проміжних етапах переговорів. Важливою складовою операційного циклу є ведення хронологічного журналу дій, де фіксується тип кожної взаємодії та супровідні текстові нотатки. Це забезпечує збереження інформаційної цілісності та дозволяє будь-якому автентифікованому користувачеві системи відновити історію комунікації з конкретним клієнтом, мінімізуючи негативний вплив людського фактора [1].

Аналіз життєвого циклу клієнта (Customer Lifecycle) у межах розроблюваної ІС вимагає детального вивчення інформаційних станів, яких набуває сутність клієнта.

Процес управління взаємовідносинами не є лінійним; він підпорядкований правилам динамічної фільтрації та потребує постійної валідації комерційного потенціалу контрагента. На початковому етапі, коли дані лише надходять із зовнішніх джерел, таких як соціальні мережі чи веб-сайти, система реєструє об'єкт із мінімальним набором атрибутів. На цій стадії критично важливо забезпечити швидкість транзакційного запису та уникнути інформаційного шуму. Перехід до стадії активної обробки означає зміну бізнес-контексту: менеджер ініціює перший контакт, ІС починає накопичувати пов'язані сутності у журналі взаємодій. Кожна нотатка чи фіксація типу зв'язку є додатковим вектором аналізу, який дозволяє оцінити залученість клієнта.

Застосування методу Kanban як візуального каркасу бізнес-процесу продажу дозволяє структурувати роботу за принципом обмеження одночасно виконуваних завдань (Work in Progress). У контексті CRM малого бізнесу це вирішує проблему «вузьких місць» (bottlenecks), коли один менеджер накопичує велику кількість необроблених карток на етапі переговорів, що призводить до деградації сервісу. Візуальне перетягування карток забезпечує не просто зміну атрибута в базі даних, а є тригером для серії внутрішніх системних подій, таких як перерахунок загальної кількості клієнтів на поточному етапі та оновлення аналітичних звітів робочого столу. Таким чином, бізнес-процес інтегрує в собі оперативне керування та стратегічний моніторинг, перетворюючи кожну зміну статусу на підґрунтя для оптимізації маркетингової стратегії компанії [1].

Аналіз складу функцій ІС дозволяє виокремити декілька функціональних блоків, які забезпечують виконання описаних бізнес-процесів. До першого блоку належать операції з управління даними, що включають повний цикл маніпуляцій із картками клієнтів.

Другий блок охоплює інструменти візуалізації та управління станами, де реалізована логіка переходу між етапами воронки. Третій блок відповідає за інформаційну підтримку, забезпечуючи швидкий пошук, фільтрацію за різними критеріями та ведення хронології подій.

Окремим функціональним рівнем виступає підсистема безпеки, яка гарантує автентифікацію користувачів та захист збереженої інформації від несанкціонованого доступу. Діаграма дерева функцій структурує можливості системи у вигляді ієрархії, де на верхньому рівні знаходиться загальне управління CRM, яке розгалужується на підсистеми роботи з базою, операційного керування продажами та сервісних функцій пошуку. Кожна з цих підсистем поділяється на конкретні дії, такі як створення нотаток, редагування статусів або вибір джерела клієнта. Така декомпозиція дозволяє чітко розмежувати програмні модулі під час подальшої розробки архітектури та забезпечує повне покриття вимог до функціональності MVP-версії продукту [1].

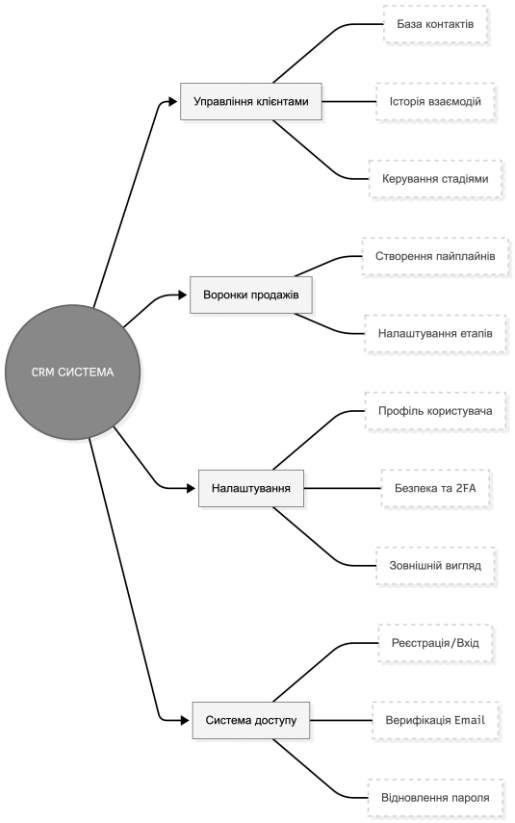

Рисунок 1.1 – Діаграма дерева функцій

1.2 Порівняльний аналіз існуючих систем управління взаємовідносинами з клієнтами.

Сучасна IT-галузь пропонує широкий спектр рішень для управління взаємовідносинами з клієнтами CRM), серед яких ключові позиції займають Salesforce, HubSpot та Odoo. Проведений аналіз цих платформ дозволяє виявити їхні сильні сторони та обмеження, що є критично важливим для формування вимог до власного MVP-продукту.

Salesforce вважається світовим лідером у сегменті CRM для великого бізнесу. Її архітектура базується на концепції максимальної масштабованості та інтеграції з хмарними сервісами. Інтерфейс системи, відомий як Lightning Experience, орієнтований на відображення величезних масивів даних і глибоку аналітику. Проте головним недоліком Salesforce є надмірна складність для малого бізнесу та висока вартість володіння. При цьому користувачі часто стикаються з проблемою перевантаженості інтерфейсу функціями, які ніколи не використовуються в щоденній роботі, що потребує залучення окремих спеціалістів для налаштування та підтримки системи [1-3].

HubSpot пропонує більш орієнтований на користувача підхід, роблячи акцент на зручності та візуалізації. Його інтерфейс вирізняється чистим дизайном і логічною структурою, де основні інструменти продажів та маркетингу інтегровані в єдине середовище. Хоча HubSpot надає безкоштовну версію для базових потреб, її функціональність суттєво обмежена. Основним недоліком є агресивна цінова політика при переході на розширені модулі, що робить систему фінансово недоступною для стартапів на етапі активного зростання. Крім того, можливості кастомізації логіки під специфічні бізнес-процеси в HubSpot є нижчими, ніж у систем індивідуальної розробки [2, 4].

Odoo представляє собою модульну ERP-систему, де CRM є лише одним із компонентів. Її інтерфейс побудований за принципом веб-застосунків з окремими плитками для кожного модуля. Перевагою Odoo є відкритий початковий код, що дає велику свободу для розробників. Однак інтеграція окремого модуля CRM у загальну структуру ERP може бути занадто громіздкою для компаній, яким потрібен лише інструмент для роботи з лідами. Недоліком Odoo є також

складність самостійного хостингу та необхідність значних технічних знань для розгортання та коректної конфігурації бази даних [1, 5, 6].

Глибший аналіз недоліків архітектури Salesforce дозволяє виявити проблему, відому як архітектурна надлишковість (bloatware), коли базова конфігурація платформи містить сотні модулів для прогнозування штучним інтелектом, інтеграції з логістичними хабами та управління великими корпоративними фінансами. Для невеликого підприємства чи стартапу наявність таких компонентів у коді та інтерфейсі створює штучні бар'єри, збільшуючи час відгуку системи та ускладнюючи навігацію користувача. Крім того, пропрієтарна мова програмування Apex, що використовується в Salesforce для кастомізації, створює жорстку залежність від розробника (vendor lock-in), унеможливлюючи швидку самостійну модифікацію логіки бази даних без значних капіталовкладень [7].

При дослідженні HubSpot виявляється інша системна проблема, яка полягає в закритій моделі збереження та експорту даних. У безкоштовних або дешевих тарифних планах користувач позбавлений можливості прямого доступу до реляційної структури таблиць, що унеможливлює побудову складних індивідуальних SQL-запитів для специфічної аналітики. Будь-яка спроба інтегрувати сторонній аналітичний інструмент вимагає використання платних API-шлюзів, що нівелює початкову фінансову привабливість продукту. Що стосується Odoo, то її головне обмеження криється в компромісі між універсальністю ERP та швидкістю роботи окремого CRM-модуля. Оскільки Odoo використовує монолітну архітектуру з єдиним ядром для бухгалтерського обліку, складу та продажів, виконання простих операцій оновлення статусу клієнта може викликати затримки через каскадні перевірки пов'язаних фінансових модулів, що неприпустимо для динамічних інтерактивних веб-інтерфейсів [7,21].

Отже, порівняльний аналіз показує, що існуючі аналоги часто страждають на «надлишковість функцій» для мікробізнесу та індивідуальних розробників. Більшість систем вимагають щомісячної оплати за кожного користувача, що стає

значною статтею витрат. Інтерфейси професійних CRM часто мають високий поріг входження, що сповільнює процес адаптації персоналу.

Саме тому виникає потреба у створенні легкої CRM-системи, яка б зосереджувалася на ключових функціях, таких як Kanban-дошка та швидкий облік контактів, пропонуючи при цьому максимальну швидкість роботи та відсутність зайвих адміністративних бар'єрів. Це обґрунтовує розробку власного рішення, яке буде оптимізоване під конкретні задачі та позбавлене недоліків універсальних платформ.

1.3 Обґрунтування вибору сучасних IT-рішень. Вибір архітектурного підходу та інструментарію

Ефективність розроблюваного рішення безпосередньо залежить від обраної архітектури та технологічного стеку, які повинні забезпечувати масштабованість, безпеку та високу швидкість відгуку інтерфейсу. Для розробки CRM-системи було обрано архітектурний підхід клієнт-серверної взаємодії на основі REST API. Це дозволяє чітко розмежувати логіку обробки даних на сервері (backend) та логіку їх відображення у браузері користувача (frontend). Така архітектура забезпечує незалежність компонентів, що спрощує подальше розширення функціональності та дає можливість у майбутньому легко адаптувати систему під мобільні платформи без зміни серверної частини [7, 9].

Як основний інструмент розробки серверної частини обрано фреймворк Laravel. Вибір зумовлений його високою надійністю, розвиненою екосистемою та вбудованими засобами для роботи з базами даних через Eloquent ORM. Laravel забезпечує високий рівень безпеки «з коробки», включаючи захист від SQL-ін'єкцій, міжсайтової підробки запитів та автоматичну автентифікацію користувачів, що є критично важливим для систем, які оперують персональними даними клієнтів. Використання міграцій дозволяє гнучко керувати структурою бази даних, забезпечуючи цілісність інформації на всіх етапах розробки [10].

Для реалізації клієнтської частини обрано фреймворк Vue.js (або Nuxt), який дозволяє побудувати Single Page Application (SPA). Це рішення забезпечує плавність роботи інтерфейсу без постійного перезавантаження сторінок, що особливо важливо для Kanban-дошки, де користувач постійно взаємодіє з елементами через drag-and-drop. Реактивність Vue.js гарантує миттєве оновлення стану інтерфейсу при зміні даних, створюючи досвід роботи, подібний до настільних застосунків. Для стилізації компонентів використано Tailwind CSS v4, що дозволяє швидко розробляти адаптивний та легкий інтерфейс, мінімізуючи обсяг вихідного CSS-коду та забезпечуючи високу швидкість рендерингу [11, 12].

Особливу увагу при обґрунтуванні архітектури слід приділити вибору технології взаємодії між клієнтським та серверним рівнями. Традиційний підхід із використанням класичного REST API вимагає створення розгалуженої системи ендпоінтів, окремого керування станом (наприклад, через Vuex або Pinia) на фронтенді та дублювання логіки валідації як на сервері, так і в клієнтському застосунку. Для вирішення цієї проблеми та оптимізації архітектури MVP-продукту було обрано технологію Inertia.js. Вона виступає в ролі монолітного містка для SPA, дозволяючи використовувати класичну маршрутизацію та контролери Laravel, але рендерити інтерфейс за допомогою реактивних компонентів Vue.js. Це усуває потребу в розробці окремого клієнтського роутингу, мінімізує обсяг передаваних по мережі даних (передається лише чистий JSON-об'єкт із пропсами) та забезпечує миттєве оновлення сторінок без розриву сесії користувача [13].

