Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

РОЗРОБКА CRM-СИСТЕМИ З ФУНКЦІЯМИ КОМУНІКАЦІЇ ТА МОНІТОРИНГУ ЕФЕКТИВНОСТІ ПРАЦІВНИКІВ З ВИКОРИСТАННЯМ LARAVEL ТА ІНТЕГРАЦІЄЮ GOOGLE API

DEVELOPMENT OF A CRM SYSTEM WITH COMMUNICATION AND EMPLOYEE PERFORMANCE MONITORING FUNCTIONS USING LARAVEL AND GOOGLE API INTEGRATION

спеціальність 121 «Інженерія програмного забезпечення»
освітня програма «Інженерія програмного забезпечення»

Виконав: здобувач вищої освіти
групи ІПЗ-42
Кіц Іван Іванович

Керівник:
Ліщина Наталія Миколаївна

Кваліфікаційну роботу
допущено до захисту
«10» 06 2025 р.
Гарант освітньої програми:
к.т.н., доцент
Ліщина Наталія Миколаївна

Луцьк – 2025 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *інженерії програмного забезпечення*
Ступінь вищої освіти *бакалавр*
Галузь знань: *12 «Інформаційні технології»*
Спеціальність: *121 «Інженерія програмного забезпечення»*
Освітня програма: «*Інженерія програмного забезпечення»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____
«*20*» ___*12*___ 20*24*р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Кіцу Івану Івановичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *«Розробка CRM-системи з функціями комунікації та моніторингу ефективності працівників з використанням Laravel та інтеграцією Google API».*
Керівник роботи: *к.т.н., доцент Ліщина Н. М.*
затверджені наказом закладу вищої освіти від *«19» грудня 2024 р.* № *474/01–02*
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи бакалавра *«10» червня 2025 р.*
3. Вихідні дані до роботи*: платформа створення дизайну Figma, UML діаграми, Docker Desktop, Docker, Visual Studio Code, Beaver, Blade, Laravel, PHP, PostgreSQL, Google Maps API, методичні вказівки до виконання кваліфікаційної роботи бакалавра.*
4. Зміст розрахунково–пояснювальної записки (перелік питань, що потрібно розробити): *проаналізувати проблеми комунікації в будівельній сфері, визначити вимоги до системи та виконати її проєктування, спроєктувати архітектуру й реалізовано функціонал оформлення замовлень, призначення завдань і контролю виконання, провести оцінку зручності використання, впливу на бізнес-процеси, тестування продуктивності та користувацького досвіду.*
5. Перелік графічного матеріалу: *13 рисунків, 8 таблиць, 1 додаток.*

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз предметної області	*Ліщина Н. М.*		
Специфікація вимог до розробленої системи	*Ліщина Н. М.*		
Розробка об`єкта проектування	*Ліщина Н. М.*		
Нормоконтроль	*Вознюк А. В.*		
Гарант ОП	*Ліщина Н. М.*		
Показник запозичень тексту	3,41 %		
Академічна доброчесність	*Ліщина Н. М.*		

7. Дата видачі завдання *«20» грудня 2024 р.*

№	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Примітка
1	Огляд літературних джерел по темі кваліфікаційної роботи бакалавра	до 04.02.2025 р.	
2	Аналіз проблеми розробки та впровадження об'єкту проектування	до 01.03.2025 р.	
3	Обгрунтування вибору шляхів, технологій і засобів вирішення поставленого завдання	до 15.03.2025 р.	
4	Розробка функціонально–структурної схеми роботи об'єкта проектування та проектування бази даних	до 29.03.2025 р.	
5	Практична реалізація об'єкта проектування та розробка бази даних	до 26.04.2025 р.	
6	Тестування та налагодження об'єкта проектування	до 03.05.2025 р.	
7	Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедру	до 10.06.2025 р.	

Здобувач вищої освіти _____ Іван КІЦ

Керівник кваліфікаційної роботи _____ Наталія ЛІЩИНА

АНОТАЦІЯ

Кіц І. І. Розробка CRM-системи з функціями комунікації та моніторингу ефективності працівників з використанням Laravel та інтеграцією Google API. Рукопис. Кваліфікаційна робота бакалавра ОП «Інженерія програмного забезпечення» спеціальності «Інженерія програмного забезпечення». Луцький національний технічний університет. Луцьк, 2025.

