ДЕРЖАВНИЙ УНІВЕРСИТЕТ

ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ

ІНФОРМАЦЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «CRM система мовами PHP та JavaScript засобами Laravel та Vue.js»

на здобуття освітнього ступеня бакалавра

зі спеціальності 122 Комп'ютерні науки

(код, найменування спеціальності)

освітньо-професійної програми Комп'ютерні науки

(назва)

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

_____ Андрій ЯРЕМЕНКО

(підпис) *(Ім'я, ПРІЗВИЩЕ здобувача)*

Виконав:

здобувач вищої освіти Андрій ЯРЕМЕНКО

група КНД-43

Керівник: Віктор ВИШНІВСЬКИЙ

(науковий ступінь, вчене звання) д.т.н., професор

Рецензент: _____

(Ім'я, ПРІЗВИЩЕ)

(науковий ступінь, вчене звання)

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ

ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Комп'ютерних наук

Ступінь вищої освіти Бакалавр

Спеціальність Комп'ютерні науки

Освітньо-професійна програма Комп'ютерні науки

ЗАТВЕРДЖУЮ

Завідувач кафедри Комп'ютерних наук

_____ В. В. Вишнівський

«____» _____2025 р.

ЗАВДАННЯ

НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

Яременко Андрій Олексійович

(прізвище, ім'я, по батькові)

1. Тема роботи: «CRM система мовами PHP та JavaScript засобами Laravel та Vue.js»

керівник кваліфікаційної роботи

Віктор ВИШНІВСЬКИЙ д.т.н., професор.

(ім'я, ПРИЗВІЩЕ, науковий ступінь, вчене звання)

Затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025р. № 56.

2. Строк подання студентом роботи «30» травня 2025 року

3. Вихідні дані до роботи:

Теоретико-методологічні засади побудови CRM-систем;

Аналіз предметної області й постановка задачі розробки CRM;

Проєктування та реалізація CRM-системи засобами Laravel і Vue.js.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

4.1 Вступне пояснення теми та актуальності дослідження.
4.2 Вимоги до сучасної CRM.
4.3 Розробка CRM.
4.4 Узагальнення висновків.

5. Перелік таблиць, графічного матеріалу

5.1 Презентаця PowerPoint.

6. Дата видач завдання «25» лютого 2025р

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Аналіз існуючих CRM, формування вимог	03.03.2025 р.	виконано
2	Формулювання цілей, завдань дослідження та визначення методів реалізації	09.03.2025 р.	виконано
3	Проєктування архітектури додатку	18.03.2025 р.	виконано
4	Реалізація та тестування	27.03.2025 р.	виконано
5	Написання пояснювальної записки, оформлення, підготовка презентації	22.04.2025 р.	виконано
6	Завершення оформлення дипломної роботи відповідно до вимог	14.05.2025 р.	виконано
7	Підготовка доповіді та метеріалу до захисту	19.05.2025 р.	виконано

Здобувач вищої освіти _____ Андрій ЯРЕМЕНКО
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник
кваліфікаційної роботи _____ Віктор ВИШНІВСЬКИЙ
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 43 стор., 3 табл., 17 рис., 20 джерел.

Мета роботи – розробка прототипу сучасної CRM-системи на базі відкритих технологій Laravel (PHP 8.2) і Vue.js 3, що інтегрує ключові функціональні блоки для ефективного управління клієнтською базою, угодами, завданнями, подіями та аналітикою.

Об'єкт дослідження – сучасні CRM-системи як інтеграційні платформи для управління взаємовідносинами з клієнтами у бізнес-процесах.

Предмет дослідження – архітектура, технологічний стек і методи реалізації веб-орієнтованої CRM-платформи із застосуванням Laravel і Vue.js.

Короткий зміст роботи:
Робота охоплює аналіз еволюції CRM-систем, формулювання вимог користувачів, вибір технологій, моделювання даних, розробку бекенд- та фронтенд-логіки, а також впровадження механізмів безпеки і масштабованості. Розроблений прототип демонструє сучасний SPA-інтерфейс, захищене API, контейнеризацію та інтеграцію з інструментами розробки, що відповідає сучасним стандартам і відкриває перспективи подальшого розвитку системи.

КЛЮЧОВІ СЛОВА: CRM, LARAVEL, VUE.JS, КЛІЄНТСЬКІ ДАНІ, БІЗНЕС-ПРОЦЕСИ, ВЕБ-РОЗРОБКА, API, SPA, PHP, JAVASCRIPT, DOCKER, MYSQL.

ЗМІСТ

ВСТУП

Сучасні бізнес-процеси дедалі більше орієнтуються на ефективну взаємодію з клієнтами як стратегічну цінність, що формує конкурентну перевагу компанії. У цих умовах CRM-системи (Customer Relationship Management) перестають бути просто інструментами обліку — вони перетворюються на інтеграційні платформи, що об'єднують дані, аналітику, автоматизацію та комунікацію в єдиний інформаційний простір. В умовах цифрової трансформації саме CRM є тим ядром, навколо якого будуються маркетингові кампанії, організовуються процеси продажів, здійснюється підтримка клієнтів та приймаються обґрунтовані управлінські рішення.

Метою даної дипломної роботи є розробка прототипу сучасної CRM-системи на базі відкритих технологій Laravel (PHP 8.2) і Vue.js 3, яка продемонструє здатність інтегрувати ключові функціональні блоки — від ведення клієнтської бази до управління угодами, задачами, подіями та аналітичними панелями. Вибір саме цього технологічного стека зумовлений його популярністю, зрілістю, активною спільнотою підтримки, а також можливістю досягнути високого рівня безпеки, продуктивності та масштабованості навіть у рамках навчального проєкту.

У вступі доцільно окреслити не лише загальну актуальність тематики, але й виклики, які постають перед розробниками CRM-рішень у 2020-х роках: зростання обсягів даних, вимоги до мобільності, необхідність захисту персональної інформації відповідно до GDPR, очікування клієнтів щодо омніканального обслуговування та персоналізованої взаємодії. З огляду на ці фактори, підхід до проєктування системи має відповідати сучасним стандартам: модульна архітектура, API-орієнтованість, контейнеризація, дотримання принципів DevOps та CI/CD.

Структурно дипломна робота поділена на кілька частин: теоретико-методологічну (де проаналізовано еволюцію концепції CRM, класифікацію

функціональних моделей, порівняння технологічних платформ і патернів архітектури), аналітичну (де окреслено вимоги користувачів, обґрунтовано вибір

технологій, побудовано модель даних), інженерну (де описано логіку реалізації системи на рівні бекенду і фронтенду), а також розділ, присвячений питанням безпеки, масштабованості й версіонування проєкту.

Результатом є побудова прототипу CRM-платформи, який, хоча і має обмежений функціонал у порівнянні з комерційними аналогами, проте демонструє усі ключові компоненти сучасної веб-архітектури: інтерактивний SPA-інтерфейс, захищене API, централізоване управління станом, контейнерну інфраструктуру та дотримання принципів безпеки.

1 ТЕОРЕТИКО-МЕТОДОЛОГІЧНІ ЗАСАДИ ПОБУДОВИ CRM-СИСТЕМ

1.1 Еволюція концепції Customer Relationship Management та її місце у цифровій трансформації бізнес-процесів

У сучасному бізнес-середовищі, що характеризується високою конкуренцією, стрімкими технологічними змінами та зростаючими очікуваннями клієнтів, взаємовідносини з ними стали критично важливим елементом стратегічного управління. Концепція Customer Relationship Management (CRM) набула особливої актуальності як інструмент не лише для організації продажів, а й як стратегічна парадигма управління цінністю клієнта протягом усього життєвого циклу. З розвитком цифрових технологій CRM трансформувалась від простих картотек до складних інтелектуальних платформ, що підтримують цифрові бізнес-процеси.

Історично CRM пройшла кілька етапів. Перший (1980–1990-ті) відзначався використанням баз даних для зберігання контактів та систем Sales Force Automation (SFA), які автоматизували частину процесів продажу. Вони були ізольованими, з обмеженою функціональністю і без інтелектуальних можливостей.

Другий етап (кінець 1990-х — початок 2000-х) пов’язаний з інтеграцією CRM із ERP-системами, розширенням функціоналу аналітики, маркетингу та підтримки клієнтів. CRM почала охоплювати кілька підрозділів і набула системного характеру.

Наступним став етап Web-орієнтованих CRM, що співпав із розвитком інтернету, мобільних та хмарних технологій. Це дозволило компаніям знизити витрати на інфраструктуру та забезпечити доступ до даних цілодобово з будь-якої точки. Зросла інтеграція із соцмережами, месенджерами, e-commerce, що дало початок концепції Social CRM, орієнтованої на багатоканальну взаємодію.

Сучасна CRM є частиною цифрової трансформації бізнесу — процесу переосмислення бізнес-моделей та клієнтського досвіду з допомогою цифрових технологій. CRM стала центральним елементом цифрового підприємства, акумулюючи дані про клієнтів, їх поведінку, історію покупок і рівень задоволеності.

Сучасні CRM-системи виконують роль «нервової системи» організації, забезпечуючи цілісність інформації, автоматизацію, інтеграцію з AI і машинним навчанням. Вони допомагають прогнозувати попит, адаптувати пропозиції та будувати персоналізовані сценарії взаємодії. Інструменти Big Data, AI, RPA, IoT формують нове покоління — розумні CRM.

Особливої ваги набуває концепція омніканального клієнтського досвіду (omnichannel CX), яка стала можливою завдяки CRM-системам. Компанії забезпечують єдину взаємодію з клієнтами через різні канали — сайт, мобільний застосунок, месенджери, соціальні мережі, електронну пошту, кол-центр — при цьому зберігаючи контекст і історію кожної взаємодії. Такий рівень узгодженості та персоналізації неможливо досягти без CRM як єдиного джерела клієнтської інформації.

Крім того, CRM-системи стали не лише інструментом для менеджерів із продажу, а й джерелом аналітики для вищого керівництва, платформою для взаємодії з партнерами, інструментом для автоматизації маркетингу (Marketing Automation), каналом зворотного зв’язку, інструментом рекрутингу та навіть джерелом ідей для інновацій. Усе це свідчить про те, що сучасна CRM — це не просто «база контактів», а інтегрована бізнес-платформа.

Таким чином, еволюція CRM (таблиця 1.1) — це не лінійний процес розширення функціоналу, а глибинна трансформація способів ведення бізнесу. CRM стала не просто допоміжною технологією, а основою цифрової стратегії підприємства, яка дозволяє створювати довготривалі, персоналізовані, вигідні

взаємовідносини з клієнтами, забезпечуючи сталий розвиток бізнесу в умовах цифрової економіки.

Таблиця 1.1 — Еволюція концепції CRM

Етап розвитку	**Характеристика**	**Технології**	**Основні функції**	**Роль у бізнесі**
CRM 1.0 (1980–1995)	Картотеки, SFA	Локальні бази даних, MS Access	Збереження контактів, облік продажів	Підтримка продажів
CRM 2.0 (1996–2005)	Корпоративні CRM	Client-Server, ERP інтеграція	Продажі, маркетинг, підтримка клієнтів	Централізація взаємодії
CRM 3.0 (2006–2012)	Web-орієнтовані CRM	Інтернет, SaaS, інтеграція з email	Веб-доступ, звіти, базова аналітика	Підвищення мобільності та гнучкості
CRM 4.0 (2013–2020)	Social CRM	Хмари, соціальні мережі, мобільні додатки	Омніканальність, персоналізація	Формування клієнтоорієнтованої стратегії
CRM 5.0 (2020–дотепер)	Інтелектуальна CRM (iCRM)	AI, ML, Big Data, IoT, чат-боти	Прогнозування, автоматизація, рекомендації	Ядро цифрової трансформації бізнесу

1.2 Функціональні моделі CRM: оперативна, аналітична й колаборативна складові

У сучасних умовах високої конкуренції та насиченості ринку якісна взаємодія з клієнтами є критичним чинником успішності бізнесу. Саме тому все більше компаній впроваджують CRM-системи (Customer Relationship Management) як інструмент для систематизації, автоматизації й оптимізації процесів, пов’язаних із управлінням взаємодією з клієнтами. Залежно від фокусу функціоналу CRM-системи поділяються на три основні функціональні моделі: оперативну, аналітичну та колаборативну. Кожна з них виконує окремі завдання, проте в комплексі забезпечують цілісний підхід до побудови довготривалих і продуктивних відносин з клієнтами.