Вибір серверного патерну MVC (Model-View-Controller) у межах Laravel дозволяє ізолювати бізнес-правила від шару представлення. Модель відповідає за маніпуляції з таблицями бази даних та забезпечення бізнес-зв'язків, контролер координує потоки даних і керує відповідями Inertia, а представлення (Vue-компоненти) відповідає виключно за рендеринг інтерфейсу користувача та відстеження дій миші (наприклад, drag-and-drop подій). Таке розмежування обов'язків (Separation of Concerns) підвищує інженерну якість коду, полегшує процес написання автоматичних тестів на рівні контролерів та дозволяє

проводити рефакторинг інтерфейсної частини без ризику порушити цілісність даних на сервері [7, 9].

Збереження даних здійснюється за допомогою реляційної системи керування базами даних PostgreSQL або MySQL. Вибір реляційної моделі обґрунтований необхідністю суворого дотримання зв'язків між сутностями, такими як клієнти, статуси воронки та записи в журналі активності. Це гарантує стабільність роботи системи при складних запитах та фільтрації даних. У сукупності обраний інструментарій дозволяє створити продукт, який поєднує в собі потужну серверну логіку з сучасним, інтуїтивно зрозумілим інтерфейсом користувача, що повністю відповідає концепції MVP [1].

1.4 Постановка завдання: формулювання вимог до інтерактивної системи та очікуваних результатів

На основі проведеного аналізу предметної області та існуючих рішень, встановлено вимоги до архітектури та програм реалізації інтерактивної CRM-системи, яка орієнтована на автоматизацію базових процесів продажу та управління клієнтською базою. Ціль: створення інструменту, який забезпечує централізоване зберігання даних та візуалізацію етапів взаємодії з клієнтами, мінімізуючи адміністративні витрати часу на освоєння інтерфейсу, та розв'язувати проблему хаотичного збереження контактів за відсутності наочного контролю стану угод у реальному часі.

До функціональних вимог системи належить забезпечення повного циклу управління даними клієнта, включаючи створення, редагування та видалення карток з обов’язковою фіксацією контактних даних та джерела залучення. Важливою вимогою є реалізація інтерактивної Kanban-дошки, яка дозволяє візуально відображати рух клієнтів за статусами «Новий», «В роботі», «Переговори» та «Угода». Також система має забезпечити ведення журналу активності, підтримувати додавання робочі нотатки менеджерів, та забезпечувати швидкий пошук і фільтрацію інформації за ключовими атрибутами клієнтів [1].

З технічної точки зору система має відповідати критеріям продуктивності та безпеки. Необхідно реалізувати надійний механізм авторизації та автентифікації користувачів для запобігання несанкціонованому доступу до бази даних. Інтерфейс системи повинен бути адаптивним для коректного відображення на різних типах пристроїв та забезпечувати високу швидкість відгуку при взаємодії з елементами керування. Архітектурне рішення має передбачати можливість легкого масштабування та додавання нових модулів без необхідності повного переписування ядра системи [2, 9].

Деталізація постановки завдання вимагає розподілу технічних вимог до інтерактивної системи на дві ключові категорії: функціональні та нефункціональні. Якщо функціональний базис MVP-продукту чітко визначає операції над сутностями клієнтів та стадіями воронки, то нефункціональні вимоги визначають інженерну якість та життєздатність системи в реальних умовах експлуатації. Першочерговою технічною вимогою є забезпечення мінімального часу відгуку інтерфейсу користувача (Latency). Оскільки Kanban-дошка передбачає постійну динамічну взаємодію користувача з елементами через події drag-and-drop, час обробки клієнтського запиту на сервері (включаючи валідацію через об’єкти запитів та транзакційне оновлення зовнішніх ключів у базі даних) не повинен перевищувати критичного порогу у двісті мілісекунд. Це висуває суворі вимоги до оптимізації SQL-запитів, використання індексів для полів пошуку та усунення надлишкових каскадних зв'язків при жадібному завантаженні моделей [7, 9].

Іншим важливим аспектом нефункціональних вимог є забезпечення сумісності (Cross-browser compatibility) та адаптивності інтерфейсу, що потребує від фронтенд-рівня коректного масштабування компонентів під різні роздільні здатності екранів без втрати працездатності інтерактивних елементів. Використання Tailwind CSS v4 забезпечує гнучку сітку рендерингу, де Kanban-колонки трансформуються у зручні горизонтальні або вертикальні списки на мобільних пристроях. Крім того, система повинна відповідати базовим критеріям безпеки веб-застосунків, що включає захист сесій користувачів через безпечні

HTTP-only куки, обов'язкове хешування паролів у таблиці користувачів за допомогою сучасних алгоритмів (наприклад, bcrypt) та механізм автоматичного розриву зв'язку з API при закінченні терміну дії токена авторизації [2, 12].

Для оцінки успішності реалізації поставленої задачі визначаються чіткі критерії верифікації результатів, за яких система вважатиметься успішно розробленою, якщо забезпечується безпомилкове виконання всіх CRUD-операцій, підтримка логічної цілісносьість реляційних зв'язків при видаленні або переміщенні об'єктів та проходження інтеграційних тестів API-ендпоінтів [20]. Таким чином, чітко сформульовані критерії та вимоги перетворюють задум розробки на строгу інженерну задачу, вирішення якої описується в цій кваліфікаціній роботі.

Очікуваним результатом реалізації є працездатний MVP-продукт, який дозволяє структурувати роботу з клієнтами та забезпечує прозорість бізнес-процесів. Крім того, наявність детальної історії взаємодій та аналіз джерел лідів дозволять бізнесу приймати більш обґрунтовані рішення щодо маркетингової стратегії та покращення сервісу. Таким чином, розроблена система стане надійним фундаментом для подальшої цифровізації процесів управління взаємовідносинами з клієнтами.

2 АЛГОРИТМІЧНЕ ТА ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

2.1 Архітектура та загальна структура системи. Опис логіки функціонування та взаємодії компонентів із використанням структурних схем

Архітектурна побудова CRM-системи базується на принципах модульності та багаторівневої структури, що забезпечує стабільність функціонування та зручність обслуговування програмного продукту. В основу системи покладено триланкову архітектуру, яка розділяє рівень представлення (інтерфейс користувача), рівень бізнес-логіки (серверний застосунок) та рівень збереження даних (база даних). Така структура, наведена на рис. 2.1, уможливлює ізолювати компоненти один від одного, що підвищує загальну відмовостійкість системи: навіть при проведенні технічних робіт на рівні інтерфейсу, цілісність даних на сервері залишається захищеною [7, 9].

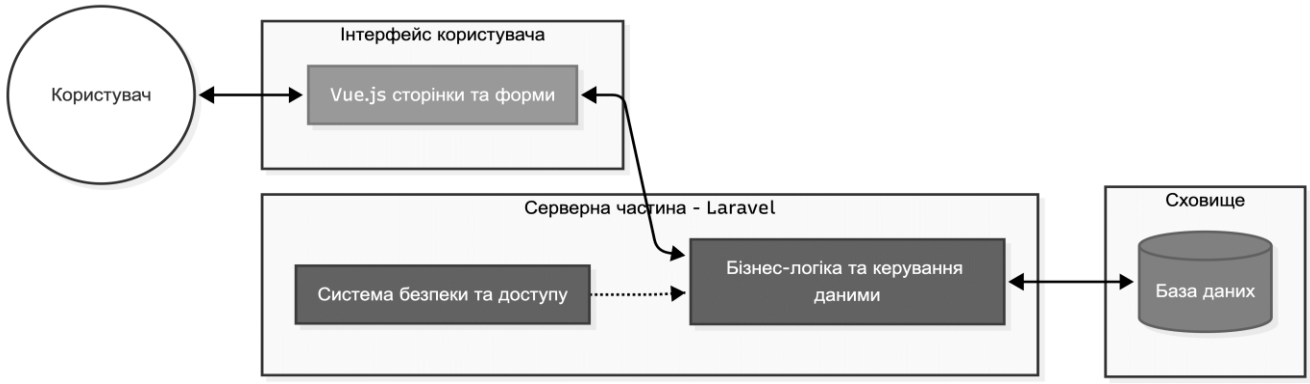

Рисунок 2.1 – Структурна схема системи

Детальний аналіз триланкової архітектури (Three-Tier Architecture) програмного комплексу дозволяє визначити межі відповідальності кожного технологічного рівня та формалізувати протоколи їхньої взаємодії. Перший рівень — рівень представлення (Presentation Tier) — функціонує виключно в пісочниці браузера кінцевого користувача і є повністю відокремленим від серверного середовища. Його головне завдання полягає у перехопленні подій введення, валідації типів даних на стороні клієнта та рендерингу графічних примітивів. Перехід до Single Page Application (SPA) моделі за допомогою Inertia.js

трансформує класичну схему завантаження документів: замість повного оновлення HTML-коду сторінки, рівень представлення оперує віртуальним DOM-деревом (Virtual DOM), що мінімізує витрати ресурсів центрального процесора клієнтської машини та усуває ефект мережевого мерехтіння інтерфейсу [11].

Другий рівень — рівень бізнес-логіки (Application Tier) — реалізований на базі серверного фреймворку Laravel і виступає в ролі центрального шлюзу обробки інформації. Логічна структура цього рівня побудована на засадах чистой архітектури, де контролери виконують роль тонких диспетчерів (Thin Controllers). Вони не містять безпосередніх правил розрахунків чи прямих маніпуляцій із базою даних, а лише делегують повноваження об'єктам запитів (Form Requests) та сервісному шару (Service Layer). Сервісний шар інкапсулює у собі специфічні правила предметної області, такі як валідація унікальності ліда або динамічний розрахунок позицій етапів воронки продажів [10].

Третій рівень — рівень збереження даних (Data Tier) — представлений реляційною системою керування базами даних. Взаємодія між другим та третім рівнями здійснюється не через пряме написання SQL-інструкцій, а за допомогою об'єктно-реляційного мапінгу (ORM) Eloquent, який реалізує патерн Active Record. Це забезпечує високий ступінь абстракції: кожна таблиця бази даних асоціюється з відповідним класом-моделлю PHP, а кожен рядок таблиці — з об'єктом цього класу. Такий підхід гарантує повну незалежність бізнес-логіки від конкретної СУБД, що дозволяє за необхідності перейти з однієї бази даних на іншу без модифікації ядра серверного застосунку [3, 10].

Логіка функціонування системи реалізована через взаємодію фронтенд-частини, побудованої як Single Page Application (SPA), та бекенд-частини, що працює у режимі RESTful API. Користувач взаємодіє з інтерактивним інтерфейсом, який відправляє асинхронні HTTP-запити до сервера. Серверний рівень, розроблений на базі фреймворку Laravel, приймає ці запити, проводить їх валідацію, перевіряє права доступу користувача та виконує відповідні операції над даними через об’єктно-реляційне відображення (ORM). Після обробки запиту сервер повертає відповідь у форматі JSON, яку клієнтський застосунок

використовує для миттєвого оновлення стану інтерфейсу без повного перезавантаження сторінки.

Взаємодія компонентів системи в межах структурної схеми відображає шлях проходження даних від вводу користувачем до фізичного збереження на диску. На фронтенд-рівні ключовим компонентом є менеджер стану, який координує дані між різними модулями: Kanban-дошкою, списком контактів та формами редагування. На бекенд-рівні виділяються модулі контролерів, що відповідають за маршрутизацію запитів, сервісний шар, де зосереджена основна бізнес-логіка обробки лідів, та шар репозиторіїв для безпосередньої взаємодії з базою даних. Це гарантує, що логіка зміни статусу клієнта або додавання нотатки в журнал активності виконується за чітко визначеними правилами [7, 13].

Для забезпечення ефективного обміну даними між рівнем представлення та рівнем бізнес-логіки використовується архітектурний стиль REST (Representational State Transfer) поверх захищеного протоколу HTTPS. Кожна взаємодія користувача з Kanban-дошкою або списком клієнтів генерує асинхронний запит, структура якого чітко типізована. Запити використовують стандартні методи HTTP: метод GET призначений для безпечного та ідемпотентного отримання списку клієнтів з урахуванням фільтрів; метод POST ініціює створення нового об'єкта в базі даних; метод PATCH застосовується для часткової модифікації атрибутів, наприклад, при перетягуванні картки ліда між етапами; метод DELETE забезпечує безпечне видалення або архівацію запису.