Кваліфікаційна робота бакалавра складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатка.

У першому розділі здійснено теоретичний огляд предметної області: розглянуто сутність CRM-систем, їх основні функції, переваги та недоліки, а також представлено аналіз існуючих аналогів. На основі цього було сформульовано завдання дипломної роботи – розробка CRM-системи для будівельної компанії з урахуванням потреб клієнтів і співробітників.

В другому розділі проаналізовано вимоги до розроблюваної системи: користувацькі, функціональні та нефункціональні. Визначено архітектуру майбутньої системи, обрано стек технологій, розроблено структуру бази даних та описано основні компоненти інтерфейсу користувача. Також розглянуто питання безпеки та адаптивності.

У третьому розділі описано безпосередньо процес реалізації системи: наведено фрагменти коду, реалізовано функціонал реєстрації, авторизації, ролей користувачів (адміністратор, клієнт, працівник), а також механізм створення та управління завданнями. Продемонстровано інтерфейс користувача, розглянуто приклади роботи основних функцій.

У висновках підбито підсумки виконаної роботи, підтверджено досягнення поставленої мети, наведено практичну значущість створеної CRM-системи та окреслено можливі напрями подальшого розвитку проєкту.

Ключові слова: Docker Desktop, Docker, VS Code, Beaver, Blade, Laravel, PHP, PostgreSQL, Google Maps API.

ABSTRACT

Kits I. Development of a CRM System with Communication and Employee Performance Monitoring Functions Using Laravel and Google API Integration. Manuscript. Qualification work of the bachelor's program "Software engineering" in the specialty "Software engineering". Lutsk National Technical University. Lutsk, 2025.

The bachelor's qualification thesis consists of an introduction, three chapters, conclusions, a list of references, and appendice.

The first section provides a theoretical overview of the subject area: the essence of CRM systems, their main functions, advantages and disadvantages are considered, and an analysis of existing analogues is also presented. Based on this, the thesis task was formulated - the development of a CRM system for a construction company taking into account the needs of customers and employees.

The second section analyzes the requirements for the developed system: user, functional and non-functional. The architecture of the future system is determined, a technology stack is selected, a database structure is developed and the main components of the user interface are described. Security and adaptability issues are also considered.

The third section describes the process of implementing the system itself: code fragments are provided, the functionality of registration, authorization, user roles (administrator, client, employee) is implemented, as well as the mechanism for creating and managing tasks. The user interface is demonstrated, examples of the operation of the main functions are considered.

The conclusions summarize the work performed, confirm the achievement of the set goal, present the practical significance of the created CRM system, and outline possible directions for further development of the project.

Keywords: Docker Desktop, Docker, VS Code, Beaver, Blade, Laravel, PHP, PostgreSQL, Google Maps API.

ЗМІСТ

ВСТУП

Актуальність теми. У сучасному світі, де темп життя постійно зростає, питання ефективного використання часу стає надзвичайно важливим як для окремих людей, так і для організацій. З кожним роком усе більше сфер людської діяльності переходять у цифровий формат, і це створює нові можливості для оптимізації повсякденних та бізнес-процесів. Проте попри стрімкий розвиток цифрових технологій, багато компаній, особливо у сфері будівництва, досі використовують застарілі моделі взаємодії з клієнтами, що призводить до значних втрат часу, зниження продуктивності та невдоволення споживачів. Однією з найпоширеніших причин неефективності в цій галузі є відсутність сучасних онлайн-інструментів для оперативного та прозорого обміну інформацією між замовниками та виконавцями.

На практиці це означає, що клієнти змушені особисто відвідувати офіси будівельних компаній, довго чекати відповіді на телефонні дзвінки або електронні листи, самостійно узгоджувати графіки робіт і відстежувати їх виконання. Усе це призводить до надмірного навантаження на персонал, частих непорозумінь, затримок у виконанні завдань і, як наслідок, зниження рівня довіри до компанії. Такий підхід вже не відповідає очікуванням сучасного клієнта, який прагне зручності, швидкості та максимальної автоматизації всіх етапів замовлення послуг.

Попри наявність низки цифрових платформ, які частково вирішують подібні проблеми, більшість з них є універсальними і не адаптованими до потреб саме будівельної галузі. Вони часто мають перевантажений інтерфейс, складну навігацію, а також вимагають значних витрат часу на навчання персоналу та клієнтів. Крім того, такі сервіси рідко враховують специфіку будівельних робіт, необхідність гнучкого управління бригадами, ведення звітності, контролю виконання завдань та прямої комунікації між сторонами. Через це значна частина замовників продовжує використовувати традиційні, часто неефективні методи взаємодії з будівельними компаніями.