- Оперативна CRM (Operational CRM) — це підсистема, яка зосереджена на автоматизації ключових бізнес-процесів взаємодії з клієнтом. До неї входять модулі управління продажами, маркетингом і сервісом. В рамках оперативної CRM відбувається фіксація контактів з клієнтами, формування лідів, ведення угод, управління кампаніями, контроль дзвінків, листування, обробка звернень та запитів. Основне завдання оперативної CRM — забезпечити централізоване збереження інформації про клієнтів і зробити взаємодію з ними максимально ефективною та персоналізованою. Такі системи часто мають інтеграції з телефонією, електронною поштою, соціальними мережами та месенджерами. Оперативна CRM є основою для щоденної роботи відділів продажу, маркетингу та підтримки.
- Аналітична CRM (Analytical CRM) — складова, орієнтована на обробку, аналіз та інтерпретацію даних про клієнтів з метою прийняття обґрунтованих управлінських рішень. Вона дозволяє компанії глибше зрозуміти поведінку клієнтів, їх потреби, рівень лояльності, життєвий цикл, а також ефективність різних каналів комунікації та маркетингових кампаній. До аналітичної CRM входять модулі побудови звітів, візуалізації даних, сегментації клієнтської

бази, прогнозування продажів та оцінки рентабельності. На її основі реалізуються такі концепції, як Customer Lifetime Value (CLV), RFM-аналіз, кластеризація клієнтів тощо. Завдяки аналітичній складовій CRM-компанія може приймати стратегічні рішення на основі фактів, а не інтуїції, що особливо важливо в умовах високої вартості залучення нового клієнта.

- Колаборативна CRM (Collaborative CRM) — модель, що забезпечує координацію й ефективну комунікацію між різними підрозділами організації, а також з партнерами і зовнішніми учасниками, залученими до процесу обслуговування клієнта. Вона сприяє узгодженості дій між маркетингом, продажами, логістикою, технічною підтримкою тощо. Основне завдання — зробити взаємодію з клієнтом єдиною, без розривів, дублювання або втрат інформації. Колаборативна CRM охоплює такі функції, як обмін контактами й нотатками між відділами, спільне управління запитами, погодження дій, узгодження графіків зустрічей, спільне ведення клієнтських кейсів. В сучасних умовах особливої актуальності набуває підтримка омніканальних комунікацій — тобто єдина взаємодія з клієнтом незалежно від того, чи він звернувся через сайт, телефон, пошту, чат-бот або соціальну мережу.

Взаємозв’язок між цими трьома складовими є критично важливим для ефективного функціонування CRM-системи як комплексного інструменту управління клієнтською взаємодією. Оперативна CRM збирає первинні дані про взаємодію з клієнтами, які потім аналітична CRM обробляє й перетворює на знання, а колаборативна CRM забезпечує належне використання цих знань у міжвідділовій роботі. Таким чином, CRM-система перетворюється на цілісне середовище, що не лише обслуговує клієнтів, але й навчається від кожної взаємодії з ними, постійно підвищуючи якість сервісу та персоналізації (Рисунок 1.1).

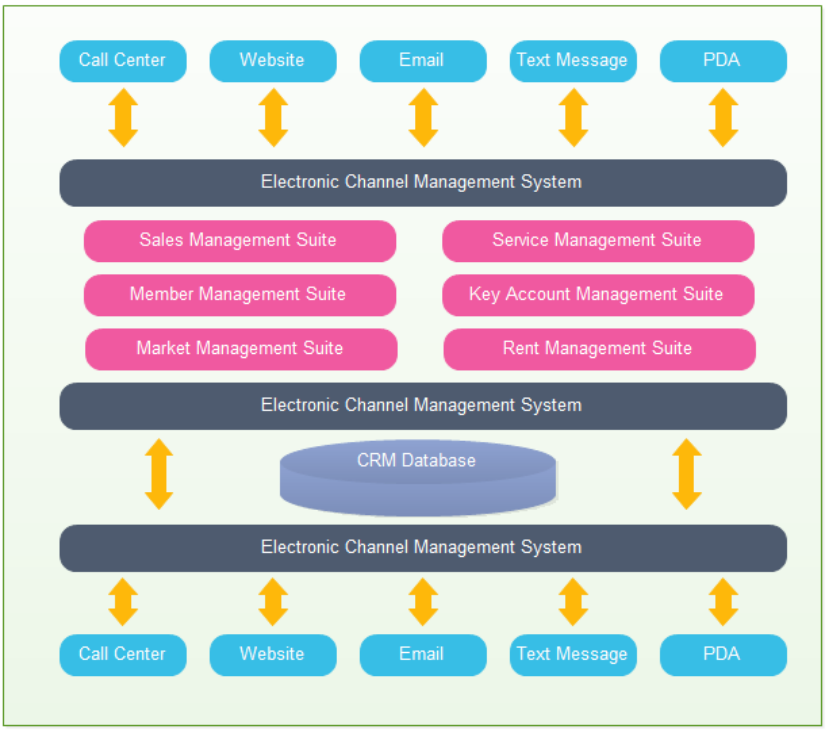


Рисунок 1.1 — Архітектура CRM

У практичному сенсі, більшість сучасних CRM-рішень прагнуть поєднати в собі всі три функціональні моделі, створюючи єдину платформу для взаємодії з клієнтами. Наприклад, системи на зразок Salesforce, Zoho CRM або Microsoft Dynamics CRM мають модулі, які відповідають усім трьом підходам. У випадку розробки власної CRM-системи, наприклад, на базі стеку Laravel + Vue.js, доцільно спочатку реалізувати оперативну складову, поступово додаючи аналітичні та колаборативні функції відповідно до потреб бізнесу та готовності інфраструктури.

Отже, функціональні моделі CRM-системи утворюють триєдину основу ефективного управління взаємодією з клієнтами. Їх гармонійне поєднання дозволяє не лише автоматизувати процеси, але й перетворити взаємодію з клієнтом у джерело стратегічної переваги та довгострокової лояльності.

1.3 Порівняльний огляд технологічних платформ PHP-екосистеми та JavaScript-стеку для розробки корпоративних інформаційних систем

У сучасній практиці створення корпоративних інформаційних систем (КІС), зокрема CRM, ERP, HRM-рішень або внутрішніх порталів підприємств, особливу увагу приділяють вибору технологічної платформи. Від цього вибору залежить не лише технічна ефективність програмного продукту, а й вартість розробки, можливість масштабування, зручність підтримки, продуктивність, рівень безпеки та навіть подальший розвиток. Серед найбільш популярних технологій у сфері веб-розробки домінують дві екосистеми — PHP (зокрема з фреймворками Laravel, Symfony) та JavaScript-стек (із використанням Node.js, Express на сервері та таких фронтенд-фреймворків як Vue.js, React, Angular).

PHP — це мова з багаторічною історією, яка починалась як скриптова мова для динамічних веб-сторінок, але за останні роки значно еволюціонувала. Завдяки таким фреймворкам як Laravel, вона стала сучасним інструментом для побудови повноцінних систем із чіткою архітектурою, підтримкою REST API, шаблонізацією, ORM, обробкою подій, чергами, а також високим рівнем автоматизації завдяки інтегрованим інструментам (наприклад, Artisan CLI). Laravel надає зручні засоби для розробки backend-сервісів, реалізації складної бізнес-логіки та швидкого прототипування корпоративних рішень.

З іншого боку, JavaScript, який традиційно використовувався лише для фронтенду, після появи Node.js став універсальною мовою для повного циклу розробки (full stack). Розробники можуть використовувати єдину мову як для клієнтської, так і для серверної частини. Це спрощує обмін даними, структуру проєкту, організацію командної роботи та дозволяє ефективно будувати сучасні SPA (Single Page Applications), які широко застосовуються у корпоративних веб-застосунках. За допомогою фреймворків Express.js, NestJS та базових можливостей Node.js створюються продуктивні та масштабовані серверні рішення, які зручно поєднуються з фронтендом на Vue.js або React.

Обидві платформи мають свої переваги та слабкі сторони. PHP орієнтований насамперед на класичну архітектуру MVC, має чудову підтримку реляційних баз даних, велику спільноту та десятки тисяч доступних пакетів. JavaScript, натомість, забезпечує високу інтерактивність та реальну двосторонню взаємодію з клієнтом у режимі реального часу, завдяки WebSocket, EventEmitter, реактивності та іншим механізмам.

Оскільки корпоративні інформаційні системи потребують високої продуктивності, гнучкої взаємодії з користувачем, захищеного доступу до даних та можливості подальшої інтеграції, важливо проаналізувати ці дві платформи з точки зору низки ключових критеріїв. Нижче наведено порівняння (Таблиця 1.2) основних характеристик PHP (на прикладі Laravel) та JavaScript (на прикладі Node.js + Vue.js/React):

Таблиця 1.2 — Порівняння платформ

Критерій	**PHP (Laravel)**	**JavaScript (Node.js + Vue/React)**
Архітектурна модель	MVC, REST, Modular	SPA, MVVM, REST, Event-driven
Повноцінний Full Stack	Частково	Так
Продуктивність при великій кількості з’єднань	Середня	Висока
Крива навчання	Низька — легкий старт	Середня — потребує розуміння реактивності та асинхронності
Підтримка WebSockets	Через сторонні пакети або Laravel Echo	Вбудовано через socket.io, ws

Продовження таблиці 1.2

Критерій	**PHP (Laravel)**	**JavaScript (Node.js + Vue/React)**
Масштабованість	Добре для монолітів або мікросервісів	Відмінно для мікросервісів і real-time застосунків
Актуальність у корпоративних проєктах	Дуже популярний у CRM, ERP, B2B-рішеннях	Зростає, особливо для інтерактивних дашбордів і SPA
Зрілість технологій	Висока	Висока для фронтенду, середня для бекенду
Спільнота та ресурси	Дуже активна, велика база знань	Дуже активна, особливо у фронтенді
Стилі розробки	Чітка серверна логіка, Blade шаблони, REST	Компонентний підхід, SPA, реактивний інтерфейс

Отже, як видно з таблиці, Laravel на PHP демонструє сильні позиції в розробці надійних серверних систем із чіткою логікою, де фронтенд є або класичним (на Blade), або окремим SPA-клієнтом. Це чудовий вибір для побудови стійких монолітів, інтеграції з базами даних, створення адміністративних панелей, API-сервісів. При цьому фронтенд можна поступово модернізувати за допомогою Vue.js або React, підключаючи їх лише до окремих компонентів.

У свою чергу, JavaScript-стек демонструє надзвичайну гнучкість і зручність у побудові реактивних систем, які повинні працювати в режимі реального часу, або забезпечувати багатий користувацький досвід. SPA-додатки на основі Vue.js або React, підключені до backend-API на Node.js, дозволяють створювати інтерактивні CRM із гнучкими інтерфейсами, багатомовністю, drag-and-drop, inline-

редагуванням, повідомленнями в реальному часі тощо. Node.js є особливо доцільним для побудови масштабованих мікросервісів і real-time сервісів на кшталт чату або push-нотифікацій.

На практиці часто використовуються гібридні підходи, коли Laravel виступає бекендом із потужною бізнес-логікою, а Vue.js забезпечує SPA-фронтенд. Така інтеграція дозволяє поєднати надійність і стабільність PHP з гнучкістю та інтерфейсною свободою JavaScript.

У підсумку, обидві платформи є конкурентоспроможними, і вибір між ними має базуватись на вимогах конкретного проєкту, кваліфікації команди та довгострокових цілях компанії. Laravel більше підходить для проектів зі складною серверною логікою, тоді як JavaScript-стек — для побудови гнучких, взаємодіючих у реальному часі, високодинамічних інтерфейсів. У випадку CRM-системи, найбільш збалансованим підходом є саме поєднання обох екосистем у вигляді Laravel API + Vue.js SPA.

1.4 Архітектурні патерни MVC і MVVM у веб-додатках: теоретичні основи та доцільність використання в Laravel і Vue.js

У сучасному світі веб-технологій вибір архітектурного патерну має вирішальне значення для структури, логіки та ефективності розробки програмного забезпечення. Залежно від характеру додатку, його цільової аудиторії, функціональних вимог та масштабу, архітектурне рішення може як спростити, так і ускладнити подальшу розробку, тестування, підтримку й масштабування системи. Особливої актуальності ця тема набуває у контексті побудови комплексних систем, таких як CRM, які вимагають чіткого розділення відповідальностей, взаємодії між численними модулями та забезпечення зручного інтерфейсу для кінцевого користувача.