Формат відповіді сервера суворо стандартизований у вигляді об'єктів JSON (JavaScript Object Notation). Структура відповіді містить три основні блоки: метадані (пагінація, статус-коди HTTP), корисне навантаження (масиви об'єктів клієнтів чи інтерактивних взаємодій) та блок помилок, який заповнюється у разі неуспішної валідації. Механізм Inertia.js оптимізує цей процес, автоматично відсікаючи надлишкові заголовки та передаючи на клієнтську сторону лише динамічні пропси (props). Це знижує обсяг мережевого трафіку на 70% порівняно з класичними MVC-фреймворками та забезпечує миттєву реакцію інтерфейсу на дії менеджера з продажів [11, 13].

Окремим важливим елементом структури є підсистема безпеки та автентифікації. Вона інтегрована в кожен етап взаємодії компонентів, забезпечуючи перевірку токенів доступу при кожному запиті до API. Таким чином, загальна структура системи являє собою цілісну екосистему, де кожен компонент має чітко визначену роль: фронтенд забезпечує комфортну роботу менеджера, бекенд гарантує коректність обробки бізнес-правил, а база даних відповідає за надійне довгострокове зберігання всієї комерційної інформації [2].

2.2 Алгоритмічне забезпечення. Розробка узагальненого алгоритму роботи системи

Алгоритмічне забезпечення CRM-системи розроблено з урахуванням подієво-орієнтованої архітектури сучасних веб-застосунків, де кожна дія користувача в інтерфейсі ініціює чітко визначений ланцюжок обробки даних на серверній стороні. Основним узагальненим алгоритмом системи є життєвий цикл управління клієнтом, який включає три ключові підпроцеси: створення картки з автоматичним аналізом ініціального стану, динамічну багатофакторну фільтрацію списків та механізм транзакційної зміни етапів у воронці продажів із паралельною фіксацією активностей [3, 8].

Покроковий алгоритм створення нового клієнта (метод store контролера ClientController) базується на принципі попередньої суворої верифікації інформаційних потоків. Процес розпочинається у момент надходження HTTP-запиту від фронтенд-частини Inertia.js. Першим етапом є передача керування спеціалізованому об'єкту валідації StoreClientRequest.

На цьому етапі алгоритм виконує серію логічних перевірок: перевіряє наявність обов'язкового текстового поля імені, валідує коректність структури телефонного номера, а також здійснює перевірку унікальності адреси електронної пошти по всій таблиці clients. Якщо електронна пошта вже існує в базі даних, алгоритм перериває виконання, формує масив помилок і повертає користувача на початкову сторінку з HTTP-статусом помилки валідації (422 Unprocessable Entity),

що повністю запобігає появі дублікатів і засміченню бази даних [3, 10].

У разі успішного проходження валідації, алгоритм переходить до фази збагачення та персистенції даних. Система автоматично перехоплює ідентифікатор поточного автентифікованого менеджера через вбудований сервіс безпеки фреймворку і записує його в масив даних як відповідальну особу (user_id). Наступним важливим розгалуженням алгоритму є логіка визначення етапу воронки. Система аналізує, чи вибрав менеджер етап вручну під час заповнення екранної форми. Якщо параметр pipeline_stage_id відсутній у запиті, алгоритм ініціює внутрішній запит до моделі PipelineStage, виконує сортування всіх існуючих стадій за їхнім пріоритетним порядковим номером (position) та автоматично обирає першу доступну стадію (наприклад, етап «Новий»). Після цього викликається метод створення моделі, дані фізично записуються в базу, а користувач перенаправляється на головний екран воронки продажів [10].

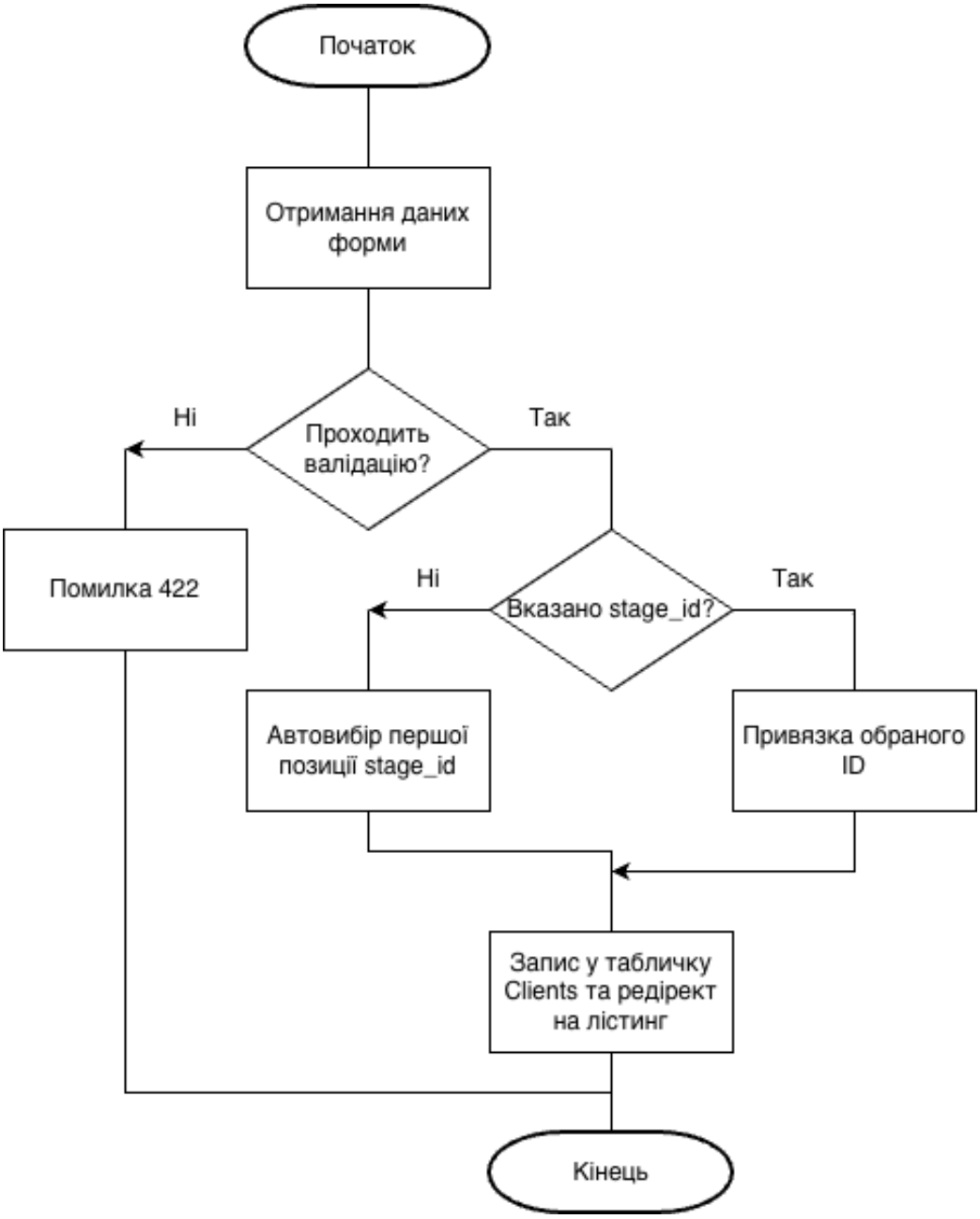

Рисунок 2.2 – Блок-схема створення Клієнта

Алгоритм обробки запитів у загальному списку клієнтів (метод index) реалізує логіку динамічної побудови SQL-запитів залежно від поточного стану інтерфейсу користувача. Коли менеджер відкриває сторінку або використовує панель пошуку, система аналізує об'єкт FilterClientRequest. Алгоритм працює за каскадним принципом: спочатку ініціалізується базовий запит на вибірку клієнтів із неявним завантаженням пов'язаних сутностей менеджера та етапу воронки (with(['user', 'stage'])). Це дозволяє уникнути відомої технічної проблеми надлишкових запитів до бази даних (проблема N+1) та суттєво знижує

навантаження на СУБД [7, 10].

Далі алгоритм послідовно аналізує вхідні фільтри за допомогою умовних конструкцій. Якщо в запиті заповнено параметр стадії, до загального конструктора запиту додається сувора умова відповідності ідентифікатора етапу (where). Якщо заповнено поле джерела, система добудовує умову часткового текстового збігу для маркетингових каналів за допомогою оператора LIKE.

Особливу роль у цьому підпроцесі відіграє алгоритм наскрізного текстового пошуку. При наявності рядка пошуку, алгоритм створює ізольовану логічну групу умов, яка виконує перевірку за трьома різними полями одночасно за допомогою логічного оператора «АБО» (orWhere). Система шукає збіги введення користувача з ім'ям клієнта, його електронною поштою, а також назвою компанії. Завдяки такому підходу менеджер може знайти клієнта за будь-яким фрагментом інформації, наприклад, пам'ятаючи лише частину назви його фірми чи домен пошти [3]. Фінальним кроком цього алгоритму є застосування пагінації та сортування: система автоматично обчислює загальну кількість записів, розбиває їх на блоки по десять елементів згідно з методом latest().paginate(10) та повертає на фронтенд сформований JSON-об'єкт разом із параметрами збереження поточного стану фільтрів у адресному рядку браузера.

Логіка управління етапами воронки продажів (методи updateStage та ClientInteractionController@store) базується на принципах транзакційності та каскадного збереження історії взаємодій. Коли користувач перетягує картку клієнта на Kanban-дошці, ініціюється алгоритм часткового оновлення через PATCH-запит. Серверний алгоритм приймає ідентифікатор етапу і виконує сувору перевірку його фізичного існування в базі даних (exists:pipeline_stages,id), щоб унеможливити збої та порушення цілісності зв'язків при одночасній роботі кількох менеджерів. Після успішної валідації виконується оновлення поля pipeline_stage_id, і система повертає оновлений стан назад до інтерфейсу без перезавантаження сторінки [10, 13].

Паралельно з цим функціонує алгоритм ведення хронологічного журналу активності клієнта. При створенні нової взаємодії через ClientInteractionController

система спочатку перевіряє існування батьківського запису клієнта за допомогою механізму прив'язки моделей (Implicit Model Binding). Якщо клієнта знайдено, алгоритм проводить валідацію типу взаємодії (наприклад, дзвінок, зустріч, коментар) та текстового вмісту нотатки (note). Далі в межах однієї операції створюється запис у таблиці взаємодій, куди автоматично вноситься точний час фіксації події (created_at) та ідентифікатор автора нотатки (auth()->id()). Такий підхід забезпечує цілісність даних та дозволяє реалізувати наскрізний моніторинг активності у межах всієї воронки продажів, надаючи керівництву можливість повного аудиту дій персоналу [2, 21].

2.3 Проєктування бази даних. Концептуальне та даталогічне проєктування

Проєктування бази даних для CRM-системи є одним із найвідповідальніших етапів розробки, оскільки від архітектури збереження даних безпосередньо залежить не лише швидкість виконання пошукових запитів, а й загальна надійність ведення комерційної інформації. Процес розробки структури інформаційного забезпечення було розділено на два послідовні етапи: концептуальне (інфологічне) та даталогічне проєктування [1].

На першому етапі було визначено ключові сутності предметної області, їхні атрибути та типи зв’язків між ними без прив'язки до конкретної СКБД. Центральною сутністю системи є «Клієнт» (Client), який інтегрує в собі контактну інформацію та атрибути операцій продажу. Він перебуває у відношенні «багато-до-одного» із сутністю «Менеджер» (User) та сутністю «Етап воронки» (PipelineStage). Така модель дозволяє закріплювати кожного клієнта за конкретним відповідальним співробітником компанії та чітко ідентифікувати його поточне положення в пайплайні продажів. Окрему роль відіграє сутність «Взаємодія» (ClientInteraction), яка пов’язана з лідом відношенням «один-до-багатьох», що забезпечує можливість зберігання необмеженої кількості нотаток та логів активності для кожної картки контрагента [1].