У зв’язку з цим метою даної роботи є розробка спеціалізованої веб-програми для будівельної фірми, яка забезпечить сучасний, зручний та швидкий спосіб оформлення замовлень, формування та призначення завдань працівникам, а також надасть інструменти для прозорого контролю за їх виконанням. Передбачається створення цифрового середовища, яке сприятиме більш тісній взаємодії між замовником та виконавцем, з можливістю відслідковування статусу виконання робіт у реальному часі.

Об’єктом кваліфікаційної роботи виступає інформаційна система для замовлення та організації будівельних послуг.

Предметом кваліфікаційної роботи – розроблений веб-додаток, що реалізує функціонал дистанційної взаємодії між клієнтами та компанією. У процесі реалізації розглядаються не лише технічні аспекти створення веб-програми, але й питання побудови зручного користувацького інтерфейсу, забезпечення стабільної роботи системи, її масштабованості, безпеки обробки даних та інтеграції з іншими сервісами.

З огляду на поставлену мету, у роботі передбачено розв’язання таких завдань:

– проаналізувати сучасні проблеми у сфері комунікації між клієнтами та будівельними компаніями;

– дослідити вимоги до майбутньої системи спеціалізованої веб-програми та виконати її проєктування;

– розробити UML-діаграми, які проілюструють її логіку і створити дизайн сторінок;

– спроєктувати архітектуру додатку, реалізувати функціонал дистанційного оформлення замовлень, призначення завдань та контролю за їх виконанням;

– оцінити ефективність запропонованого рішення з точки зору зручності користування та впливу на внутрішні процеси компанії;

– провести тестування системи та аналіз її ефективності з точки зору користувацького досвіду та продуктивності.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАНЬ НА КВАЛІФІКАЦІЙНУ РОБОТУ

1.1 Аналіз сучасного стану проблеми

Сучасний будівельний ринок України перебуває у стані динамічного зростання, проте супроводжується низкою викликів, які обумовлюють необхідність впровадження інноваційних інструментів управління. Зокрема, у 2024 році обсяг ринку зріс на 6 % порівняно з 2023 роком, досягнувши 170 млрд. грн. [1]. Найбільш активними сегментами стали складські (+111 %) та торгові об’єкти (+22 %), тоді як ринок офісної нерухомості стикнувся з вакантністю на рівні 25 %, що змусило девелоперів переглянути стратегії розвитку [2]. Така динаміка свідчить про потребу в гнучких управлінських рішеннях, здатних адаптуватися до змін структури попиту.

Впровадження програми «єОселя» сприяло зростанню попиту на житлову нерухомість (+12 % у 2024 році), причому третина кредитів припала на новобудови [3]. Однак первинний ринок залишається нестабільним через фінансові та безпекові ризики. Для будівельних компаній це означає необхідність швидкої адаптації до коливань попиту, де CRM-система може забезпечити оперативний аналіз замовлень і прогнозування трендів.

На сьогоднішній день існує низка готових CRM-рішень, що пропонують інструменти для управління бізнес-процесами в будівельній галузі. Серед найпоширеніших – Zoho CRM, Buildertrend та PlanRadar. Проте більшість із них мають низку обмежень щодо адаптації до українського ринку та специфіки локального будівництва.

Наприклад, Zoho CRM, попри широкі можливості автоматизації продажів і звітності, не враховує технічні аспекти будівельних процесів, таких як облік етапів зведення об’єктів, інтеграція з геоданими або контроль підрядників у режимі реального часу [4].

У свою чергу, Buildertrend, хоч і спеціалізується на будівництві, зосереджений переважно на ринку США і не підтримує україномовний інтерфейс, локальні нормативи чи гнучкі розрахунки кошторису відповідно до українських стандартів, а також має високу вартість ліцензування – понад 399 доларів на місяць [5].

PlanRadar пропонує корисні геолокаційні функції та зручну мобільну інтеграцію, однак можливість гнучкого налаштування під специфіку конкретного будівельного підприємства обмежена, а внесення змін у функціонал вимагає втручання розробників [6].