Серед найбільш поширених архітектурних патернів у веб-розробці виділяють MVC (Model-View-Controller), MVVM (Model-View-ViewModel) та SPA (Single Page Application). У цьому розділі розглянемо сутність кожного з них, порівняємо їх переваги й недоліки, а також проаналізуємо доцільність їх застосування у рамках використання фреймворку Laravel на бекенді та бібліотеки Vue.js на фронтенді.

Model-View-Controller (MVC) — це один з найстаріших і найпоширеніших патернів проєктування, що вперше був запропонований у 1970-х роках у середовищі Smalltalk. Його ключова ідея полягає в розділенні структури додатку на три взаємопов’язані, але незалежні компоненти:

- Model (Модель) — описує бізнес-логіку застосунку, обробляє дані, здійснює доступ до бази даних і підтримує внутрішній стан системи. У контексті Laravel ця роль реалізується за допомогою ORM Eloquent, що надає зручні засоби для взаємодії з реляційними базами даних.
- View (Представлення) — відповідає за відображення інформації користувачу. У Laravel це реалізується за допомогою шаблонізатора Blade, який дозволяє створювати динамічні HTML-сторінки з використанням серверних змінних і логіки відображення.
- Controller (Контролер) — приймає запити користувача, викликає відповідні методи моделі, а також обирає, яке представлення слід показати у відповідь.

Такий поділ сприяє модульності, полегшує тестування і підтримку коду, дає змогу паралельної роботи різних команд розробників над окремими компонентами.

У фреймворку Laravel архітектура MVC є фундаментом. Вона реалізована за допомогою чіткої системи маршрутизації (Routes), контролерів, моделей (Eloquent) та шаблонів (Blade). Контролери в Laravel зазвичай відповідають за обробку HTTP-запитів, виклик методів моделей і повернення відповідних подань або JSON-відповідей для API.

MVVM (Model-View-ViewModel) — це еволюційний розвиток MVC, що був запропонований Майкрософт для використання у середовищі WPF (Windows Presentation Foundation). Основна ідея полягає у подальшому розділенні логіки представлення (інтерфейсу користувача) від бізнес-логіки, а також в інтеграції механізмів двостороннього зв’язку між даними і представленням.

Компоненти MVVM:

- Model (Модель) — так само, як і в MVC, відповідає за бізнес-логіку, зберігання та обробку даних.
- View (Представлення) — інтерфейс, з яким взаємодіє користувач.
- ViewModel (Модель представлення) — посередник між Model і View, який надає дані в тій формі, яка зручна для відображення, та забезпечує реактивність — автоматичне оновлення інтерфейсу при зміні стану даних.

У контексті фронтенд-розробки, Vue.js реалізує ідеї MVVM через реактивні змінні (рефактори, computed, watchers), двосторонній зв’язок (v-model), та компоненти, які в собі одночасно поєднують View і ViewModel. Це дозволяє створювати високореактивні інтерфейси з мінімальним ручним втручанням у DOM.

SPA (Single Page Application) — це концепція побудови веб-додатків, при якій уся взаємодія з користувачем здійснюється в межах однієї HTML-сторінки. На відміну від класичних багатосторінкових додатків (MPA — Multi Page Applications), де кожен перехід на іншу сторінку супроводжується повним перезавантаженням, SPA завантажує сторінку лише один раз, а подальші зміни вмісту відбуваються динамічно, за допомогою JavaScript та запитів до API.

Ключові характеристики SPA:

- Інтерактивність: зменшення затримок між діями користувача та відповіддю системи.

- Менше навантаження на сервер: передається тільки дані, а не вся сторінка.
- Використання REST або GraphQL API: для взаємодії з бекендом.

У бібліотеці Vue.js реалізація SPA здійснюється через бібліотеку маршрутизації Vue Router, централізоване управління станом за допомогою Vuex або Pinia, а також модульну структуру компонентів. Завдяки цьому можливо створювати масштабовані інтерфейси, які працюють як нативні додатки.

Поєднання Laravel і Vue.js дозволяє використовувати переваги всіх трьох архітектурних підходів:

- На стороні сервера Laravel реалізує класичну архітектуру MVC, що забезпечує чисту логіку обробки запитів, модульність, організованість коду і ефективну роботу з базами даних.
- На стороні клієнта Vue.js реалізує ідеї MVVM, що забезпечує реактивність інтерфейсу, зручний двосторонній зв’язок між даними й формами, та високий рівень ізоляції бізнес-логіки інтерфейсу.
- У взаємодії між фронтендом і бекендом використовується концепція SPA, де Vue.js відповідає за взаємодію з користувачем, а Laravel — за обробку API-запитів.

Переваги такого підходу:

1. Розділення відповідальностей: бекенд і фронтенд функціонують як окремі сервіси, що дозволяє команді розробників працювати паралельно.
2. Гнучкість і масштабованість: можна легко змінити або замінити одну частину системи (наприклад, переписати фронтенд з Vue на інший фреймворк) без істотних змін в іншій.
3. Продуктивність: SPA забезпечує швидке реагування інтерфейсу без потреби перезавантаження сторінки, що позитивно впливає на UX.

4. Повторне використання компонентів: Vue дозволяє створювати багаторазові компоненти, які можна легко інтегрувати в різні частини додатку.
5. Покращене тестування: через чітку архітектурну структуру тестування кожного шару (моделі, контролери, API, компоненти інтерфейсу) стає простішим і ефективнішим.

У сучасній розробці складних веб-додатків, таких як CRM-системи, поєднання архітектурних підходів MVC, MVVM та SPA забезпечує потужну основу для створення структурованого, модульного, зручного у підтримці програмного забезпечення. Вибір стеку Laravel + Vue.js дозволяє повністю реалізувати переваги цих патернів: з одного боку, Laravel забезпечує надійність і безпеку бекенду, з іншого — Vue.js дарує користувачам сучасний інтерфейс, зручний та швидкий у використанні.

Таким чином, архітектурне поєднання MVC + MVVM + SPA не лише теоретично обґрунтоване, але й практично доцільне, особливо в контексті розробки високоякісних веб-застосунків, орієнтованих на взаємодію з користувачами, обробку великої кількості даних та гнучкість у розширенні функціоналу.

2 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ Й ПОСТАНОВКА ЗАДАЧІ РОЗРОБКИ CRM

2.1 Вимоги користувачів і стейкголдерів до майбутньої системи

Розробка CRM-системи починається з глибокого розуміння того, хто саме буде користуватися цією системою та які завдання вона має вирішувати. Оскільки цей проєкт створюється в рамках дипломної роботи і не орієнтується на комерційне використання, його цільовою аудиторією виступають уявні користувачі — співробітники підприємств середнього рівня, які займаються управлінням взаємовідносин з клієнтами. Ці користувачі мають загальні потреби, характерні для багатьох компаній, а також вимоги, які сформовані на основі аналізу популярних CRM-систем, які вже використовуються на ринку. Такий підхід дозволяє закласти фундамент, який у майбутньому може бути адаптований під конкретні бізнес-процеси.

Майбутні користувачі цієї системи — це передусім менеджери з продажу, які щоденно взаємодіють із клієнтами, ведуть угоди, контролюють виконання завдань і планують робочі дії. Вони потребують інструментів, що спрощують і систематизують їхню роботу, надають можливість швидко знаходити інформацію про клієнтів, зручно вести історію комунікацій та відслідковувати статуси угод. Для них важливі простий і зрозумілий інтерфейс, доступність ключових функцій у кілька кліків, а також можливість автоматизувати рутинні операції, щоб підвищити продуктивність і знизити ймовірність помилок.

Крім менеджерів, суттєву роль відіграють керівники відділів, яким необхідні засоби для аналітики і контролю роботи своїх підлеглих. Вони зацікавлені у звітах, що демонструють загальні показники продажів, кількість активних і завершених угод, середній час роботи з клієнтом, ефективність менеджерів, а також у можливості швидко отримувати узагальнені дані для прийняття управлінських

рішень. Тому CRM має передбачати механізми генерації різноманітних звітів і візуалізацій.

У свою чергу маркетологи, які також можуть бути користувачами системи, потребують інструментів для сегментації клієнтської бази, щоб планувати таргетовані рекламні кампанії та відслідковувати реакцію клієнтів на маркетингові активності. Для них важливо мати доступ до повної історії взаємодій з клієнтами, а також можливість створення груп клієнтів за різними критеріями.

Не менш важливою категорією є служба підтримки, яка використовує CRM для ефективного оброблення звернень і запитів клієнтів. Для цих користувачів критично необхідним є швидкий доступ до історії комунікацій, можливість реєструвати звернення, ставити завдання іншим співробітникам і контролювати їхнє виконання.

Ще одним важливим стейкголдером виступає адміністратор системи, який відповідає за налаштування CRM, управління ролями, доступами та забезпечення безпеки даних. Він потребує зручних інструментів для управління користувачами, контролю за безпекою та стабільністю роботи системи.

На основі аналізу популярних CRM-систем, таких як Salesforce, HubSpot, Zoho CRM, а також враховуючи реальні потреби користувачів, можна сформулювати загальні вимоги до майбутньої системи. Головною вимогою є простота і зручність інтерфейсу, що дозволить будь-якому користувачу швидко освоїтися з функціоналом, а також швидко і безпомилково виконувати свої робочі задачі. Система має забезпечувати централізоване зберігання інформації про клієнтів, угоди, завдання та події, дозволяючи легко знаходити потрібні дані за допомогою зручних фільтрів і пошуку.

Користувачі очікують, що CRM буде не просто сховищем даних, а повноцінним інструментом для управління процесом продажів, який дозволяє відслідковувати всі етапи угод, ставити завдання і нагадування, автоматизувати

рутинні операції. Це підвищує ефективність роботи і знижує кількість помилок, пов’язаних із забуванням чи пропуском важливих дій.

Для керівників і аналітиків важливо мати можливість отримувати звіти і аналітику, що відображають ключові показники ефективності роботи команди і дають змогу приймати обґрунтовані управлінські рішення. Також важливо, щоб система забезпечувала надійний контроль доступу, гарантувала безпеку персональних і комерційних даних, а користувачі бачили лише ту інформацію, яка їм дозволена відповідно до їхніх ролей.

Враховуючи сучасні тренди, майбутні користувачі очікують, що CRM буде доступною не лише з десктопних комп’ютерів, а й з мобільних пристроїв, що дасть можливість працювати в будь-який час і в будь-якому місці. Крім того, інтеграція з іншими сервісами, такими як поштові системи, телефонія, бухгалтерські і маркетингові платформи, розглядається як важлива складова для підвищення продуктивності і узгодженості бізнес-процесів.

Особливістю дипломної розробки є те, що система повинна бути достатньо простою для реалізації і розуміння, але при цьому мати базовий набір функцій, які демонструють розуміння ключових процесів управління взаємовідносинами з клієнтами. Вона має бути легкою в розгортанні, з чіткою документацією і зрозумілою для тестування, що дозволить успішно захистити дипломну роботу і одночасно продемонструвати всі основні принципи побудови CRM.

Таким чином, вимоги користувачів і стейкголдерів до майбутньої системи базуються на потребі отримати інструмент, який допоможе організувати, систематизувати і автоматизувати роботу з клієнтами, забезпечить контроль і аналітику, буде безпечним і зручним у використанні, і одночасно гнучким для подальшого розвитку.

2.2 Вибір технологічного стека: обґрунтування застосування Laravel (PHP 8.2) на сервері та Vue.js 3 з Vite на клієнті

Розробка сучасної CRM-системи передбачає комплексне вирішення ряду взаємопов'язаних задач: управління клієнтськими даними, забезпечення багатокористувацького доступу, реалізація складної логіки бізнес-процесів, побудова інтерактивного інтерфейсу, захист інформації, а також масштабованість і гнучкість архітектури. Вибір технологічного стека напряму впливає на здатність системи відповідати цим вимогам як на етапі впровадження, так і в процесі подальшого розвитку. У рамках даної дипломної роботи було прийнято рішення використовувати Laravel (PHP 8.2) для серверної частини та Vue.js 3 у поєднанні з Vite — для реалізації клієнтського інтерфейсу. Такий стек є збалансованим рішенням, яке дозволяє поєднати надійність та зрілість бекенд-платформи з динамічністю й реактивністю сучасного фронтенду.