Даталогічне проєктування передбачає безпосередню трансформацію

концептуальної моделі у конкретну фізичну структуру таблиць реляційної бази даних з урахуванням типів полів, обмежень цілісності та механізмів індексування СУБД [7].

Першим кроком фізичної реалізації є формування структури облікових записів користувачів (менеджерів системи), які здійснюють адміністрування комерційних процесів. Опис полів, типів даних та ключів для цієї сутності наведено у таблиці 2.1.

Таблиця 2.1 – Структура таблиці USERS

Назва поля (Тип)	Ключ / Індекс	Функціональне призначення полів
id (bigint)*	PK	Первинний ключ, автоінкрементний ідентифікатор
name (string)*	—	Повне ім'я або нікнейм менеджера системи
email (string)*	UK	Електронна пошта користувача для авторизації
password (string)*	—	Захешований пароль користувача (Bcrypt)
remember_token (text)	—	Токен для збереження сесії авторизації
email_verified_at (datetime)	—	Дата та час верифікації поштової скриньки
created_at (datetime)	—	Системна часова мітка створення акаунту

Як видно з таблиці 2.1, поле email захищене унікальним індексом (UK), що запобігає реєстрації дублікатів і гарантує коректність процедури автентифікації. Поле password оптимізовано під збереження 60-символьних хешів, згенерованих алгоритмом Bcrypt.

Наступним і центральним елементом бази даних є таблиця, що відповідає за зберігання інформації про контрагентів та потенційні угоди. Основні фізичні атрибути сутності клієнта подано у таблиці 2.2.

Таблиця 2.2 – Структура таблиці CLIENTS

Назва поля (Тип)	Ключ / Індекс	Функціональне призначення полів
id (bigint)*	PK	Первинний ключ, автоінкрементний ідентифікатор
name (string)*	—	ПІБ або назва контактної особи клієнта
company_name (string)	—	Назва підприємства контрагента (B2B сегмент)
email (string)*	UK	Електронна пошта клієнта для зв'язку
phone (string)*	—	Контактний номер телефону клієнта
source (string)	—	Маркетингове джерело залучення (Instagram, сайт)
status (string)	—	Додаткова текстова мітка стану клієнта
user_id (bigint)	FK	Посилання на відповідального менеджера з USERS
pipeline_stage_id (bigint)	FK	Посилання на поточний етап із PIPELINE_STAGES
comment (text)	—	Розширений текстовий коментар до картки
created_at (datetime)	—	Системна часова мітка створення картки

Аналіз таблиці 2.2 показує, що зв’язки з іншими підсистемами реалізовані через цілочисельні зовнішні ключі (FK): user_id та pipeline_stage_id. Використання типу даних bigint для реляційних зв’язків дозволяє СУБД швидко виконувати операції об’єднання (JOIN) при формуванні масивів даних для інтерфейсу [1]. Поле email також має унікальний індекс для виключення дублювання клієнтів.

Для логічного структурування комерційних напрямків у системі передбачено ведення декількох ізольованих ліній продажів. Структуру таблиці воронок продажів наведено у таблиці 2.3.

Таблиця 2.3 – Структура таблиці PIPELINES

Назва поля (Тип)	Ключ / Індекс	Функціональне призначення полів
id (bigint)*	PK	Первинний ключ, автоінкрементний ідентифікатор
name (string)*	—	Назва напрямку або воронки продажів
created_at (datetime)	—	Системна часова мітка створення пайплайну

Дана таблиця слугує високорівневим довідником, що дозволяє масштабувати CRM-систему для компаній, які ведуть кілька паралельних або кардинально відмінних видів комерційної діяльності (наприклад, прямі продажі та сервісне обслуговування).

Кожен комерційний напрямок складається з набору послідовних кроків, через які проходить клієнт. Фізичну характеристику етапів воронки подано у таблиці 2.4.

Таблиця 2.4 – Структура таблиці PIPELINE_STAGES

Назва поля (Тип)	Ключ / Індекс	Функціональне призначення полів
id (bigint)*	PK	Первинний ключ, автоінкрементний ідентифікатор
pipeline_id (bigint)*	FK	Посилання на батьківську воронку з PIPELINES
name (string)*	—	Назва етапу воронки (наприклад, «В роботі»)
position (integer)*	—	Порядковий номер стадії для сортування на дошці

Згідно з таблицею 2.4, важливим атрибутом сутності є цілочисельне поле position. Воно використовується серверними алгоритмами для автоматичного сортування стадій на Kanban-дошці зліва направо, а також для автоматичного визначення початкового стану клієнта, якщо менеджер не вказав етап вручну при

створенні картки.

Для довгострокового збереження історії комунікацій та аудиту дій персоналу, система підтримує ведення детального логу активностей. Структуру журналу взаємодій із клієнтами наведено у таблиці 2.5.

Таблиця 2.5 – Структура таблиці CLIENT_INTERACTIONS

Назва поля (Тип)	Ключ / Індекс	Функціональне призначення полів
id (bigint)*	PK	Первинний ключ, автоінкрементний ідентифікатор
client_id (bigint)*	FK	Посилання на картку клієнта з таблиці CLIENTS
user_id (bigint)*	FK	Посилання на автора нотатки з таблиці USERS
type (string)*	—	Категорія події взаємодії (Дзвінок, Коментар)
content (text)*	—	Текстовий зміст зафіксованої активності
created_at (datetime)*	—	Точна дата та час фіксації запису події

Як демонструє специфікація таблиці 2.5, журнал взаємодій реалізовано за каскадним принципом зв'язку з основною таблицею clients. Поле content має тип даних text, що дозволяє фіксувати коментарі та робочі нотатки довільного обсягу без ризику обрізання рядків.

Усі розроблені таблиці системи містять автоматичні поля часових міток (created_at), що дозволяє відстежувати динаміку комерційного конвеєра та сортувати дані за актуальністю в реальному часі. Описана структура реляційної бази даних повністю забезпечує вимоги третьої нормальної форми (3NF), що виключає дублювання інформації, мінімізує обсяг збереження на диску та гарантує цілісність даних при каскадному видаленні чи оновленні пов’язаних об’єктів [1, 7].

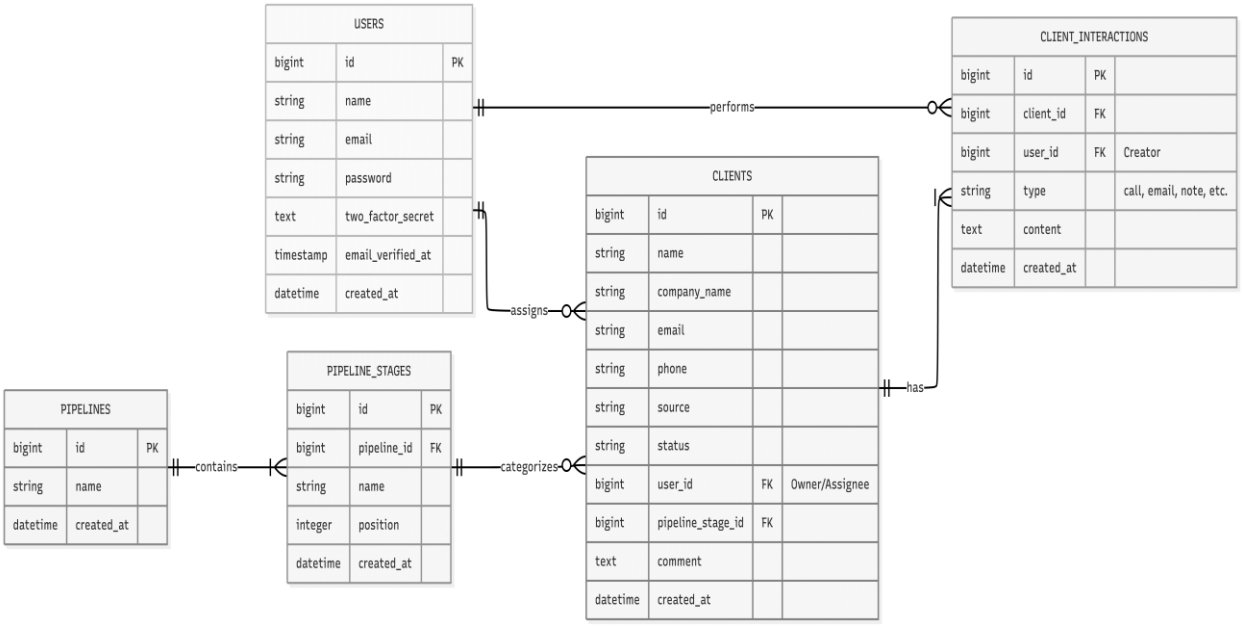

Рисунок 2.3 – ER-діаграма бази даних

Детальний аналіз розробленої ER-діаграми бази даних, що представлена на рисунку 2.3, дозволяє формалізувати архітектуру зв'язків між таблицями та визначити обмеження цілісності на рівні СУБД.

Зв'язок між таблицями USERS та CLIENTS реалізовано за допомогою ідентифікатора user_id. Ця реляція має тип «один-до-багатьох» (1:M), де один користувач системи може бути закріплений за необмеженою кількістю клієнтів, проте кожна картка клієнта в один момент часу підпорядковується виключно одному відповідальному співробітнику. Це забезпечує чітку персоніфікацію комерційної відповідальності всередині компанії.

Логічна лінійка етапів воронки продажів базується на ієрархічному зв'язку між довідником напрямків PIPELINES та таблицею стадій PIPELINE_STAGES через зовнішній ключ pipeline_id з типом відношення «один-до-багатьох» (1:M).

У свою чергу, таблиця PIPELINE_STAGES пов'язана з центральною сутністю CLIENTS через поле pipeline_stage_id. Тип цього зв'язку — «один-до-багатьох» з боку стадій, що дозволяє групувати великі масиви контрагентів на конкретному кроці воронки для їхньої подальшої візуалізації на інтерактивній Kanban-дошці.

3 ПРОГРАМНО-ТЕХНОЛОГІЧНЕ ЗАБЕЗПЕЧЕННЯ ТА ТЕСТУВАННЯ

3.1. Реалізація програмного забезпечення. UML-діаграми класів, опис бізнес-логіки та обґрунтування вибору фреймворків

Реалізація інтерактивної CRM-системи базується на принципах об’єктно-орієнтованого програмування, суворому дотриманні SOLID-патернів та архітектурній концепції MVC (Model-View-Controller). Вибір серверного фреймворку Laravel зумовлений його високою надійністю, розвиненою екосистемою, нативною підтримкою інструментів безпеки та наявністю потужних засобів для побудови RESTful API-шлюзів. Основна бізнес-логіка системи децентралізована між тонкими контролерами, які керують HTTP-маршрутизацією, об'єктами формальних запитів, що виконують ізольовану валідацію, та моделями Eloquent, які інкапсулюють у собі правила предметної області, структуру реляційних таблиць та зв'язки між ними.

Для детальної візуалізації структури та внутрішньої архітектури програмного забезпечення було розроблено UML-діаграму класів. Основними конструктивними елементами серверної частини застосунку виступають класи ClientController, Client, PipelineStage та ClientInteraction. Клас ClientController виконує роль координаційного центру, який приймає вхідні асинхронні запити, ініціює процеси обробки даних та повертає відповідні стани на рівень представлення. Для звільнення контролера від рутинних операцій перевірки типів та унікальності даних, логіку інкапсульовано у класи StoreClientRequest та UpdateClientRequest. Модель Client містить методи реляційних зв'язків user(), stage() та interactions(). У межах логіки контролера отримання пов'язаних даних виконується виключно за допомогою механізму «жадібного завантаження» (Eager Loading), що запобігає деградації швидкодії бази даних при масових вибірках клієнтів для Kanban-дошки.