Таким чином, існуючі системи або занадто складні для впровадження в умовах малого та середнього будівельного бізнесу, або не відповідають практичним потребам середньостатистичної української компанії. Це створює передумови для розробки власного адаптивного рішення, орієнтованого на реальні потреби підприємства, включаючи локалізацію, спрощений інтерфейс, підтримку українських стандартів та оптимізацію витрат.

Для розробки CRM-системи з функціями комунікації та моніторингу ефективності працівників з використанням Laravel та інтеграцією Google API було обрано сучасний технологічний стек, в основі якого – фреймворк Laravel на мові програмування PHP. Завдяки архітектурі MVC (Model–View–Controller) Laravel забезпечує гнучку, масштабовану і безпечну структуру додатку, дозволяючи ефективно розділяти бізнес-логіку, інтерфейс користувача та роботу з базою даних [7].

Інтерфейс користувача створюється з використанням шаблонізатора Blade, що входить до складу Laravel. Він дозволяє швидко й зручно розробляти адаптивні HTML-сторінки із вбудованою PHP-логікою. Для інтеграції геолокаційної функціональності в систему використовується Google Maps API, що дозволяє відображати розташування об’єктів і будувати маршрути для клієнтів [8].

Як система управління базами даних використовується PostgreSQL – надійне, продуктивне рішення з підтримкою складних запитів, транзакцій і

масштабування. Робота з базою даних також здійснюється через графічний клієнт Beaver, що полегшує адміністрування і аналіз структури даних [9].

Розробка та розгортання середовища виконуються за допомогою Docker та Docker Desktop, що забезпечує ізольоване і передбачуване середовище для кожного сервісу (Laravel, PostgreSQL, Nginx тощо), а також спрощує перенесення проєкту між різними машинами. Збірка і конфігурація контейнерів виконуються через docker–compose [10].

Для розробки програмного забезпечення обрано середовище Visual Studio Code (VS Code), яке є одним із найпопулярніших редакторів коду завдяки широким можливостям налаштування та підтримці численних розширень [11]. Для ефективної розробки на фреймворку Laravel використовуються спеціалізовані розширення, що допомагають автоматизувати створення маршрутів, моделей і контролерів, а також спрощують навігацію по структурі проєкту [12].

Для роботи з мовою PHP застосовується розширення PHP Intelephense, яке забезпечує підсвітку синтаксису, автодоповнення коду, виявлення помилок та інтеграцію з дебагером, що підвищує продуктивність розробника [13]. Для зручного редагування шаблонів Blade використовується розширення Laravel Blade Spacer, яке покращує читабельність HTML-шаблонів із вбудованими PHP-компонентами [14].

Крім того, для роботи з контейнерами Docker застосовується відповідне розширення, що дозволяє запускати, налаштовувати та контролювати Docker-контейнери безпосередньо з інтерфейсу VS Code, що суттєво спрощує процес розгортання проєкту в ізольованому середовищі [15].

1.2 Постановка завдання на кваліфікаційну роботу бакалавра

Для досягнення поставленої мети розробки CRM-системи з функціями комунікації та моніторингу ефективності працівників з використанням Laravel та інтеграцією Google API були сформовані такі завдання:

– проаналізувати сучасні проблеми у сфері комунікації між клієнтами та будівельними компаніями;

– дослідити вимоги до майбутньої системи спеціалізованої веб-програми та виконати її проєктування;

–розробити UML-діаграми, які проілюструють її логіку і створити дизайн сторінок;

– спроєктувати архітектуру додатку, реалізувати функціонал дистанційного оформлення замовлень, призначення завдань та контролю за їх виконанням;

– оцінити ефективність запропонованого рішення з точки зору зручності користування та впливу на внутрішні процеси компанії;

– провести тестування системи та аналіз її ефективності з точки зору користувацького досвіду та продуктивності.

Висновки до розділу 1

Сучасний будівельний ринок України, незважаючи на динамічне зростання (6 % у 2024 році), стикається із структурними викликами: різке збільшення попиту на складські та торгові об’єкти супроводжується високою вакантністю в офісному сегменті та нестабільністю житлового ринку, що вимагає від компаній гнучкості та інноваційних інструментів управління. Розробка CRM-системи з функціями комунікації та моніторингу ефективності працівників з використанням Laravel та інтеграцією Google API спрямована на вирішення цих проблем через автоматизацію взаємодії з клієнтами, оперативний аналіз даних, масштабованість та захист інформації. Система інтегрує інтуїтивний інтерфейс для оформлення замовлень, механізми модерації, комунікаційні інструменти та тестування продуктивності, що дозволяє компанії адаптуватися до коливань попиту, оптимізувати ресурси та підвищити конкурентоспроможність на тлі змін ринкових умов.