PHP 8.2 — це черговий стабільний реліз популярної мови програмування, випущений 8 грудня 2022 року. Ця версія продовжує курс, взятий із випуском PHP 8.0, на суттєве покращення як у плані продуктивності, так і якості коду, збереження типізації, безпечності та розширюваності. PHP 8.2 не містить таких радикальних змін, як впровадження JIT-компіляції в PHP 8.0, проте вона суттєво доповнює і розвиває існуючі механізми, водночас демонструючи відмінні показники швидкодії. Хоча PHP 8.2 не включає масштабних змін на рівні віртуальної машини (як це було з JIT у PHP 8.0), він демонструє помітні покращення продуктивності порівняно з попередніми версіями. У таблиці 2.3 наведено порівняння продуктивності різних версій PHP.

Таблиця 2.3 — Порівняння продуктивності версій PHP

Застосунок	**PHP 7.4**	**PHP 8.0**	**PHP 8.1**	**PHP 8.2**
WordPress	100 req/s	145 req/s	150 req/s	156 req/s
Laravel	90 req/s	130 req/s	135 req/s	140 req/s

Продовження таблиці 2.3

Застосунок	**PHP 7.4**	**PHP 8.0**	**PHP 8.1**	**PHP 8.2**
Symfony	75 req/s	112 req/s	118 req/s	122 req/s

Laravel — це сучасний високорівневий вебфреймворк для мови програмування PHP, який створений для розробки вебзастосунків будь-якої складності, з акцентом на чисту архітектуру, безпеку, продуктивність і зручність у використанні. Фреймворк був вперше представлений у 2011 році і відтоді став одним із найпопулярніших серед PHP-розробників завдяки своїй елегантності, великій спільноті та постійній підтримці з боку розробника — компанії Laravel LLC. Його структура базується на архітектурному шаблоні MVC (Model–View–Controller), що забезпечує логічне розділення відповідальностей у коді, сприяє кращій підтримуваності та масштабованості застосунків. Однією з ключових переваг Laravel є його багатофункціональне ядро та наявність великої кількості вбудованих можливостей «з коробки». Це, зокрема, система маршрутизації, механізми автентифікації та авторизації, інструменти для роботи з базами даних (ORM Eloquent), підтримка черг, подій, кешування, тестування, обробки запитів і шаблонізації. Завдяки цим можливостям Laravel значно пришвидшує процес розробки, зменшує кількість повторюваного коду і дозволяє реалізовувати складні бізнес-логіки з меншими зусиллями. Крім того, Laravel підтримує REST-архітектуру, що є критично важливою для побудови API-орієнтованих клієнт-серверних застосунків, які взаємодіють із фронтендом або мобільними клієнтами. Завдяки зручному механізму розширення через пакети та модулі Laravel легко адаптується під специфічні потреби проєкту. Він має офіційні додатки, такі як Laravel Jetstream, Breeze, Fortify, Horizon, Nova, Scout та інші, які спрощують реалізацію автентифікації, управління чергами, адміністративних панелей і

пошуку. Laravel також підтримує інтеграцію з популярними хмарними сервісами, сторонніми API, системами електронної пошти та платіжними шлюзами.

Vue.js — це прогресивний JavaScript-фреймворк для створення інтерфейсів користувача, який надає розробникам гнучкий та сучасний інструментарій для побудови масштабованих, динамічних та інтерактивних вебзастосунків. Розроблений у 2014 році Еваном Ю, фреймворк швидко набув широкої популярності завдяки своїй простоті, легкості у вивченні та розширюваності. У дипломному проєкті використовується **Vue.js третьої версії**, яка представляє собою значне технічне оновлення: вона заснована на принципах реактивності нового покоління та надає розробникам потужний Composition API, що забезпечує більшу гнучкість при створенні складних компонентів. Vue.js дозволяє будувати інтерфейси за допомогою компонентного підходу, де кожен елемент сторінки є самодостатнім модулем зі своїм шаблоном, логікою та стилями. Це сприяє повторному використанню коду, полегшує тестування та підтримку системи. У межах даного проєкту фреймворк використовується для створення клієнтської частини, яка динамічно взаємодіє з API, отримуючи та відображаючи дані, а також керуючи станом інтерфейсу у відповідь на дії користувача. Інтеграція з сервером відбувається через HTTP-запити, що дозволяє реалізувати повноцінну клієнт-серверну взаємодію в режимі реального часу. Третьою версією Vue було покращено систему реактивності, що позитивно впливає на продуктивність у великих проєктах. Водночас зберігається підтримка шаблонного синтаксису та можливість використовувати декларативний підхід до побудови інтерфейсів, що робить Vue зручним як для початківців, так і для досвідчених розробників. Особливістю фреймворку є його гнучка екосистема: Vue легко інтегрується з додатковими бібліотеками, такими як Vue Router для маршрутизації, Pinia або Vuex для глобального стану, а також з будь-якими сторонніми UI-компонентами.

Docker — це відкрита платформа для розробки, доставки та запуску програмного забезпечення в ізольованих середовищах, які називаються

контейнерами. У межах цього дипломного проєкту Docker використовується як інструмент для створення стабільного, незалежного від операційної системи середовища розробки та розгортання. Контейнеризація дозволяє уникнути проблем, пов’язаних із залежностями, сумісністю бібліотек чи відмінностями між локальним середовищем розробника та продуктивним сервером. Використання Docker у проєкті дало змогу об’єднати всі компоненти системи в окремі ізольовані контейнери: серверний застосунок на Laravel, фронтенд на Vue.js, базу даних MySQL, вебсервер Nginx. Кожен контейнер має власне оточення, конфігурацію, залежності та процеси, що забезпечує стабільність і передбачуваність роботи всієї системи незалежно від зовнішніх факторів. Усі контейнери координуються за допомогою інструменту Docker Compose, що дає змогу керувати всією інфраструктурою через єдиний конфігураційний файл. Це значно спрощує запуск проєкту як для розробників, так і в умовах продакшн-розгортання.

2.3 Модель даних CRM-системи: ER-діаграма, нормалізація та політики зберігання персональних даних

Розробка CRM-системи неможлива без чітко спроектованої моделі даних, яка забезпечує логічну структуру, ефективне зберігання, обробку та захист інформації. Модель даних виконує ключову роль у забезпеченні взаємозв’язків між сутностями, які фігурують у системі, та визначенні, яким чином ця інформація буде зберігатися, використовуватися й оброблятися.

На етапі проєктування бази даних було враховано як функціональні, так і нефункціональні вимоги до CRM-системи. Основною метою побудови моделі є забезпечення узгодженості даних, уникнення дублювання інформації, а також створення логічних зв’язків, які дадуть змогу ефективно витягувати, змінювати й аналізувати дані. Відповідно, модель повинна підтримувати гнучкість та масштабованість, враховувати можливі зміни бізнес-процесів і розвиток функціоналу.

Сутності CRM-системи (Рисунок 2.1) включають клієнтів, користувачів системи (менеджерів), угоди (ліди), пропозиції (оффери), сервіси. Кожна з цих сутностей виконує певну роль в управлінні відносинами з клієнтами. Наприклад, клієнт — це фізична або юридична особа, з якою встановлено зв’язок; угоди — це потенційні чи активні комерційні операції, що фіксують роботу менеджерів із залучення замовників; пропозиції — це спеціалізовані комерційні пропозиції товарів або послуг; сервіси — це зовнішні або внутрішні джерела трафіку), які інтегруються з CRM через API та мають унікальні API-ключі, що дозволяють автоматизовано створювати ліди в системі.

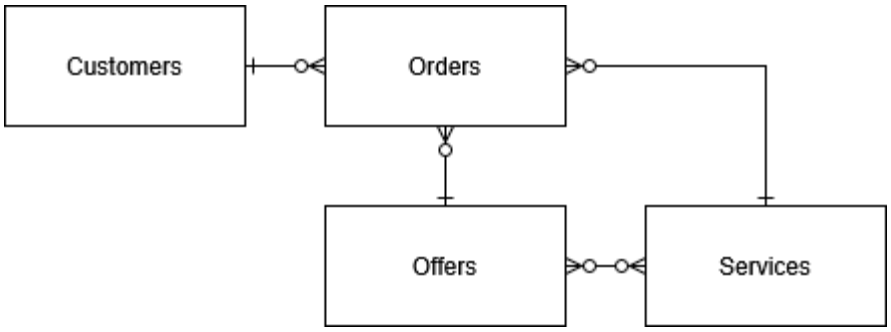


Рисунок 2.1 – ER-діаграма

Щоб уникнути аномалій під час оновлення та вилучення даних, база даних була приведена до третьої нормальної форми (3NF). На етапі нормалізації було виділено окремі таблиці для повторюваних атрибутів, усунено транзитивні залежності, а також забезпечено цілісність ключів. Наприклад, дані про компанії клієнтів зберігаються в окремій таблиці, яка пов’язується з таблицею клієнтів через зовнішній ключ. Це дозволяє зберігати корпоративних клієнтів в одному місці та ефективно оновлювати їхні дані без дублювань. Аналогічно, угоди зв’язані з конкретними клієнтами через зовнішній ключ, що дає змогу швидко отримувати інформацію про всі угоди, пов’язані з певним клієнтом.

При проєктуванні також були враховані вимоги до зберігання персональних даних згідно з Загальним регламентом захисту даних (GDPR) Європейського Союзу, а також українським законодавством — Законом України «Про захист

персональних даних». Усі поля, що містять персональні дані (наприклад, ім’я, прізвище, номер телефону, email, адреса), були чітко ідентифіковані та структуровані. В системі передбачено можливість деактивації або повного видалення персональних даних на вимогу суб’єкта даних, а також ведення журналу подій щодо обробки персональної інформації.

Дані передаються з використанням захищених протоколів (HTTPS), а в базі даних можуть зберігатися з додатковим шифруванням особливо чутливих атрибутів. Також було реалізовано механізми логічного розмежування доступу до інформації: менеджери мають доступ лише до тих клієнтів, з якими вони безпосередньо працюють. Це реалізується як на рівні додатку, так і через політики доступу на бекенді з використанням Laravel-політик. Такі політики забезпечують централізоване управління правами доступу, спрощують адміністрування та підвищують безпеку системи.

У CRM-системі передбачено механізм реєстрації згоди користувача на обробку його персональних даних, а також функціонал аудиту для моніторингу доступу до цих даних. Крім того, політики зберігання передбачають автоматичне видалення або анонімізацію даних, які не використовуються протягом певного періоду, наприклад, двох років, у відповідності до принципів "Data Minimization" та "Storage Limitation", що є основоположними в GDPR. Це гарантує, що система не зберігає зайвих даних і виконує вимоги щодо конфіденційності.

Таким чином, модель даних CRM-системи, спроєктована засобами Laravel та Vue.js, забезпечує не лише ефективну взаємодію між усіма компонентами, а й повну відповідність сучасним стандартам зберігання персональної інформації та структурну гнучкість для майбутнього масштабування. Вона підтримує як оперативні задачі, так і аналітичні запити, що робить систему зручною для щоденного використання та подальшого розвитку.

3 ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ CRM-СИСТЕМИ ЗАСОБАМИ LARAVEL І VUE.JS

3.1 Логічна та фізична архітектура рішення

Розробка CRM-системи потребує не лише реалізації функціональності відповідно до бізнес-вимог, але й ретельного проєктування логічної та фізичної архітектури, яка стане фундаментом для стабільної, масштабованої та зручної в обслуговуванні платформи. У цьому контексті обрана архітектура базується на сучасних підходах до розділення клієнтської та серверної частин, забезпечуючи ефективну взаємодію через RESTful API. Застосування моделі SPA (Single Page Application), в рамках якої уся взаємодія з сервером здійснюється асинхронно, а оновлення інтерфейсу відбувається без перезавантаження сторінки (Рисунок 3.1), дало змогу реалізувати динамічний, швидкий та інтерактивний користувацький інтерфейс. Центральною ідеєю побудови рішення стало прагнення до чіткої декомпозиції компонентів із максимальним дотриманням принципів модульності, повторного використання коду, а також відповідності загальноприйнятим стандартам розробки веб-застосунків.