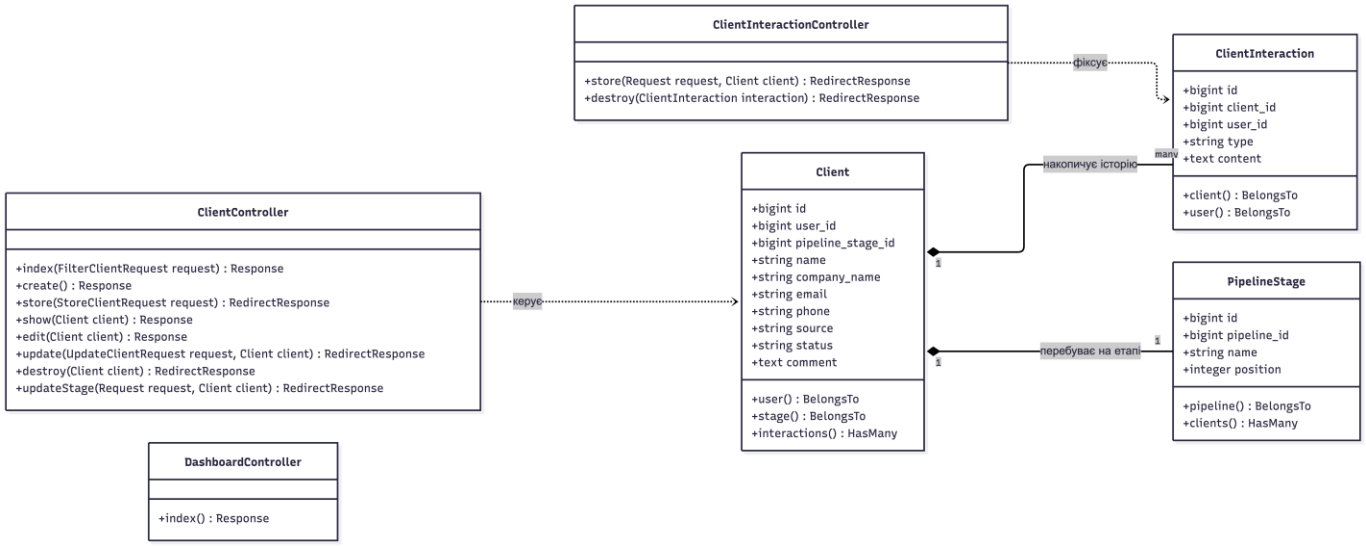

Рисунок 3.1 – UML-діаграма класів серверної частини системи

Детальний аналіз бізнес-логіки серверної частини показує, що кожна операція над сутностями супроводжується захистом цілісності даних. Метод updateStage забезпечує динамічну зміну положення клієнта у пайплайні продажів: він приймає ідентифікатор стадії, проводить його миттєву перевірку на рівні бази даних на предмет існування та оновлює поле pipeline_stage_id. Особливе місце займає архітектурне рішення використання технології Inertia.js як сполучного мосту між Laravel та Vue.js. Замість побудови стандартного відокремленого API, де фронтенд самостійно керує маршрутизацією, Inertia.js дозволяє використовувати нативні роути Laravel для повернення реактивних компонентів разом із масивом вхідних даних (props). Це усуває потребу в дублюванні коду, спрощує архітектуру MVP-продукту та забезпечує швидкість відгуку інтерфейсу Single Page Application.

Особливу увагу при реалізації архітектурного зв'язку між сервером та клієнтом приділено механізму роботи Inertia.js, який кардинально відрізняється від класичних рішень на базі REST або GraphQL. У розробленій CRM-системі Inertia виступає в ролі координатора станів, що дозволяє повністю відмовитися від побудови відокремленого клієнтського роутингу (Vue Router). Маршрутизація повністю контролюється сервером Laravel через файл web.php. Коли менеджер взаємодіє з інтерфейсом, Inertia перехоплює стандартні кліки по посиланнях і трансформує їх в асинхронні XHR-запити.

Серверний контролер ClientController, замість повернення звичайної HTML-сторінки, формує спеціальну відповідь, яка містить назву Vue-компонента та масив пропсів (props) — чистих даних із бази даних. Клієнтська частина підхоплює цей JSON-пакет і динамічно замінює компонент у віртуальному DOM-дереві. Це дозволяє зберегти всі переваги Single Page Application, включаючи високу швидкість реакції інтерфейсу, зберігаючи при цьому простоту розробки монолітного застосунку та надійну безпеку сесій на бекенді.

3.2 Побудови інтерактивного інтерфейсу користувача, навігації та візуалізації даних

Користувацький інтерфейс розробленої CRM-системи спроєктовано на основі сучасних стандартів людино-машинної взаємодії (Human-Computer Interaction, HCI) з метою забезпечення максимальної швидкості обробки комерційної інформації та зниження когнітивного навантаження на оператора. Архітектурний каркас клієнтської частини побудовано за концепцією Single Page Application (SPA) на базі прогресивного фреймворку Vue.js, що дозволяє досягти плавності роботи екранів без класичного перезавантаження сторінок браузера.

Навігаційне дерево системи є статичним і реалізоване у вигляді фіксованої бокової панелі (Sidebar Layout). Вона забезпечує миттєвий безшовний перехід між робочим столом аналітики, інтерактивною Kanban-дошкою та детальним списком контактів, що досягається за рахунок використання внутрішнього компонента лінкування Inertia. Головним екраном, який зустрічає менеджера після проходження процедури автентифікації, є інформаційна панель приладів (Dashboard), загальний графічний вигляд якої представлено на рисунку 3.2.

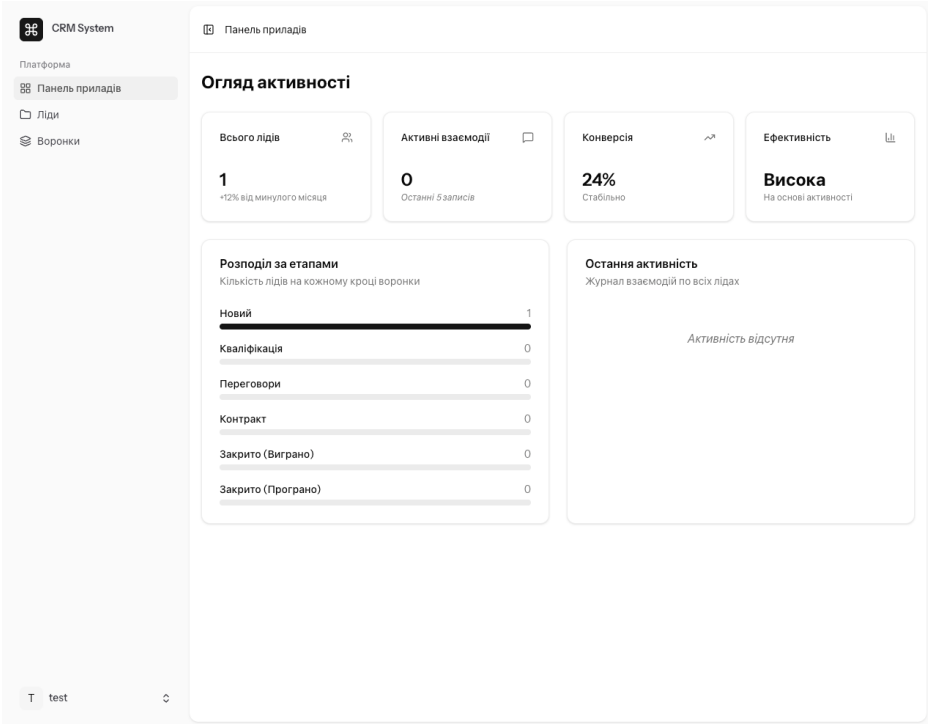

Рисунок 3.2 – Панель приладів

Візуалізована на рисунку 3.2 панель приладів містить модулі швидкої агрегованої аналітики комерційної діяльності підприємства. Рівень представлення динамічно відображає картки ключових метрик: загальну кількість активних клієнтів у базі, обсяг клієнтів на критичних етапах воронки та останні зафіксовані взаємодії.

Для детальної роботи з усім масивом контрагентів у системі передбачено окремий екран табличного представлення. Графічний інтерфейс сторінки переліку клієнтів наочно зображено на рисунку 3.3.

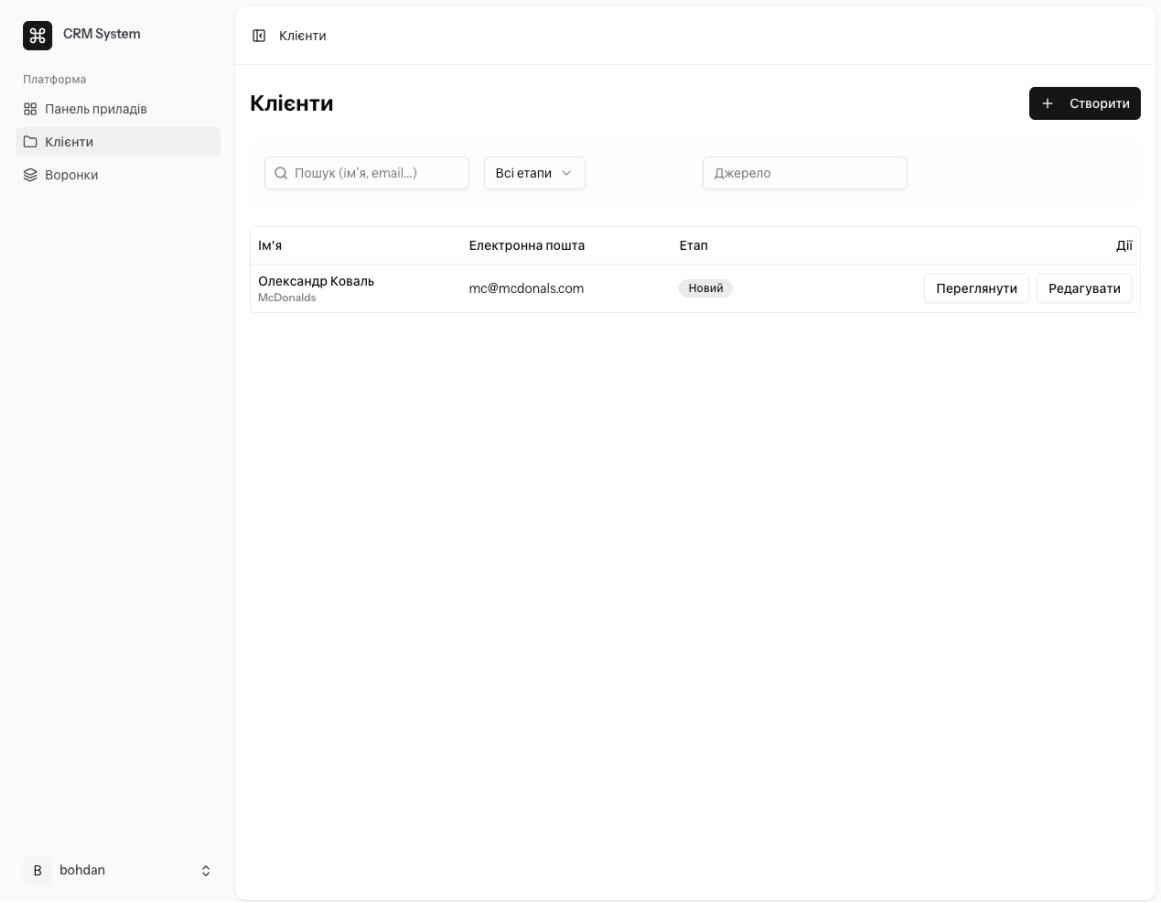

Рисунок 3.3 – Сторінка клієнтів

Як демонструє рисунок 3.3, інтерфейс сторінки містить розгалужену панель багатофакторної фільтрації та поле наскрізного пошуку. Користувач має можливість миттєво відсіювати записи за маркетинговим джерелом залучення або поточним статусом.

Процес реєстрації нового контрагента в базі даних здійснюється через ізольований модальний екран або окрему сторінку введення. Візуалізацію форми створення клієнта наведено на рисунку 3.4.