РОЗДІЛ 2
СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ

2.1 Аналіз, визначення вимог до розроблюваного програмного забезпечення та проектування програмного забезпечення

У ході виконання аналізу та визначення вимог до розроблюваного програмного забезпечення було сформульовано концепцію веб-застосунку, головною метою якого є ефективна взаємодія клієнта з будівельною фірмою через простий та інтуїтивно зрозумілий інтерфейс для подачі індивідуальних завдань конкретному працівнику. Було досліджено наявні аналоги подібного програмного забезпечення та виявлено, що більшість існуючих рішень не надає користувачеві можливості ознайомитися з кращими працівниками ще до реєстрації, а також не реалізує повноцінний механізм погодження ціни за послугу з адміністратором.

Сформовані вимоги до системи охоплюють увесь цикл взаємодії між клієнтом, працівником та адміністратором. При відкритті головної сторінки сайту користувач одразу бачить перелік працівників фірми із зазначенням їх імені, посади та досвід роботи. Такий підхід дозволяє клієнтові прийняти зважене рішення щодо вибору виконавця ще до початку процесу замовлення. Після авторизації або реєстрації користувач має можливість вибрати конкретного працівника і заповнити бланк, у якому зазначає опис завдання та бажаний термін його виконання. Далі сформоване замовлення надходить адміністратору, який бачить, якому саме працівникові воно було призначене, після чого визначає та вносить вартість послуги.

У розроблюваному застосунку передбачено погодження ціни з клієнтом: той може або прийняти запропоновану суму, або відмовитися від замовлення. У разі згоди адміністрація забезпечує виконання поставленого завдання, а після його завершення клієнт отримує сповіщення про виконання. Така структура дозволяє забезпечити прозорість, контрольованість і зручність бізнес–процесу, уникаючи посередництва та непорозумінь між сторонами.

Особливу увагу в розробці було приділено візуальному оформленню інтерфейсу. Обрано мінімалістичний стиль із переважанням білого фону, що дозволяє створити візуально приємне середовище, вільне від надмірної графіки чи інформаційного перевантаження. Це сприяє швидкому засвоєнню логіки навігації новими користувачами та покращенню загального користувацького досвіду. Простота і лаконічність дизайну підтримують загальну ідеологію сервісу – зосередженість на головному функціоналі без відволікаючих елементів.

Окрім функціональних вимог, було визначено також низку нефункціональних, що стосуються швидкодії, безпеки, стійкості до збоїв, адаптивності до мобільних пристроїв, зручності масштабування, підтримки багатокористувацького режиму, захисту персональних даних та надійності механізмів авторизації.

Було також сформовано діаграму варіантів використання, що наведена нище (рис. 2.1), дозволила наочно відобразити взаємодію між основними учасниками процесу: гостем, клієнтом, і адміністратором.