Рисунок 3.1 – Життевий цикл SPA

Логічна архітектура системи охоплює кілька рівнів, кожен з яких виконує власну функцію і взаємодіє з іншими за допомогою чітко визначених інтерфейсів. На клієнтському рівні реалізовано повноцінний SPA-додаток за допомогою Vue.js. Його головним завданням є відображення даних, отриманих із серверної частини, обробка подій інтерфейсу, збереження локального стану, а також ініціація HTTP-запитів. Компонентний підхід у реалізації дозволив досягти високого ступеня повторного використання коду, а також забезпечити ізольованість функціональних блоків, що спрощує тестування та розширення інтерфейсу. Всі маршрути, якими користувач переміщується у межах SPA-додатку, налаштовуються через Vue Router. Це рішення дало змогу забезпечити плавну навігацію без перезавантаження сторінки та інтегрувати механізми захисту маршрутів відповідно до ролей користувачів.

Керування глобальним станом застосунку реалізовано за допомогою Vuex. Це дозволило централізовано зберігати критичні дані, такі як інформація про автентифікацію користувача, поточний сеанс, доступні модулі або права доступу. Завдяки Vuex стало можливим створити ефективну модель реактивного оновлення інтерфейсу, коли зміни в даних одразу відображаються у всіх відповідних компонентах, що значно підвищує зручність використання системи та зменшує кількість потенційних помилок, пов’язаних із неконсистентністю даних.

На стороні сервера архітектура побудована на фреймворку Laravel. Його використання дозволило структурувати логіку обробки запитів згідно з концепцією MVC. Контролери приймають запити з клієнтської частини, викликають відповідні сервіси або моделі, і повертають результати у вигляді JSON-відповідей. При проєктуванні серверної частини особлива увага приділялася розділенню відповідальностей: бізнес-логіка не розміщується безпосередньо в контролерах, а виноситься у сервіси, що сприяє підтримуваності й тестованості коду. Також у рамках API реалізовано чітку систему обробки помилок, валідації запитів, а також логування подій, що дозволяє швидко реагувати на проблеми при роботі системи.

Для захисту API використовується механізм Laravel Sanctum, який забезпечує безпечну роботу з токенами доступу в контексті SPA-додатків. Завдяки підтримці cookie-based аутентифікації, Sanctum дозволяє уникати складних конфігурацій, притаманних OAuth, водночас гарантуючи належний рівень безпеки. Усі запити з Vue-додатку супроводжуються необхідними заголовками й cookie, що дозволяє ідентифікувати користувача без потреби передавати токен у кожному запиті вручну. Крім цього, Sanctum підтримує механізм CSRF-захисту, що особливо важливо для SPA, де передача маркерів відбувається автоматично, і це суттєво спрощує реалізацію безпечного клієнта.

З технічного боку важливою особливістю є фізична реалізація компонентів через Docker-контейнери. Це дозволило уніфікувати середовище розробки, тестування та продакшну. У процесі проєктування архітектури передбачено розгортання окремих контейнерів для Laravel-застосунку, бази даних MySQL, сервера Nginx, а також середовища для збору і обслуговування фронтенду. Завдяки цьому забезпечується чітке розмежування між обов’язками кожного сервісу, спрощується масштабування, оновлення та обслуговування системи.

Інструмент Vite використовується для швидкої збірки фронтенду. Його головною перевагою є висока швидкість старту розробницького середовища та миттєве оновлення змін у коді під час розробки. У порівнянні з традиційними збирачами, такими як Webpack, Vite надає значно вищу продуктивність, особливо при роботі з великими проєктами, де кожна секунда має значення. Сама ж збірка фронтенду виконується через NPM, що керує залежностями, скриптами та пакетами, необхідними для запуску і розгортання Vue-додатку.

Всі сервіси комунікують між собою через внутрішню мережу Docker, що дозволяє уникати відкриття зовнішніх портів для служб, які не мають взаємодіяти з користувачем напряму. Це суттєво підвищує рівень безпеки та спрощує налаштування мережевої взаємодії. Зокрема, Nginx виступає як точка входу для всіх HTTP-запитів, передаючи їх відповідному сервісу – чи то Laravel, чи Vue – залежно

від маршруту та типу запиту. Завдяки цьому досягається централізоване керування потоками трафіку та оптимізація продуктивності.

Конфігураційні параметри, такі як налаштування бази даних, шляхи до ключів, секрети для токенів, а також інші змінні середовища, зберігаються в окремих .env файлах для кожного сервісу. Це дозволяє гнучко змінювати середовище без внесення змін до коду, а також полегшує роботу з CI/CD, де часто необхідна автоматизація під час розгортання нових версій. Зберігання таких налаштувань окремо від коду також сприяє безпеці, оскільки ці файли не потрапляють до систем контролю версій.

Фізична структура репозиторію відображає принципи модульності, чіткого розділення обов’язків між клієнтською і серверною частинами, а також зручність у навігації для розробників. У корені репозиторію (Рисунок 3.2) розміщено основні конфігураційні файли для Docker, Laravel та Node.js середовища. Серверна частина розміщується у стандартних директоріях фреймворку Laravel. Клієнтська частина розміщена в директорії resources/, що відповідає за SPA-додаток.

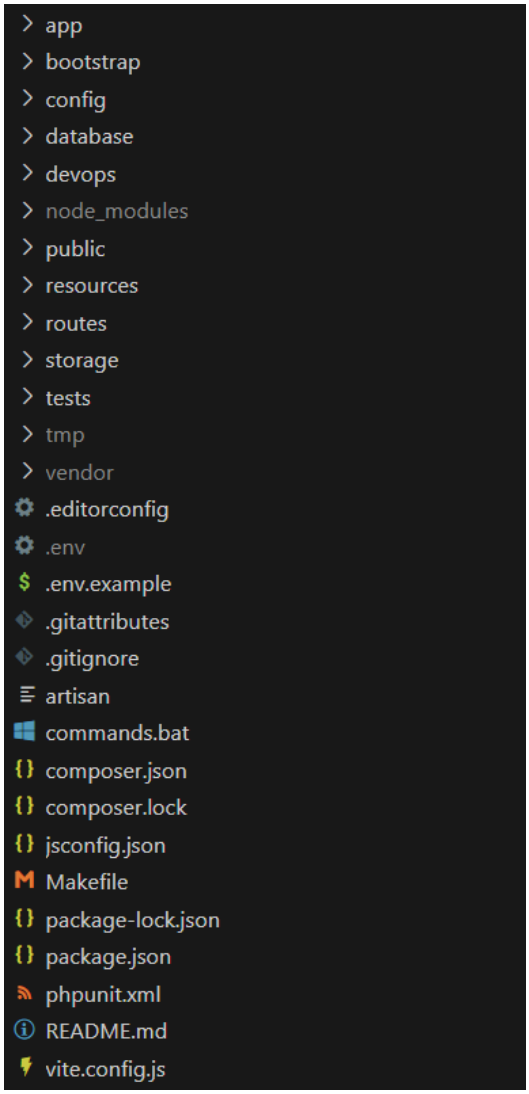


Рисунок 3.2 – Вигляд директорії проекту

У результаті побудована система демонструє збалансовану комбінацію технологій і підходів, що забезпечують високу продуктивність, безпеку та зручність у розробці. Архітектура дозволяє легко адаптуватися до нових вимог, додавати функціональність, не порушуючи вже існуючих залежностей, та оперативно масштабувати проєкт відповідно до зростання навантаження або розширення функціональних можливостей. Такий підхід дає змогу створити не лише ефективну CRM-систему, а й надійну технологічну платформу для подальшого розвитку.

3.2 Реалізація бекенду

Основна мета бекенд-частини — не просто надання даних через API, а формування повноцінного рівня бізнес-логіки, що дозволяє централізовано контролювати права доступу, обробку даних, валідацію, логіку взаємодії між сутностями, а також забезпечити масштабованість і надійність всієї системи.

Особливу увагу в реалізації було приділено авторизації користувачів (Рисунок 3.3). У системі використано Laravel Sanctum, який дозволяє реалізувати безпечну токен-авторизацію з підтримкою SPA-архітектури. Sanctum дозволяє автентифікувати користувача через cookie або персональні токени, і в даному випадку був обраний cookie-based варіант, що ідеально підходить для взаємодії з фронтендом на Vue 3 у форматі SPA. Після проходження авторизації користувач отримує токен, який автоматично додається до кожного наступного HTTP-запиту, що дозволяє уникнути повторного введення облікових даних або необхідності зберігати токен у локальному сховищі.

```
Route::middleware(middleware: ['guest'])->group(callback: function (): void {
    Route::get(uri: '/login', action: function (): View {
        return view(view: 'app');
    })->where(name: 'any', expression: '.*')->name(name: 'login');
    Route::post(uri: '/login', action: [AuthController::class, 'login']);

    Route::get(uri: '/register', action: function (): View {
        return view(view: 'app');
    })->where(name: 'any', expression: '.*')->name(name: 'register');
    Route::post(uri: '/register', action: [AuthController::class, 'register']);
});
```

Рисунок 3.3 – Код роутів авторізації

Архітектурно вся авторизація та контроль доступу до ресурсів реалізовані на стороні сервера. Фронтенд не має жодного уявлення про набір дозволів (permissions) користувача, не отримує списку ролей чи політик доступу. Замість цього після входу система визначає на бекенді, які саме компоненти, маршрути та функціональність доступні даному користувачу, і формує відповідний набір структурованого контенту, що передається клієнтській частині у вигляді JSON-відповіді. Такий підхід дозволяє гарантувати безпеку навіть у разі потенційної

компрометації фронтенду, оскільки усі критичні рішення приймаються виключно на рівні бекенду.

Маршрутизація системи (Рисунок 3.4) реалізована через стандартний механізм Laravel Routing. Для зручності й масштабованості усі маршрути згруповані за middleware та префіксами, що дозволяє чітко розмежовувати зони доступу — публічні, приватні, службові. Використовується REST-підхід: усі запити формуються за допомогою HTTP-методів POST, GET, PUT, DELETE з логічними назвами ресурсів. Приклади маршрутів включають, зокрема, ендпоінти для отримання списків клієнтів, створення нових заявок, редагування профілів партнерів та інші дії, які доступні лише при наявності відповідних прав.

```php
Route::prefix(prefix: 'pages')->group(callback: function (): void {

    Route::get(uri: 'users', action: [UsersController::class, 'index']);
    Route::get(uri: 'users/{id}', action: [UsersController::class, 'show']);
    Route::post(uri: 'users', action: [UsersController::class, 'store']);
    Route::put(uri: 'users/{id}', action: [UsersController::class, 'update']);
    Route::delete(uri: 'users/{id}', action: [UsersController::class, 'destroy']);
    Route::get(uri: 'users/options', action: [UsersController::class, 'options']);

    Route::get(uri: 'services', action: [ServicesController::class, 'index']);
    Route::get(uri: 'services/{id}', action: [ServicesController::class, 'show']);
    Route::post(uri: 'services', action: [ServicesController::class, 'store']);
    Route::put(uri: 'services/{id}', action: [ServicesController::class, 'update']);
    Route::delete(uri: 'services/{id}', action: [ServicesController::class, 'destroy']);

    Route::get(uri: 'orders', action: [OrdersController::class, 'index']);
    Route::get(uri: 'orders/{id}', action: [OrdersController::class, 'show']);
    Route::post(uri: 'orders', action: [OrdersController::class, 'store']);
    Route::put(uri: 'orders/{id}', action: [OrdersController::class, 'update']);
    Route::delete(uri: 'orders/{id}', action: [OrdersController::class, 'destroy']);


    Route::get(uri: 'profile/options', action: [UsersController::class, 'options']);
});
```

Рисунок 3.4 – Код роутів обробки ресурсів на компонентів

Однією з важливих частин функціоналу стала можливість отримання замовлень (Рисунок 3.5) із зовнішніх джерел через інтеграцію з партнерами. У системі реалізовано механізм реєстрації партнерських сервісів, кожен з яких має унікальний API-ключ, що створюється при додаванні партнера в адміністративній панелі. Для передачі даних із зовнішньої системи (наприклад, лендингу або сторонньої платформи) передбачено спеціальний маршрут, який очікує POST-

запити з набором параметрів. Обов’язковими полями запиту є offer_id, name, phone, email, sub1, sub2, sub3, sub4, sub5, source_id. Усі запити до цього маршруту перевіряються на валідність API-ключа, що передається у заголовках запиту, а також проходять додаткову валідацію вхідних даних.

```php
class LeadController extends Controller
{
    2 references
    protected LeadService $leadService;

    0 references | 0 overrides
    public function __construct(LeadService $leadService)
    {
        $this->leadService = $leadService;
    }

    0 references | 0 overrides
    public function store(StoreLeadRequest $request): JsonResponse
    {
        $lead = $this->leadService->createLead($request->validated(), $request->getPartner());

        return response()->json([
            'success' => true,
            'lead_id' => $lead->id,
        ], 201);
    }
}
```

Рисунок 3.5 – Код контроллера по створенню лідів

Обробка лідів реалізована через чітку бізнес-логіку, винесену в окремий шар — Business Logic Layer. Такий підхід дозволяє відокремити обробку запитів від контролерів та перенести логіку в окремі сервісні класи. Наприклад, у випадку створення нового замовлення система спочатку перевіряє автентичність партнера за його ключем, потім виконує валідацію параметрів, формує новий об’єкт моделі Lead, додає відповідні зв’язки до партнерів, джерел та пропозицій (offers), після чого зберігає інформацію в базу даних. Використання Eloquent ORM дозволяє працювати з усіма цими сутностями на високому рівні абстракції, забезпечуючи не лише простоту синтаксису, але й автоматичне керування зв’язками між таблицями.