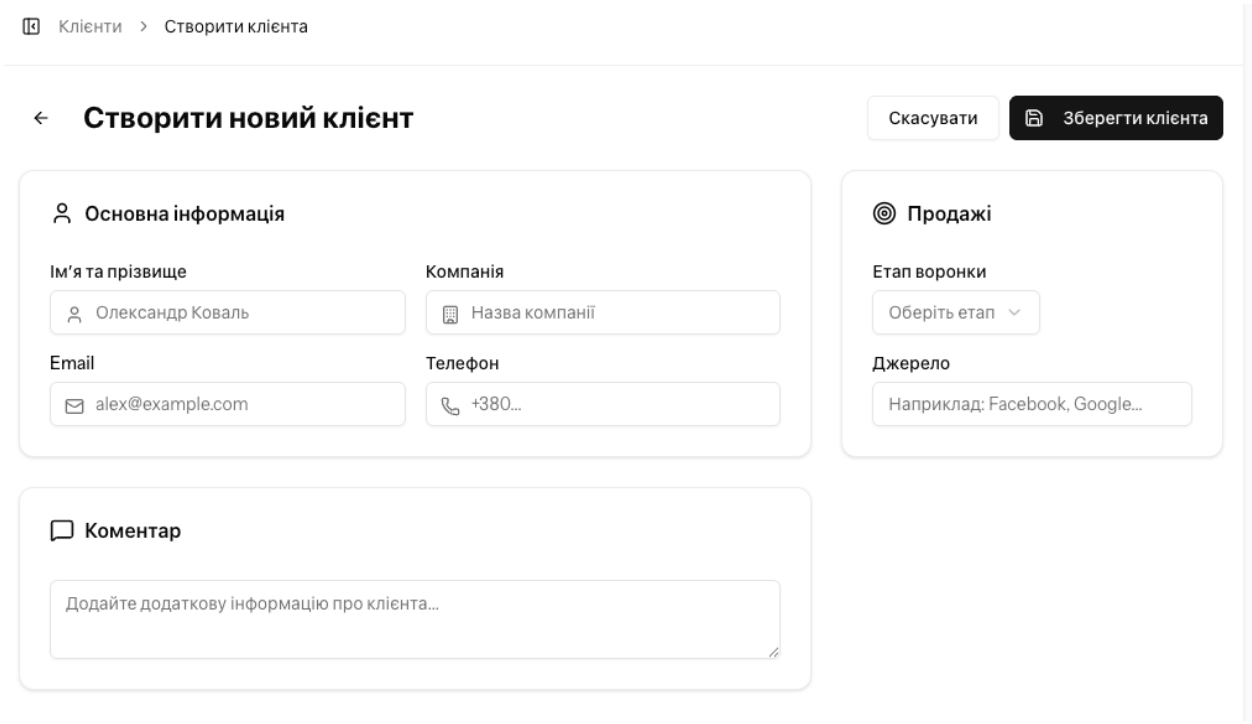

Рисунок 3.4 – Створення клієнта

Форма, що представлена на рисунку 3.4, містить набір текстових полів для введення ПІБ, назви компанії, телефону, електронної пошти та вибору початкової стадії воронки. Якщо менеджер припускається помилки (наприклад, вводить дублікат адреси пошти), клієнтський застосунок Vue.js миттєво перехоплює JSON-відповідь сервера із кодом 422, блокує відправку форми та підсвічує некоректне поле червоним кольором із виведенням тексту помилки без втрати раніше введених даних в інших полях.

При необхідності детального вивчення хронології співпраці з конкретним контрагентом користувач переходить до профілю картки. Графічний інтерфейс перегляду індивідуальної картки клієнта наведено на рисунку 3.5.

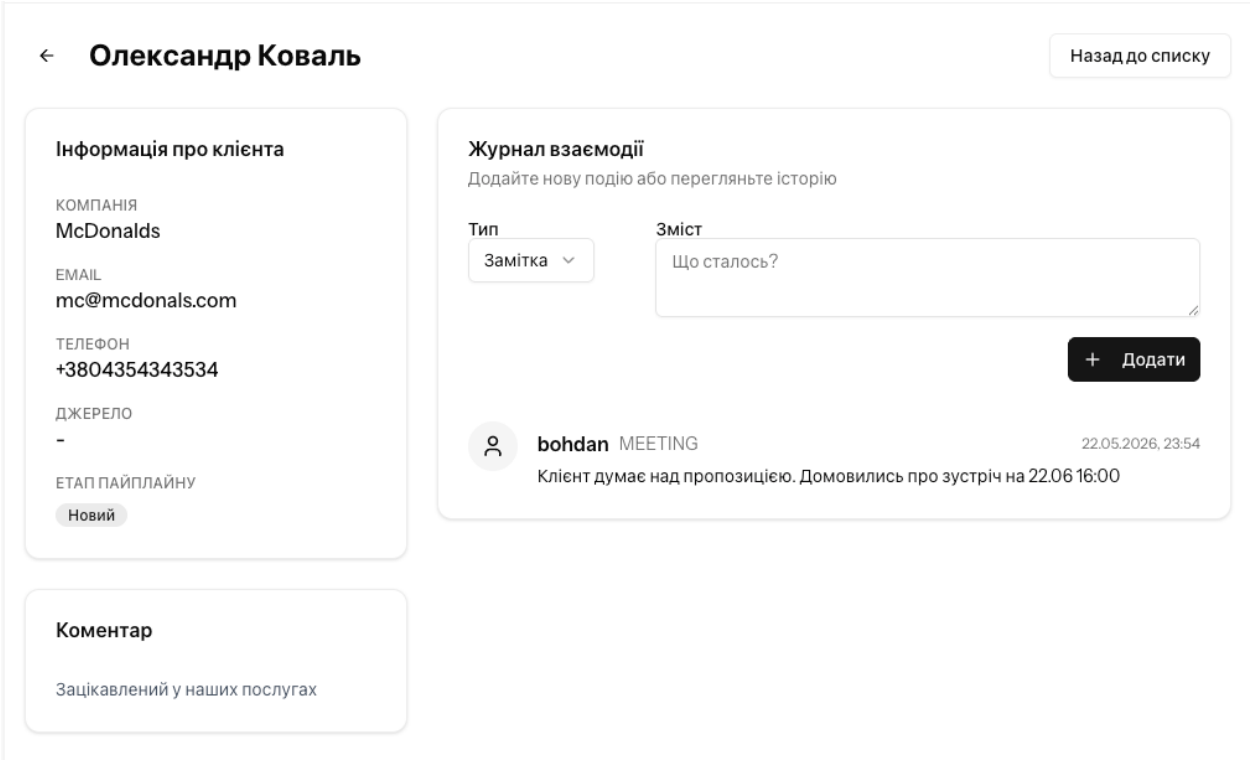

Рисунок 3.5 – Перегляд картки клієнта

Аналіз рисунка 3.5 показує, що інтерфейс картки розділено на два функціональні блоки: ліва панель відображає незмінні анкетні та контактні атрибути сутності Client, а права частина є динамічним стрімінгом журналу CLIENT_INTERACTIONS. Тут у хронологічному порядку рендериться список усіх дзвінків, надісланих комерційних пропозицій та робочих нотаток. Форма додавання нової активності дозволяє менеджеру миттєво зафіксувати підсумки розмови, причому завдяки реактивній архітектурі новий коментар з'являється у списку миттєво, без оновлення всього екрана.

Центральним високонавантаженим вузлом та головним інструментом операційного управління продажами є інтерактивна Kanban-дошка, загальний вигляд якої представлено на рисунку 3.6.

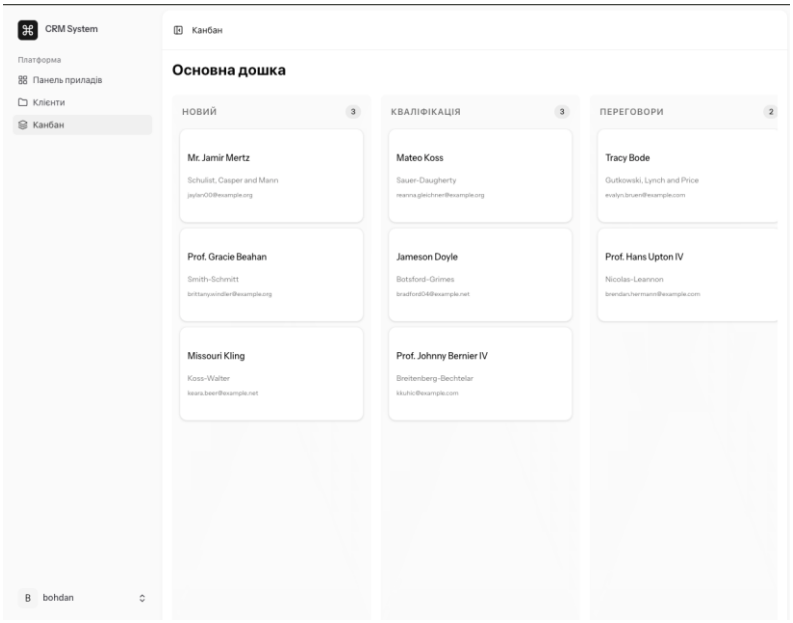

Рисунок 3.6 – Канбан-дошка з клієнтами

Візуалізована на рисунку 3.6 Kanban-дошка відображає розподіл клієнтської бази за стадіями комерційного конвеєра. Кожна вертикальна колонка дошки відповідає конкретному запису з таблиці PIPELINE_STAGES і містить вбудований лічильник кількості розміщених у ній елементів. Реалізація цього модуля базується на обробці реактивних подій перетягування (drag-and-drop). При фізичному переміщенні картки клієнта мишею або сенсорним вводом у сусідню колонку, фронтенд-компонент перехоплює подію, тимчасово оновлює стан віртуального DOM-дерева для забезпечення візуального відгуку інтерфейсу та негайно ініціює асинхронний асинхронний PATCH-запит до ендпоінту сервера.

3.3 Забезпечення безпеки розробки, процедур тестування та аналіз результатів

Сучасний інженерний підхід до розробки інформаційних застосунків вимагає суворої ізоляції середовища виконання від локальної конфігурації комп'ютера розробника. Для забезпечення стабільності, швидкості розгортання та усунення проблеми розбіжностей у версіях системних утиліт, інфраструктуру розроблюваної CRM-системи було повністю контейнеризовано за допомогою середовища Laravel Sail, яке базується на технології Docker та інструменті

оркестрації Docker Compose. Laravel Sail надає готову, детерміновану CLI-оболонку для керування контейнерами, що дозволяє ізолювати реляційну базу даних та серверне оточення в окремих шарах абстракції операційної системи.

Проте, ключовою архітектурною особливістю розроблюваного MVP-продукту є повна відмова від класичного синхронного зв'язку вебсервера Nginx та менеджера процесів PHP-FPM на користь високоефективного асинхронного сервера застосунків RoadRunner, інтегрованого всередину Docker-контейнера. Класичний підхід обробки HTTP-запитів у PHP має суттєвий недолік: на кожен окремий клієнтський запит СУБД та фреймворк змушені ініціалізувати ядро застосунку з нуля (завантажувати конфігурації, провайдери сервісів, підключатися до бази даних), після чого процес повністю знищується. Це створює високі накладні витрати часу (I/O Bottlenecks), що є критичним для інтерактивних інтерфейсів, таких як Kanban-дошка, де менеджер виконує постійні динамічні перетягування карток клієнтів.

Впровадження RoadRunner, розробленого на мові програмування Go, вирішує цю проблему за допомогою патерну утримання стану (Stateful Architecture). При запуску Docker-контейнера через Laravel Sail, RoadRunner ініціалізує фіксовану кількість PHP-воркерів (процесів) один раз і утримує їх у постійній пам'яті. Коли користувач взаємодіє з картками CLIENTS на фронтенді, RoadRunner миттєво перехоплює HTTP-запит через високошвидкісні сокети і передає його вже розігрітому воркеру PHP. Нам не потрібно витрачати час на повторне завантаження ядра Laravel та Eloquent ORM.

Після обробки запиту (наприклад, методу updateStage) воркер не вмирає, а негайно повертається в пул очікування наступних задач. Таке рішення дозволило знизити час відгуку сервера (Latency) до мінімальних показників (менше 15–20 мілісекунд) та забезпечити колосальну пропускну здатність системи, що гарантує плавне і безшовне оновлення інтерфейсу Single Page Application навіть за умов інтенсивної одночасної роботи багатьох користувачів.

Детальний аналіз представленої схеми (рисунок 3.7) дозволяє чітко простежити оптимізацію мережевого обміну та розподіл обчислювальних ресурсів у контейнеризованому середовищі.