Для взаємодії з базою даних у системі використовується ORM-інструмент Laravel — Eloquent, який забезпечує зручну та виразну роботу з таблицями бази даних у вигляді об'єктів-моделей. Усі основні сутності, що використовуються в системі — користувачі, партнери, ліди (замовлення), пропозиції (offers) —

реалізовані у вигляді окремих Eloquent-моделей, кожна з яких відображає відповідну таблицю в базі даних. Завдяки використанню Eloquent, розробка взаємодії з БД стала швидкою, читабельною та масштабованою. При цьому не втрачається контроль над складними SQL-запитами — у разі потреби розробник може використовувати Fluent Query Builder або навіть Raw SQL у межах моделей або сервісів. Загалом, реалізація моделей та взаємодії з базою даних в рамках системи відповідає стандартам Laravel-екосистеми і забезпечує гнучкість, стабільність та можливість легкого масштабування у майбутньому.

Окремо варто відзначити, що усі критичні дії логуються, а також проходять через механізми обробки винятків. Це дозволяє централізовано управляти помилками, зберігати деталі запитів у разі неуспішної обробки, а також інформувати відповідальних осіб у разі повторюваних збоїв. Таким чином, система має високу надійність та захист від неправильної інтеграції сторонніх систем.

Реалізація бекенду була виконана з урахуванням принципів розділення відповідальностей, повторного використання коду, масштабованості та відповідності загальноприйнятим стандартам Laravel-розробки. Усі функціональні модулі побудовані таким чином, щоб полегшити майбутнє розширення системи, а також інтеграцію з новими сервісами або зовнішніми API без необхідності зміни вже існуючого коду.

3.3 Реалізація фронтенду

Фронтенд-структура організована в директорії resources, що містить усі основні модулі для роботи застосунку. Всередині знаходиться папка src, яка виконує роль кореневого каталогу джерельного коду. Вона містить логічно поділені підкаталоги: components, pages, router та store, що відповідають відповідним аспектам архітектури клієнтського застосунку.

Папка components містить багаторазові елементи інтерфейсу, які використовуються у різних частинах системи. Це можуть бути кнопки, поля введення, модальні вікна, сповіщення тощо. Їхня повторна використаність забезпечує підтримку єдиного стилю й спрощує обслуговування коду.

Папка pages включає повноцінні сторінки застосунку — окремі екрани, які відповідають певним маршрутам, наприклад: сторінка списку клієнтів, сторінка створення угоди, профіль користувача тощо. Кожна сторінка зазвичай містить кілька компонентів, об’єднаних у логічну структуру, яка представляє собою завершену одиницю функціональності.

Окрему роль у проєкті відіграє папка router, де зберігаються всі маршрути, які визначають, яка саме сторінка буде відображатися при переході на відповідну URL-адресу. Для цього використовується бібліотека Vue Router, що є стандартом маршрутизації у Vue 3. Завдяки цьому забезпечується навігація без перезавантаження сторінки, а також реалізація механізмів захисту маршрутів — наприклад, редирект на сторінку входу у разі відсутності авторизації.

У папці store знаходиться конфігурація централізованого сховища даних (Рисунок 3.6), реалізованого за допомогою Vuex. Це сховище відповідає за збереження стану застосунку — таких даних, як інформація про автентифікованого користувача, поточний стан сесії, сповіщення, кешовані дані та інші глобальні змінні, які мають бути доступні у різних частинах інтерфейсу. Vuex дозволяє централізовано керувати даними та їх змінами за допомогою мутацій, дій і геттерів, що значно спрощує відлагодження та розширення проєкту.

```
import { createStore } from "vuex";
import createPersistedState from "vuex-persistedstate";
const store = createStore({
  state: {
    user: null,
    links: null,
  },
  mutations: {
    setUser(state, user) {
      state.user = user;
    },
    // інші
  },
  actions: {
    setUser({ commit }, user) {
      commit("setUser", user);
    },
    // інші
  },
  getters: {
    getUser: (state) => state.user,
    getLinks: (state) => state.links,
  },
  plugins: [
    createPersistedState({
      paths: ["user", "links"],
    }),
  ],
});
export default store;
```

Рисунок 3.6 – Налаштування Vuex (store.js)

Маршрутизація реалізована за допомогою бібліотеки Vue Router, яка надає можливість створювати SPA з навігацією без перезавантаження сторінки. Для кращої організації коду, конфігурація маршрутів винесена у два окремі файли: routes.js (Рисунок 3.7) — для декларативного опису маршрутів, і router.js — для створення та налаштування екземпляра маршрутизатора.

```
import { createWebHistory, createRouter } from 'vue-router'
import routes from '@/router/routes.js'

const router = createRouter({
    history: createWebHistory(),
    routes,
});

export default router;
```

Рисунок 3.7 – Налаштування router.js

Файл App.vue (Рисунок 3.8) є кореневим компонентом усього SPA-застосунку. Саме в ньому визначено основну HTML-структуру застосунку, базові стилі, а також блок <router-view>, який відповідає за динамічне завантаження контенту відповідно до маршруту. Іншими словами, цей компонент слугує контейнером для всіх сторінок і виконує функцію шаблону візуального оформлення.

```vue
<template>
    <main class="main">
        <router-view></router-view>
    </main>
</template>

<script>
export default {
    setup() {
        return {};
    },
};
</script>
```

Рисунок 3.8 – Шаблон App.vue

Файл app.js (Рисунок 3.9) — це вхідна точка JavaScript-застосунку. Саме тут ініціалізується Vue-додаток, підключається маршрутизатор, реєструється сховище Vuex, а також імпортується App.vue. Із цього файлу починається виконання клієнтської логіки, він забезпечує запуск усіх модулів та їхню інтеграцію.

```js
import './bootstrap';
import { createApp } from 'vue';
import App from '@/App.vue';
import 'bootstrap/dist/css/bootstrap.min.css';
import 'bootstrap/dist/js/bootstrap.bundle.min.js';
import 'bootstrap-icons/font/bootstrap-icons.css';
import Router from '@/router/router.js';
import Store from '@/store/store.js'

createApp(App).use(Store).use(Router).mount('#app');
```

Рисунок 3.9 – Код налаштування app.js

Файл bootstrap.js (Рисунок 3.10) є проміжною точкою, в якій ініціалізуються глобальні залежності та налаштування, зокрема — HTTP-клієнт, CSRF-токени, обробники помилок, налаштування локалізації або інші бібліотеки, необхідні для коректної роботи застосунку. Його використання відповідає загальноприйнятим стандартам у Laravel-проєктах, де підготовка оточення для фронтенду відбувається централізовано.

```js
import axios from 'axios';
window.axios = axios;

axios.defaults.withCredentials = true;
axios.defaults.withXSRFToken = true;

window.axios.defaults.headers.common['X-Requested-With'] = 'XMLHttpRequest';
```

Рисунок 3.10 – Код налаштування bootstrap.js

Збірка та обробка модулів виконується за допомогою сучасного інструмента Vite, який прийшов на зміну Webpack і забезпечує надзвичайно швидку розробку завдяки hot module replacement (HMR), оптимізації імпортів та мінімізації часу збирання. Використання Vite дозволило значно скоротити час компіляції, пришвидшити інтерактивне тестування змін у компоненті в реальному часі, а також легко масштабувати проєкт, додаючи нові залежності без перевантаження конфігураційних файлів.

Загалом, така структура фронтенду (Рисунок 3.11) дозволяє чітко організувати код, спростити навігацію у проєкті, забезпечити стабільну основу для розвитку системи, а також легко адаптувати CRM-платформу до нових функціональних вимог без суттєвих змін у вже існуючих компонентах.

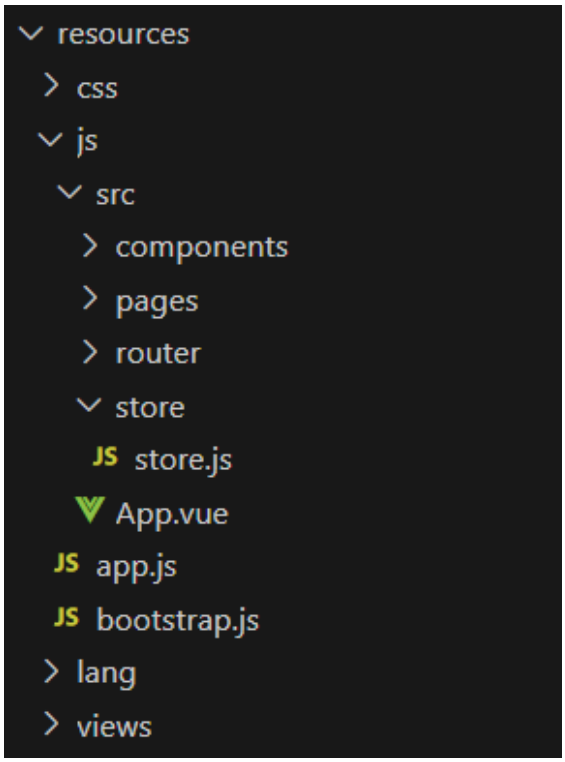


Рисунок 3.11 – Архітектура директорії фронтенд частини проекту

Варто зазначити, що під час розробки основний акцент був зроблений не на візуальному оформленні чи естетиці інтерфейсу, а передусім на реалізації функціональності, обробці даних, маршрутизації, авторизації, валідації форм та інтеграції з бекендом через REST API. Верстці у цьому проєкті було приділено

мінімум часу, оскільки вона виконувала допоміжну роль — забезпечити базову зручність взаємодії з елементами інтерфейсу, не відволікаючи ресурси від реалізації основної бізнес-логіки. Такий підхід дозволив зосередитися на створенні повноцінного функціонального середовища, яке є ядром системи, залишаючи можливість подальшого вдосконалення дизайну на наступних етапах розробки. Приклад оформлення інтерфейсу показаний на рисунку 3.12.