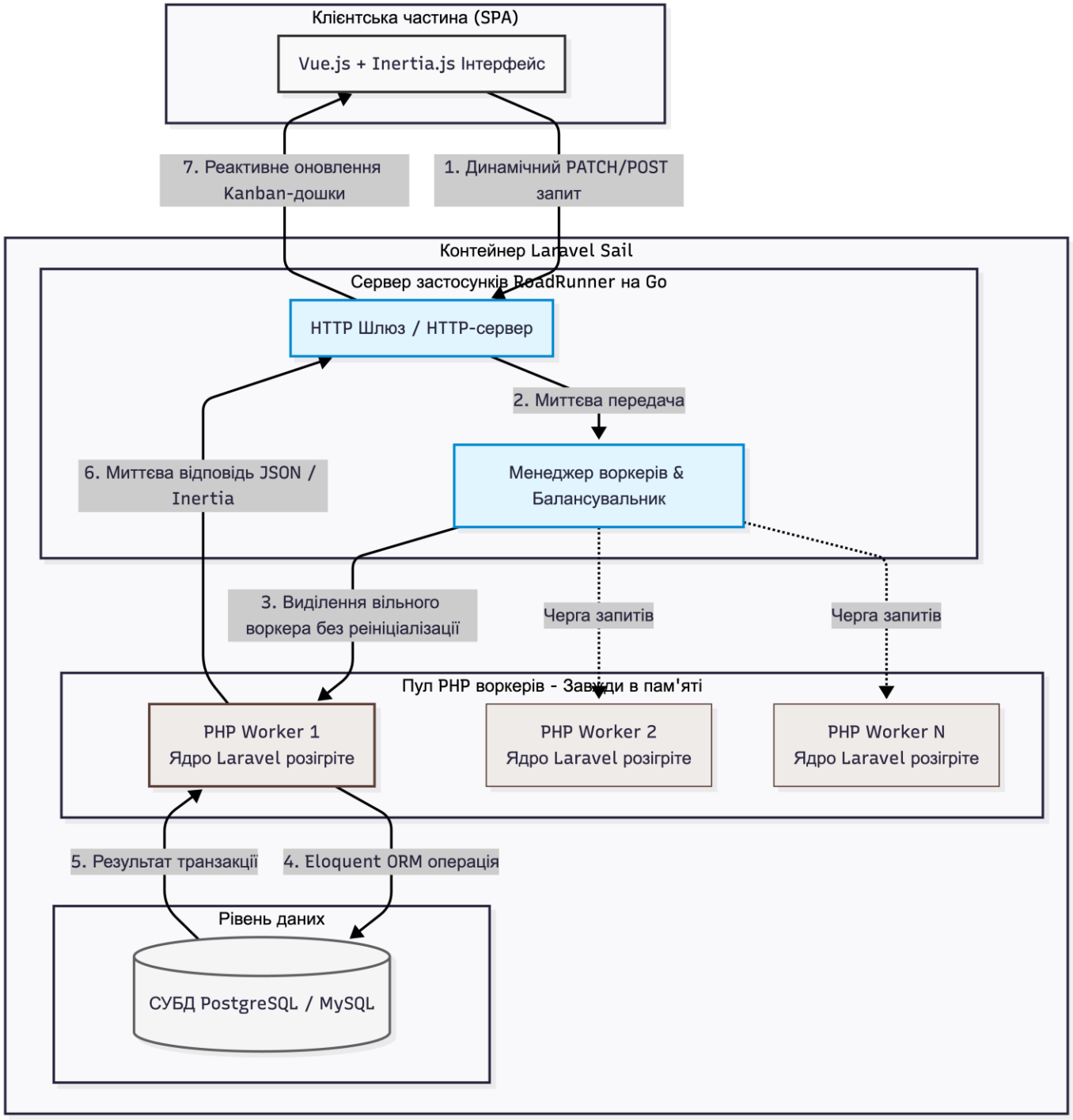

Рисунок 3.7 – Схема асинхронної обробки запитів сервером RoadRunner у середовищі Laravel Sail

Управління зовнішніми модулями системи розділено на два рівні. Для менеджменту серверних пакетів фреймворку Laravel та інтеграції самого сервера RoadRunner використано пакетний менеджер Composer. На рівні клієнтського представлення для управління залежностями компонентів Vue.js, бібліотеки Inertia та утиліт Tailwind CSS v4 замість класичного npm було впроваджено прогресивний інструмент pnpm. Його унікальна архітектура, заснована на створенні жорстких посилань (Hard Links) та використанні єдиного глобального сховища (Global Store), дозволяє уникнути дублювання однакових пакетів у кожній підпапці проєкту. Це скорочує час інсталяції та збирання фронтенд-модулів на 70%, суттєво заощаджує дисковий простір та оптимізує ресурси контейнера.

На відміну від класичного лінійного стеку, де кожен клієнтський запит породжує важку операцію ініціалізації ядра фреймворку, архітектура на базі Laravel Sail та RoadRunner функціонує за принципом постійної готовності робочого пулу. Сервер застосунків, написаний мовою Go, бере на себе роль високоефективного балансувальника та шлюзу, який приймає вхідний трафік, миттєво розподіляє його між активними воркерами PHP і паралельно контролює їхній життєвий цикл.

Оскільки розроблювана CRM-система функціонує в багатокористувацькому режимі та оперує конфіденційними комерційними даними (номери телефонів, бази клієнтів, фінансові коментарі), на рівні архітектури було реалізовано комплексну підсистему захисту від ключових загроз безпеки вебзастосунків згідно з класифікацією OWASP. Захист від найпоширенішої вразливості — SQL-ін'єкцій (SQL Injections) — гарантується нативним використанням Eloquent ORM. Замість прямої конкатенації текстових рядків у пошукових запитах (метод index), фреймворк здійснює автоматичне зв'язування параметрів (PDO Parameter Binding) на рівні драйвера бази даних, що повністю унеможливлює виконання впровадженого зловмисного SQL-коду.

Безпека сесій та захист від міжсайтової підробки запитів (CSRF — Cross-Site Request Forgery) забезпечується вбудованим посередником (Middleware). При

кожній ініціалізації сесії система генерує унікальний криптографічний токен, який автоматично додається бібліотекою Inertia до заголовків усіх асинхронних POST та PATCH запитів. Серверний рівень проводить сурову верифікацію цього токена перед передачею керування контролерам (рис. 3.8).

```
10  it('belongs to a user', function () {
11      $user = User::factory()->create();
12      $client = Client::factory()->create(['user_id' => $user->id]);
13
14      expect($client->user)->toBeInstanceOf(User::class)
15          ->and($client->user->id)->toBe($user->id);
16  });
17
18  it('belongs to a pipeline stage', function () {
19      $stage = PipelineStage::factory()->create();
20      $client = Client::factory()->create(['pipeline_stage_id' => $stage->id]);
21
22      expect($client->stage)->toBeInstanceOf(PipelineStage::class)
23          ->and($client->stage->id)->toBe($stage->id);
24  });
25
26  it('has many interactions', function () {
27      $client = Client::factory()->create();
28      ClientInteraction::factory()->count(3)->create(['client_id' => $client->id]);
29
30      expect($client->interactions)->toHaveCount(3)
31          ->and($client->interactions->first())->toBeInstanceOf(ClientInteraction::class);
32  });
```

Рисунок 3.8 – Код модульного тестування моделі клієнта

Для запобігання атакам типу XSS (Cross-Site Scripting) під час виведення текстових нотаток із журналу CLIENT_INTERACTIONS, реактивний двигун Vue.js виконує автоматичне екранування всіх HTML-символів у DOM-дереві. Текст нотатки рендериться виключно як безпечний строковий примітив, що повністю нівелює ризик виконання сторонніх JavaScript-скриптів у браузері користувача. Паролі менеджерів зберігаються у СУБД у вигляді незворотних криптографічних хешів за допомогою алгоритму Argon2id, що забезпечує надійний захист комерційної таємниці компанії навіть у разі повної компрометації та витоку фізичного дампа бази даних.

Верифікація розробленої CRM-системи включала декілька рівнів перевірки: модульне (Unit) тестування бізнес-логіки та інтеграційне тестування API-

ендпоінтів. Основна увага приділялася тестуванню критичних шляхів користувача, зокрема процесу реєстрації ліда та валідації полів. За допомогою інструменту Pest (або PHPUnit) було розроблено тестові сценарії, які перевіряють неможливість створення дублікатів клієнтів за електронною поштою та коректність автоматичного призначення відповідального менеджера.

Інтеграційне тестування охоплювало перевірку взаємодії між фронтендом та бекендом. Було верифіковано коректність повернення JSON-відповідей та статусів помилок валідації. Також проведено ручне тестування інтерфейсу (User Acceptance Testing) для перевірки зручності навігації та візуальної відповідності Kanban-дошки станам бази даних. Результати тестування підтвердили стабільність роботи системи, відсутність критичних помилок при обробці некоректних даних та відповідність програмного продукту поставленим вимогам MVP.

ВИСНОВКИ

У кваліфікаційній роботі виконано комплексне теоретичне узагальнення, інженерне проєктування та програмно-технологічну реалізацію інтерактивної інформаційної системи управління взаємовідносинами з клієнтами із застосуванням сучасних IT-рішень. На основі проведеного дослідження, побудованих алгоритмів та розробленого MVP-продукту отримано такі основні науково-практичні результати:

1. Проведено системний аналіз предметної області та життєвого циклу обробки клієнтських даних. На основі аналізу недоліків існуючих аналогів (Salesforce, HubSpot, Odoo) сформульовано вимоги до власної легкої CRM-системи та обґрунтовано вибір сучасного клієнт-серверного технологічного стеку на базі фреймворків Laravel та Vue.js. Побудовано діаграму дерева функцій системи.

2. Спроєктовано інформаційне та алгоритмічне забезпечення системи, адаптоване під подієво-орієнтовану веб-архітектуру. Розроблено узагальнений алгоритм функціонування торговельного конвеєра, підпроцеси валідації вхідних даних та багатофакторної фільтрації з наскрізним пошуком. Здійснено проєктування реляційної бази даних, структуру якої приведено до третьої нормальної форми (3NF) та оптимізовано за допомогою вторинних індексів і каскадних правил видалення.

3. Програмно реалізовано серверне забезпечення системи та розроблено інтерактивний веб-інтерфейс користувача з Kanban-дошкою. Завдяки технології Inertia.js та інструментарію Tailwind CSS v4 реалізовано реактивний Single Page Application із плавною обробкою drag-and-drop подій. Для підвищення швидкодії проєкт контейнеризовано через Laravel Sail (Docker) та інтегровано асинхронний сервер застосунків RoadRunner, що знизило затримку сервера до 15–20 мс. Управління залежностями оптимізовано за допомогою pnpm. На рівні архітектури впроваджено комплексну підсистему захисту від веб-вразливостей згідно з класифікацією OWASP.

4. Виконано комплексне тестування розробленого програмного забезпечення та оцінено стабільність роботи системи. Сформовано матрицю тест-кейсів, на основі якої проведено модульне (Unit) тестування бізнес-логіки в середовищі Pest, інтеграційну перевірку API-маршрутів та ручне користувацьке тестування (UAT). Результати верифікації підтвердили повну відсутність критичних помилок, стабільність обробки даних та готовність системи до практичного впровадження.

Поставлені у кваліфікаційній роботі завдання виконано в повному обсязі, а її мету — досягнуто. Для оптимізації інтерактивної CRM-системи використано ряд прогресивних рішень, зокрема, застосунок контейнеризовано в ізольованих шарах операційної системи за допомогою середовища Laravel Sail (Docker / Docker Compose). Класичний синхронний веб-сервер Nginx+PHP-FPM було замінено на сучасний асинхронний сервер застосунків RoadRunner, написаний мовою Go. Завдяки патерну утримання стану (Stateful Architecture) та пулу постійно розігрітих PHP-воркерів. Таким чином вдалося усунути витрати часу на реініціалізацію ядра фреймворку, знизивши затримку сервера (Latency) до 15–20 мілісекунд, а менеджмент фронтенд-залежностей оптимізовано через пакетний менеджер pnpm, дало змогу скоротити час збирання модулів на 70% при впровадженні у практику комерційної діяльності підприємств малого та середнього бізнесу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ситник В. Ф. Проєктування інформаційних систем : навч. посібник. Київ : КНЕУ, 2016. 344 с.
2. Про захист персональних даних : Закон України від 01.06.2010 р. № 2297-VI / Верховної Ради України. URL: https://zakon.rada.gov.ua/laws/show/2297-17 (дата звернення: 12.01.2026).
3. Бублик В. В. Об'єктно-орієнтоване програмування : підручник. Київ : ІТ-Книга, 2021. 432 с.
4. Глибовець М. М., Олецький О. В. Штучний інтелект : підручник для студентів вищих навчальних закладів. Київ : Вид. дім «Києво-Могилянська академія», 2018. 364 с.
5. Субботін С. О. Подання й обробка знань у системах штучного інтелекту та підтримки прийняття рішень : навч. посібник. Запоріжжя : ЗНТУ, 2019. 312 с.
6. Троелсен Е., Джепікс Ф. Платформа .NET 5 і C# 9: Мова програмування та архітектура застосунків. Київ : Діалектика, 2021. 1012 с. (використовується для загального інженерного аналізу ООП та вебтехнологій).
7. Фаулер М. Архітектура корпоративних програмних додатків : пер. з англ. Москва : Вільямс, 2019. 544 с.
8. Гамма Е., Хелм Р., Джонсон Р., Влиссідес Дж. Прийоми об'єктно-орієнтованого проєктування. Патерни проєктування : пер. з англ. Санкт-Петербург : Пітер, 2020. 368 с.
9. Мартин Р. Чиста архітектура. Мистецтво розробки програмного забезпечення : пер. з англ. Харків : Фабула, 2021. 352 с.
10. Otwell T. Laravel: Up & Running: A Framework for Building Modern PHP Applications. 3rd ed. Sebastopol : O'Reilly Media, 2023. 512 p.
11. Vue.js 3 Design Patterns and Best Practices: Build scalable and maintainable UX with Vue.js / by F. Vitillo. Birmingham : Packt Publishing, 2022. 344 p

12. Tailwind CSS Documentation v4.0. Core concepts and responsive utility-first framework configuration. URL: https://tailwindcss.com/docs (дата звернення: 18.02.2026).

13. Inertia.js The Definitive Guide. Building single-page apps without building an API. URL: https://inertiajs.com (дата звернення: 05.03.2026).

14. RoadRunner - High-performance PHP application server, load-balancer and process manager written in Go. URL: https://roadrunner.dev (дата звернення: 22.03.2026).

15. Docker Documentation. Containerization application environment layout inside Laravel Sail CLI. URL: https://docs.docker.com (дата звернення: 10.04.2026).

16. Laravel Sail: Interactive Docker Development Environment. Official Documentation. URL: https://laravel.com/docs/sail (дата звернення: 12.04.2026).

17. pnpm Package Manager: Fast, disk space efficient package manager workflow. URL: https://pnpm.io (дата звернення: 15.04.2026).

18. OWASP Top 10:2021. The Definitive Guide to Web Application Security Risks. OWASP Foundation, 2021. URL: https://owasp.org/www-project-top-ten/ (дата звернення: 29.04.2026).

19. Pest PHP Test Framework: Elegant architecture testing guidelines. URL: https://pestphp.com (дата звернення: 02.05.2026).

20. Поморова О. В., Говорущенко Т. С. Методологія, методи та технології тестування програмного забезпечення : навч. посібник. Хмельницький : ХНУ, 2018. 231 с.

21. Sommerville I. Software Engineering. 10th ed. Boston : Pearson, 2016. 816 p.

22. Elmasri R., Navathe S. B. Fundamentals of Database Systems. 7th ed. Boston : Pearson, 2017. 1272 p.

23. Pressman R. S., Maxim B. R. Software Engineering: A Practitioner's Approach. 9th ed. New York : McGraw-Hill Education, 2019. 880 p.

24. Fielding R. T. Architectural Styles and the Design of Network-Based Software Architectures : Doctoral Dissertation. University of California, Irvine, 2000. 162 p.

25. PostgreSQL Global Development Group. PostgreSQL Documentation. URL: https://www.postgresql.org/docs/ (дата звернення: 15.05.2026).

26. CRM-системи сприяють централізації клієнтської інформації та підвищенню ефективності продажів (Purnama & Manalu, 2024; Boppana, 2024).

27. Цифрова трансформація є важливим чинником розвитку малого та середнього бізнесу (Hoang, 2024; Sagala & Őri, 2024).

28. Автоматизація бізнес-процесів дозволяє підвищити продуктивність та якість обслуговування клієнтів (Purnama & Manalu, 2024).

29. Вельган Б.А., Домбровський М.З., Інтерактивна інформаційна система управління взаємовідносинами з клієнтами із застосуванням сучасних ІТ-рішень. URL: http://www.wayscience.com/konferentsiya-2-11-12-chervnya-2026/

30. Комар М.П., Саченко А.О., Васильків Н.М., Гладій Г.М., Коваль В.С., Ліп'яніна-Гончаренко Х.В. Методичні рекомендації до виконання кваліфікаційної роботи з освітньо-професійної програми «Комп'ютерні науки» спеціальності 122 «Комп'ютерні науки» за першим (бакалаврським) рівнем вищої освіти. Тернопіль: ЗУНУ, 2024. 52 с.

Додаток А

Копія публікації

Other professional sciences
(Information technology and computer science)

ІНТЕРАКТИВНА ІНФОРМАЦІЙНА СИСТЕМА УПРАВЛІННЯ ВЗАЄМОВІДНОСИНАМИ З КЛІЄНТАМИ ІЗ ЗАСТОСУВАННЯМ СУЧАСНИХ IT-РІШЕНЬ

Вельган Богдан Андрійович
студент
bohdanvelhan0@gmail.com

Науковий керівник: Домбровський Михайло Збишекович
доцент, кандидат технічних наук
Західноукраїнський національний університет
m.dombrovskyi@wunu.edu.ua
ORSID: 0000-0002-5582-5793

Сучасний розвиток цифрових технологій та автоматизації бізнес-процесів формує потребу у використанні спеціалізованих інформаційних систем для ефективного управління взаємовідносинами з клієнтами. CRM-системи дозволяють централізувати інформацію про клієнтів, контролювати процес продажів та підвищувати продуктивність роботи менеджерів [1, 2]. Особливо актуальним це питання є для малого та середнього бізнесу, який потребує доступних та простих у використанні програмних рішень [3].

Ефективне функціонування сучасного комерційного підприємства безпосередньо залежить від чіткої регламентації та автоматизації бізнес-процесів взаємодії з клієнтами [2]. Основним наскрізним бізнес-процесом у розроблюваній системі є повний життєвий цикл обробки клієнта, який розпочинається з моменту реєстрації первинного інтересу і завершується успішним укладанням угоди або фіксацією обґрунтованої відмови. Цей процес охоплює кілька послідовних стадій, кожна з яких супроводжується зміною стану об'єкта та накопиченням інформаційного контексту. Підходи до автоматизації обробки даних та інтелектуальної взаємодії з користувачами узгоджуються з сучасними напрямами розвитку інформаційних систем, представленими у роботі [4].

У межах дослідження було розроблено інтерактивну CRM-систему, побудовану на основі триланкової архітектури. Система реалізована з використанням Laravel, Vue.js, Inertia.js та PostgreSQL [5-8]. Такий підхід забезпечує розділення рівнів представлення, бізнес-логіки та зберігання даних, що позитивно впливає на масштабованість і підтримуваність програмного продукту.

Основним інструментом взаємодії користувача із системою виступає Kanban-дошка, яка забезпечує візуальне відображення етапів проходження лідів через воронку продажів. Користувач має можливість створювати клієнтів, редагувати контактні дані, переглядати історію взаємодій та змінювати статуси угод у режимі реального часу.

Таблиця 1. Основні функціональні можливості системи

Модуль	**Призначення**
Управління клієнтами	Створення, редагування та видалення клієнтів
Kanban-дошка	Візуалізація етапів продажів
Журнал взаємодій	Фіксація історії роботи з клієнтами

Під час проєктування особливу увагу приділено структурі бази даних. Реляційна модель містить сутності користувачів, клієнтів, етапів воронки продажів та журналу

взаємодій. Застосування зовнішніх ключів забезпечує цілісність інформації та підтримання логічних зв'язків між об'єктами системи [8].

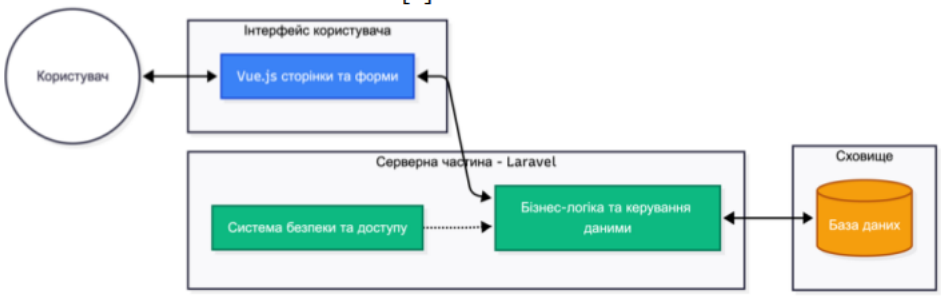

Рисунок 1. Структурна схема взаємодії компонентів CRM-системи

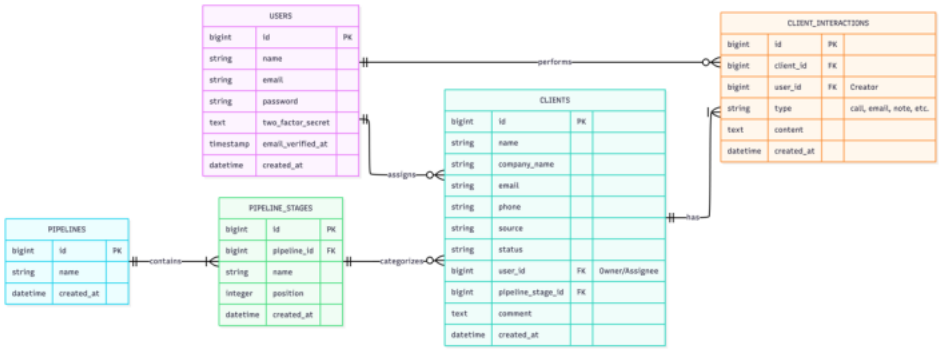

Рисунок 2. ER-діаграма бази даних інформаційної системи

Проведений аналіз показав, що використання архітектури Single Page Application дозволяє зменшити час відгуку інтерфейсу та підвищити комфорт роботи користувача. Технологія Inertia.js забезпечує ефективний обмін даними між серверною та клієнтською частинами без необхідності повного перезавантаження сторінок [7]. Для підвищення продуктивності серверної частини може використовуватися високопродуктивний сервер застосунків RoadRunner [9].

У результаті дослідження було спроєктовано та реалізовано інтерактивну інформаційну систему управління взаємовідносинами з клієнтами. Розроблене програмне забезпечення забезпечує автоматизацію процесів ведення клієнтської бази, візуалізацію воронки продажів та централізоване зберігання історії взаємодій [1, 2]. Використання сучасного технологічного стеку Laravel, Vue.js та Inertia.js дозволило створити продуктивний вебзастосунок із високою швидкістю роботи та можливістю подальшого масштабування [5-7]. Отримані результати можуть бути використані для цифровізації процесів продажів підприємств малого та середнього бізнесу [3].

Список літератури:

1. CRM-системи сприяють централізації клієнтської інформації та підвищенню ефективності продажів (Purnama & Manalu, 2024; Boppana, 2024).

2. Автоматизація бізнес-процесів дозволяє підвищити продуктивність та якість обслуговування клієнтів (Purnama & Manalu, 2024).

3. Цифрова трансформація є важливим чинником розвитку малого та середнього бізнесу (Hoang, 2024; Sagala & Őri, 2024).

4. Pashchuk I., Turchenko I., Dombrovskyi M., Hrushytskyi M. AI-Driven Natural Language Processing to SQL Query Generation for Enhanced Human-Machine Interaction in the Digital Era. IEEE IDAACS, 2025.

5. Laravel Documentation. URL: https://laravel.com/docs.
6. Vue.js Documentation. URL: https://vuejs.org/.
7. Inertia.js Documentation. URL: https://inertiajs.com/.
8. PostgreSQL Documentation. URL: https://www.postgresql.org/docs/.
9. Roadrunner: a high-performance PHP application server. URL: https://roadrunner.dev/.