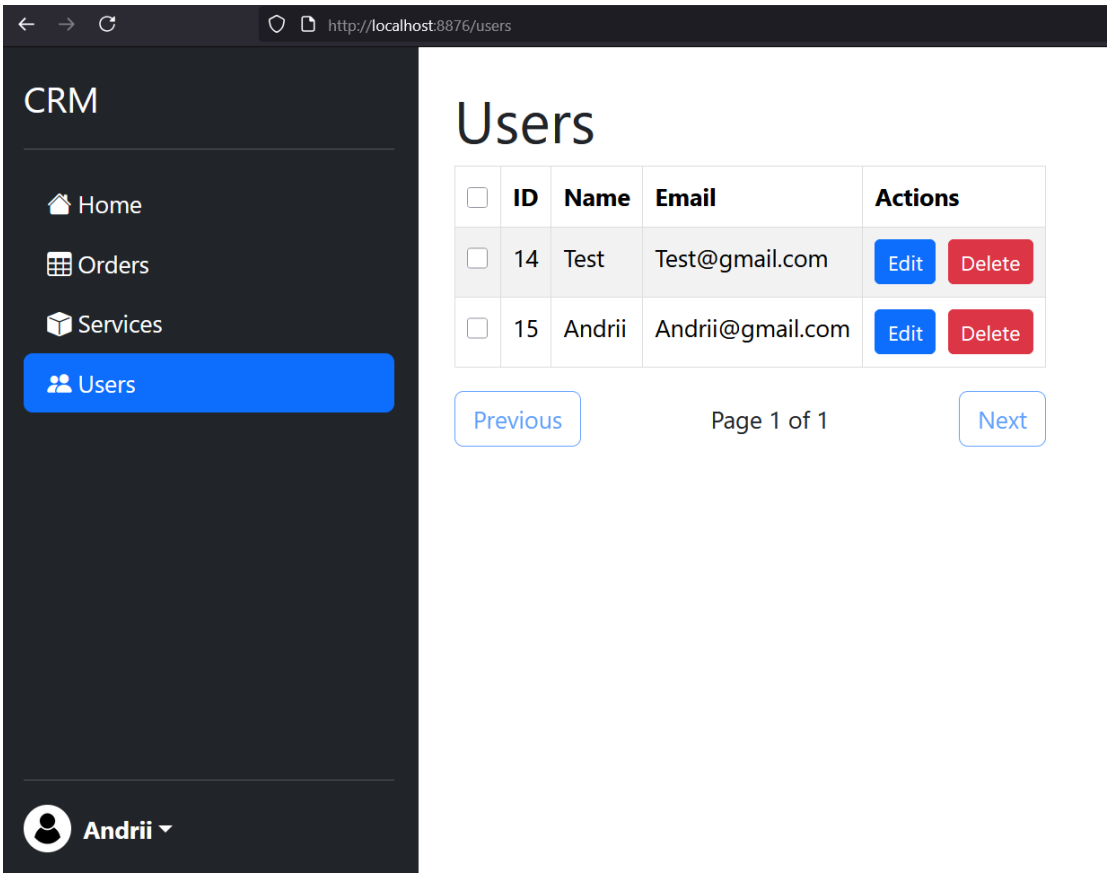


Рисунок 3.12 – Загальний вигляд додатку та сторінки користувачів

3.4 Забезпечення безпеки та масштабованості, керування версіями

У рамках реалізації CRM-системи особлива увага була приділена безпеці та масштабованості, оскільки проєкт передбачає багатокористувацьку роботу з чутливими даними клієнтів, комерційною інформацією та персональними запитами. Сучасні веб-додатки повинні не лише забезпечувати функціональність, а й дотримуватися найкращих практик захисту, ізоляції та гнучкого розгортання. З цією метою в системі були впроваджені наступні рішення: аутентифікація за

допомогою Laravel Sanctum, конфігурація політики CORS для забезпечення контролю доступу між доменами, а також контейнеризація із застосуванням Docker та Docker Compose.

Laravel Sanctum був обраний як основний механізм аутентифікації для цієї CRM-системи завдяки своїй гнучкості, простоті реалізації та безпечному підходу до керування токенами. Sanctum дозволяє реалізувати як аутентифікацію через SPA (Single Page Application), так і API-токени — відповідно, він ідеально підходить для взаємодії між Vue.js фронтендом та Laravel-бекендом. У контексті SPA-архітектури Sanctum використовує захищені HTTP-only cookie та Laravel middleware для забезпечення сесійної аутентифікації без потреби зберігати токени на стороні клієнта, що значно знижує ризики XSS-атак. При цьому ключову роль у захисті від CSRF (Cross-Site Request Forgery) атак відіграє механізм передачі CSRF-токену через спеціальний HTTP-заголовок X-XSRF-TOKEN. X-XSRF-TOKEN — це спеціальний HTTP-заголовок, який автоматично надсилається клієнтським JavaScript при кожному state-changing запиті (POST, PUT, DELETE тощо) до сервера. Він містить значення CSRF-токена, яке Laravel зберігає в cookie XSRF-TOKEN (звичайно, не HTTP-only, щоб JavaScript міг його прочитати). Як працює логіка з XSRF-TOKEN:

Laravel генерує унікальний CSRF-токен при автентифікації користувача та зберігає його у cookie XSRF-TOKEN.

Цей cookie не має атрибуту HttpOnly, тому клієнтський код Vue.js може прочитати токен.

Axios (або інша HTTP-бібліотека) налаштований автоматично брати значення з cookie XSRF-TOKEN і додавати його в заголовок X-XSRF-TOKEN кожного state-changing запиту.

Сервер перевіряє, чи співпадає токен із очікуваним значенням і дозволяє обробити запит лише якщо перевірка успішна.

Налаштування Sanctum здійснюється шляхом встановлення пакету через Composer (composer require laravel/sanctum), публікації конфігурацій (php artisan vendor:publish) та підключення middleware у routes/api.php та routes/web.php і Kernel.php. Далі, на стороні клієнта, Vue.js взаємодіє з Laravel через API, а Laravel автоматично перевіряє авторизацію запитів за допомогою cookies. Цей механізм дозволяє централізовано контролювати доступ, обмежувати сесії, реалізувати logout, та керувати правами доступу на основі ролей користувачів. Приклад захищених роутів наведено на рисунку 3.13:

```
Route::middleware(middleware: ['auth'])->group(callback: function (): void {
    Route::prefix(prefix: 'components')->group(callback: function (): void {
        Route::get(uri: 'navigation/options', action: [SidebarMenuController::class, 'options']);
        // інші
    });

    // інші

    Route::post(uri: '/logout', action: [AuthController::class, 'logout'])->name(name: 'logout');
    Route::get(uri: '{any}', action: function (): View {
        return view(view: 'app');
    })->where(name: 'any', expression: '.*');
});
```

Рисунок 3.13 – Налаштування роутів для авторизованих користувачів

CORS (Cross-Origin Resource Sharing) є необхідним компонентом у безпечній взаємодії між фронтендом і бекендом, особливо у випадку, коли сервер і клієнт розміщені на різних доменах або портах. Без правильної CORS-конфігурації браузери блокують запити через політику same-origin. У цьому проєкті фронтенд Vue.js працює через Webpack Dev Server (порт 5173), а бекенд Laravel розміщений на сервері Nginx у контейнері. Тому була необхідність у точному налаштуванні CORS, щоб дозволити лише допустимі джерела, методи та заголовки.

Для цього був використаний офіційний пакет Laravel CORS, який дозволяє централізовано контролювати політику доступу в config/cors.php. Було дозволено доступ лише з http://localhost:5173 (або інших авторизованих доменів), дозволено методи GET, POST, PUT, DELETE, а також необхідні заголовки для передачі токенів авторизації. Це не лише дозволяє клієнту взаємодіяти з API, але й мінімізує ризик несанкціонованого доступу до ресурсу.

Контейнеризація дозволила реалізувати ізольоване середовище для кожного компонента системи: додатку, бази даних, веб-сервера та допоміжних сервісів. Весь інфраструктурний стек розміщений у директорії devops, яка структурована наступним чином:

- app/ — містить Dockerfile з інструкціями побудови PHP-контейнера та файл php.ini для налаштувань середовища виконання PHP.
- workspace/ — окремий контейнер для розробницького середовища, що включає Bash, Composer, npm, artisan тощо.
- nginx/conf.d/ — конфігурації для Nginx, який працює як зворотний проксі та обслуговує фронтенд.
- tmp/ — службова директорія для тимчасових файлів.
- docker-compose.yml — центральний конфігураційний файл, що описує усі сервіси, мережі, змінні середовища та обсяги даних.

Ось вміст docker-compose.yml для повного розуміння взаємодії контейнерів:

```
version: '3'


services:
  webserver:
    image: nginx
    volumes:
      - ../:/var/www
      - ./nginx/conf.d/nginx.conf:/etc/nginx/conf.d/default.conf
    ports:
    - "8876:80"
```

```
  depends_on:
  - app
  container_name: crm_ngnix
  networks:
    - laravel

db:
  image: mysql:8.0
  restart: always
  volumes:
    - ../tmp/db:/var/lib/mysql
  environment:
    MYSQL_DATABASE: dev
    MYSQL_ROOT_PASSWORD: root
  ports:
    - "8101:3306"
  command: mysqld --character-set-server=utf8 --collation-server=utf8_unicode_ci
  container_name: crm_db
  networks:
    - laravel

app:
```

```
  build:
    context: .
    dockerfile: app/Dockerfile
  volumes:
    - ../:/var/www
  depends_on:
    - db
  container_name: crm_app
  networks:
    - laravel

workspace:
  build:
    context: .
    dockerfile: workspace/Dockerfile
  container_name: crm_workspace
  volumes:
    - ../:/var/www
  depends_on:
    - db
  networks:
    - laravel
```

tty: true

ports:

- "5173:5173"

networks:

laravel:

driver: bridge

Для підвищення зручності розробки та обслуговування проєкту були створені сценарії керування. Для Windows-середовища це commands.bat (Рисунок 3.14), що дозволяє запускати систему, накочувати міграції, відкривати термінали у контейнерах, або зупиняти всю систему за допомогою простих команд:

app > commands.bat

```
@echo off

if "%1"=="up" (
    cd /d "%~dp0devops" && docker compose up -d
    goto :eof
)

if "%1"=="settings" (
    docker exec -it crm_app bash -c "composer update --no-cache && php artisan key:generate"
    goto :eof
)

if "%1"=="migrate" (
    docker exec -it crm_app bash -c "php artisan migrate"
    goto :eof
)

if "%1"=="stop" (
    cd /d "%~dp0devops" && docker compose down
    goto :eof
)

if "%1"=="cli" (
    docker exec -it crm_workspace bash
    goto :eof
)

if "%1"=="db_cli" (
    docker exec -it crm_db /bin/bash
    goto :eof
)
```

Рисунок 3.14 – Вміст commands.bat

Для Unix-систем було створено Makefile (Рисунок 3.15), який має еквівалентні цілі та дозволяє виконувати ті самі операції через make up, make migrate, make cli тощо:

app > Makefile

```
up:
    cd devops && docker compose up -d
settings:
    docker exec -it crm_app bash -c "composer update && php artisan key:generate"
migrate:
    docker exec -it crm_app bash -c "php artisan migrate"
stop:
    cd devops && docker compose down
cli:
    docker exec -it crm_workspace bash
db_cli:
    docker exec -it crm_db /bin/bash
```

Рисунок 3.15 – Вміст Makefile

Застосування Laravel Sanctum забезпечило надійний захист користувацьких сесій та гнучку аутентифікацію в SPA-архітектурі. Налаштування політики CORS дозволило безпечно реалізувати взаємодію між фронтендом та бекендом, обмеживши доступ лише авторизованим джерелам. У свою чергу, Docker і Docker Compose створили фундамент для стабільного, контрольованого і багаторазово відтворюваного середовища, яке легко масштабувати та розгортати на різних серверах. Усі ці елементи разом формують цілісну, безпечну та технологічно зрілу систему, яка відповідає сучасним вимогам веб-розробки.

Для ефективної командної роботи, контролю змін та збереження історії проєкту, CRM-система розміщена у віддаленому репозиторії на платформі GitHub. Це забезпечує централізований доступ до коду, автоматизацію процесів CI/CD (у перспективі).

ВИСНОВКИ

У процесі реалізації дипломного проєкту було досягнуто основної мети — створено прототип CRM-системи, який об'єднує сучасні принципи проєктування програмного забезпечення, сучасний технологічний стек і практики безпечної взаємодії користувачів із системою. Розроблена система не лише виконує базові функції з управління клієнтською базою, угодами, завданнями та подіями, але й демонструє зрілість архітектурних рішень, що дозволяє масштабувати її функціонал відповідно до потреб реального підприємства.

Робота охопила повний цикл розробки — від дослідження еволюції концепції CRM та вибору відповідних технологій до моделювання даних, реалізації логіки на сервері, побудови інтерфейсу користувача й забезпечення безпеки інформації. Значна увага була приділена питанням інтеграції: у системі реалізовано чіткий розподіл відповідальностей між фронтендом і бекендом, налаштовано REST API, забезпечено підтримку токен-базованої авторизації з використанням Laravel Sanctum, а також впроваджено механізми CORS-контролю, логування, валідації та обробки помилок.

Особливістю реалізації стала архітектурна ізоляція компонентів системи через контейнеризацію. Завдяки Docker і Docker Compose забезпечено відтворюваність середовища розробки, стабільність розгортання та зручність тестування. Розмежування доступу між сервісами на мережевому рівні, централізоване управління конфігураціями через .env-файли, автоматизація

запуску через Makefile та командні скрипти — усе це дозволило максимально наблизити умови реалізації до стандартів промислової розробки.

З погляду клієнтського інтерфейсу, Vue.js і Vite забезпечили високу швидкодію, простоту в реалізації SPA-підходу та зручність управління компонентами й глобальним станом застосунку через Vuex. Реалізовано базову

маршрутизацію, захист маршрутів, повторне використання компонентів, централізовану обробку даних, а також логіку взаємодії з API на основі Axios.

Загалом, створена система відповідає вимогам модульності, повторного використання коду, безпеки, масштабованості та продуктивності. Вона демонструє концептуальну і технологічну готовність до подальшого розвитку — інтеграції з платіжними системами, email-маркетингом, зовнішніми аналітичними платформами, а також до впровадження кастомної логіки на основі ML або rule-based engine. Окрім того, система може бути доопрацьована з урахуванням мобільного клієнта, багатомовності, розширеної аналітики та підтримки нових ролей користувачів.

Таким чином, реалізована CRM-система не лише виконала своє навчальне призначення, а й заклала міцний технологічний фундамент для створення повноцінного цифрового продукту в реальних умовах. Вона ілюструє здатність студента не лише до реалізації програмного рішення, але й до розуміння складних взаємозв’язків між бізнес-вимогами, технічними обмеженнями та сучасними технологічними трендами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ ТА ЛІТЕРАТУРИ

1. Payne A., Frow P. Strategic Customer Management: Integrating Relationship Marketing and CRM. 2-nd ed. Cambridge : Cambridge University Press, 2021. 428 p.
2. ISO/IEC 25010:2011 Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models. Женева : ISO, 2011. 26 p.
3. ДСТУ 8302:2015. Бібліографічне посилання. Загальні положення та правила складання. Київ : ДП «УкрНДНЦ», 2016. 16 с.
4. The European Parliament and the Council. Regulation (EU) 2016/679 (General Data Protection Regulation). Official Journal of the European Union. 2016-04-27.
5. Laravel LLC. Laravel Documentation [Електронний ресурс]. URL: https://laravel.com/docs/ (дата звернення: 15.05.2025).
6. Vue.js. Vue.js Documentation [Електронний ресурс]. URL: https://vuejs.org/guide/introduction.html (дата звернення: 15.05.2025).
7. Docker Inc. Docker Documentation [Електронний ресурс]. URL: https://docs.docker.com/ (дата звернення: 15.05.2025).
8. Harris M. Laravel Octane in Production: A Performance Case Study // Medium. 2024-11-02. URL: https://medium.com/@mharris/octane-case-study (дата звернення: 15.05.2025).
9. Vue Foundation. Vue .js 3 Guide [Електронний ресурс]. URL: https://vuejs.org/guide(дата звернення: 15.05.2025).
10. Wieruch R. The Road to React: Your Journey to Master React.js in 2025. 6-th ed. Berlin : Robin Wieruch, 2025. 275 p.
11. Amazon Web Services, Inc. AWS Well-Architected Framework. Seattle : AWS, 2024. 121 p.

12. Grinberg M. Flask Web Development: Developing Web Applications with Python. 2-nd ed. Sebastopol : O'Reilly, 2024. 350 p
13. Newman M. Building Microservices. 2-nd ed. Sebastopol : O'Reilly, 2023. 493 p.
14. Jamshidi P. Managing Technical Debt in Software Engineering Projects // IEEE Software. 2022. Vol. 39, № 5. P. 58-65.
15. JMeter Community. Apache JMeter 5.6 User Manual [Електронний ресурс]. URL: https://jmeter.apache.org/usermanual(дата звернення: 15.05.2025).
16. Organisation for Economic Co-operation and Development. Digital Economy Outlook 2024. Paris : OECD Publishing, 2024. 221 p.
17. Humble J., Farley D. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Addison-Wesley, 2023. 512 p.
18. Gartner Inc. Magic Quadrant for Sales Force Automation Platforms 2024. Stamford : Gartner Research, 2024. 31 p.
19. Statista GmbH. Average Annual Salary of Software Developers in Ukraine 2019-2025 [Електронний ресурс]. URL: https://www.statista.com/statistics/ukraine-developer-salary (дата звернення: 15.05.2025).
20. Boardman J., Kagan C. Systems Thinking: Coping with 21st Century Problems. London : CRC Press, 2022. 278 p.

ПРЕЗЕНТАЦІЯ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «CRM система мовами PHP та JavaScript засобами Laravel та Vue.js»
на здобуття освітнього ступеня бакалавра
зі спеціальності 122 Комп'ютерні науки
освітньо-професійної програми Комп'ютерні науки

Виконав: Студент 4 курсу, групи КНД-43
Яременко Андрій Олексійович
Керівник: керівник кафедри Комп'ютерних наук
д.т.н., професор Вишнівський В.В.

Київ - 2025

Актуальність теми

CRM-системи є ядром цифрової трансформації бізнесу.
Вони дозволяють автоматизувати продажі, маркетинг, обслуговування та аналітику.

У 2020-х роках важливо:
- Обробка великих обсягів даних
- Захист персональної інформації (GDPR)
- Мобільність і персоналізація

Мета

Створення прототипу сучасної CRM-системи.

Основні завдання:

1. Реалізувати ключовий функціонал (клієнти, угоди, сервіси)
2. Побудувати SPA-інтерфейс
3. Забезпечити безпеку та масштабованість
4. Відповідність GDPR

Мета

Створення прототипу сучасної CRM-системи.

Основні завдання:

1. Реалізувати ключовий функціонал (клієнти, угоди, сервіси)
2. Побудувати SPA-інтерфейс
3. Забезпечити безпеку та масштабованість
4. Відповідність GDPR

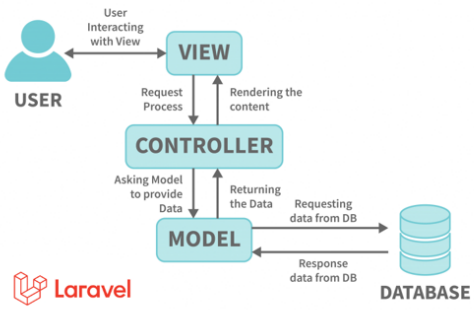
Вибір технологій

Фронтенд:
Vue.js 3 + Vite

Інфраструктура:
Docker

Бекенд:
Laravel

Архітектура бекенду



MVC — це архітектура, що розділяє застосунок на три частини: Model, View, Controller.

MVC у Laravel:

- **Model:** Eloquent ORM (наприклад, User, Client)
- **View:** Blade-шаблони або API-відповіді
- **Controller:** методи у app/Http/Controllers (наприклад, ClientController)

У SPA-додатках Laravel відповідає за **Model + Controller + API**, а **View** — у Vue.js

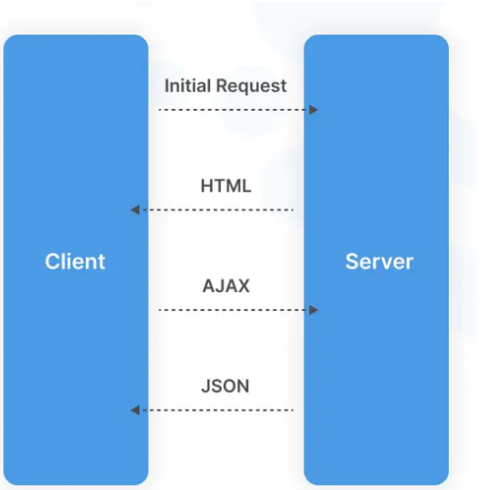
Архітектура фронтенду

SPA — це веб-застосунок, який працює на одній HTML-сторінці. Завантаження даних і оновлення інтерфейсу відбувається без перезавантаження сторінки.

Переваги SPA:

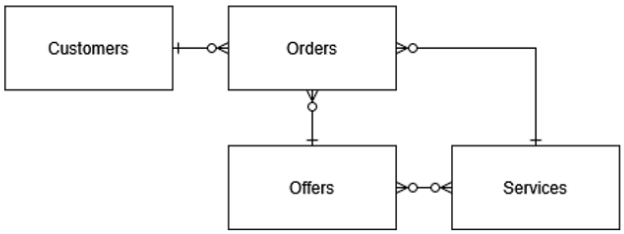
- Швидка робота інтерфейсу
- Взаємодія з сервером через API (JSON)
- Плавна навігація між сторінками
- Зручно для мобільних пристроїв

Сервер (**Laravel**) повертає лише дані — вся логіка та інтерфейс у браузері.



Модель даних

Базу даних нормалізовано до 3NF. Враховано вимоги до зберігання персональних даних згідно з GDPR.



Безпека системи

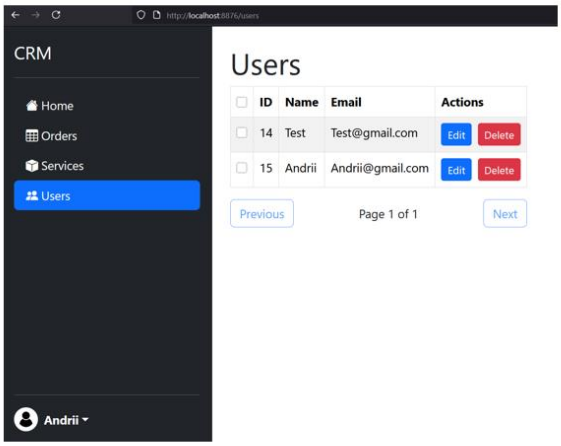
- Авторизація через Laravel Sanctum
- Рольовий доступ і політики (Backend-first)
- HTTPS, CSRF-захист, логування подій

SPA-фронтенд отримує лише дозволені маршрути, що знижує ризики.

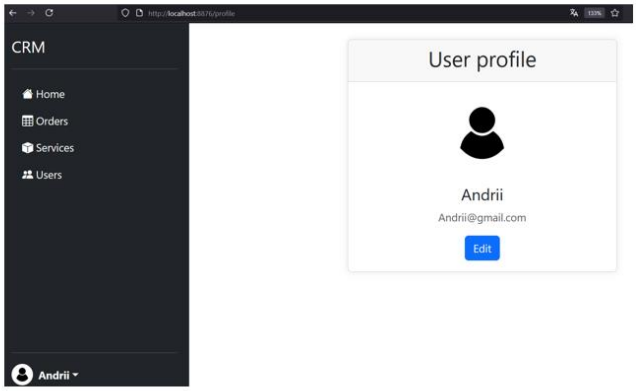
Переваги реалізованої системи

- Висока продуктивність (Vite + SPA)
- Захищеність (Laravel Sanctum + політики)
- Масштабованість (контейнерна архітектура)
- Зручність UI/UX (Vue + компонентний підхід)
- Гнучкість у розширенні (API-first, модульність)

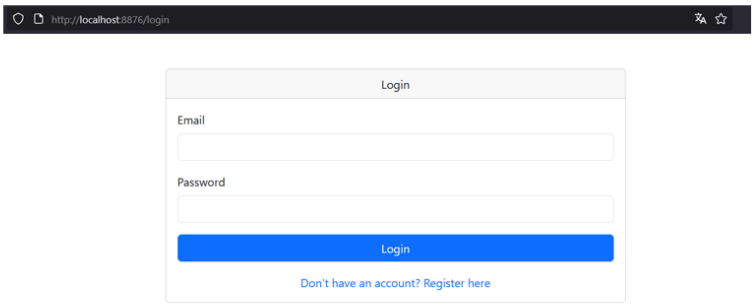
Скріншоти з додатку



Скріншоти з додатку



Скріншоти з додатку



Результати та висновки

У результаті реалізовано прототип CRM-системи, який поєднує сучасні підходи до архітектури, безпеки та UI. Система охоплює базовий функціонал для управління клієнтами, угодами, завданнями і подіями, а також демонструє готовність до масштабування та інтеграції.

Розробка включала повний цикл: аналіз CRM-концепцій, вибір технологій, побудову REST API, реалізацію SPA-інтерфейсу та захист даних. Особливу увагу приділено розділенню бекенду та фронтенду, авторизації через Laravel Sanctum, логуванню і валідації.

Завдяки Docker система легко розгортається, а інтерфейс на Vue.js з Vite забезпечує швидкодію і зручність у роботі. Реалізовано захист маршрутів, повторне використання компонентів та централізовану обробку даних.

Система відповідає вимогам модульності, безпеки та продуктивності. Вона готова до подальшого розвитку — від мобільного клієнта до інтеграції з аналітичними або платіжними сервісами.

Цей проєкт демонструє не лише технічні навички, а й сучасні вимог до бізнес-орієнтованого ПЗ.