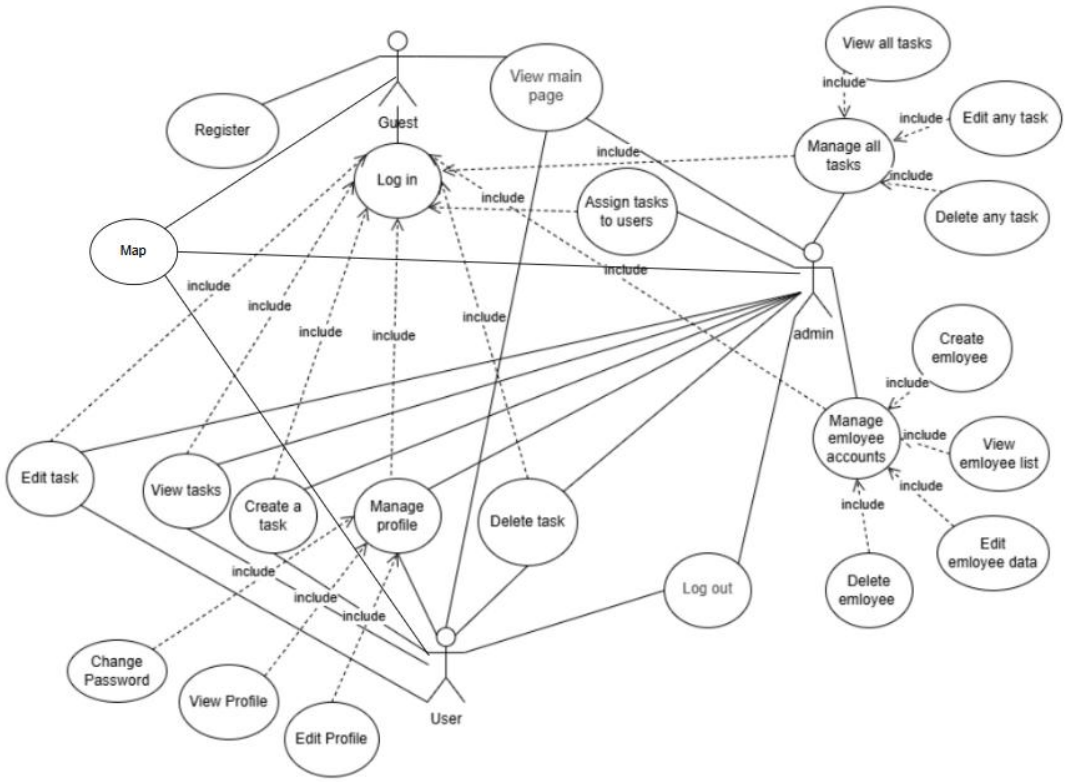

Рисунок 2.1 – Схема використання платформи для усіх типів користувачів

На її основі були сформовані основні сценарії взаємодії, які надалі будуть реалізовані в системі. Проведений аналіз дозволив сформулювати чітке бачення майбутнього веб-застосунку, визначити ключові функціональні компоненти та задати напрям подальшої розробки. Запропоновані рішення спрямовані на оптимізацію процесу замовлення послуг у будівельній сфері, забезпечення ефективної взаємодії між усіма учасниками процесу та створення зручного, інтуїтивного, безпечного й сучасного інтерфейсу для кінцевого користувача.

Деякі елементи інтерфейсу активуються лише після виконання певних дій користувача, оскільки вони логічно пов’язані між собою. Такий підхід забезпечує послідовну та передбачувану роботу системи. Повний опис елементів наведено в таблиці 2.1.

Таблиця 2.1 – Опис об’єктів діаграми

Назва об’єкта	Опис
Guest	Користувач без авторизації. Має доступ лише до базових функцій
User	Авторизований працівник. Може керувати своїми завданнями та профілем
Admin	Успадковує права Користувача. Додатково керує обліковками та завданнями
View main page	Перегляд стартової сторінки системи (без авторизації)
Register	Створення нового облікового запису
Log in	Авторизація в системі (обов’язкова для всіх функцій, окрім гостей)
Log out	Завершення сеансу роботи
Manage profile	Керування користувачем своїм профілем
View profile	Перегляд своїх персональних даних
Edit profile	Зміна імені, пошти або інших даних
Change password	Оновлення паролю облікового запису
Create a task	Додавання нового завдання до системи (для себе)
View tasks	Список усіх активних та завершених завдань користувача
Edit task	Зміна назви, опису, статусу або дедлайну завдання
Delete task	Видалення завдання з системи
Manage employee accounts	Керувати обліковими записами працівників
Create employee	Додавання нового працівника до системи
View employee list	Відображення всіх зареєстрованих працівників
Edit employee details	Зміна даних будь–якого працівника
Delete employee	Видалення облікового запису працівника
Manage all tasks	Керувати всіма завданнями
View all tasks	Список усіх завдань у системі (включаючи інших користувачів)
Edit any task	Повний контроль над будь–яким завданням

Продовження таблиці 2.1

Назва об’єкта	Опис
Edit any task	Повний контроль над будь–яким завданням
Delete any task	Видалення завдання незалежно від автора
Assign tasks to users	Розподіл завдань між працівниками (доступно тільки адміністратору)
Map	Карта до головного офісу для усіх користувачів

Розглянемо архітектуру додатку. На рисунку 2.2 зображено додаток, який складається з клієнтської та серверної частини.

Після створення діаграми використання платформи наступним етапом стало формування карти сайту, яка відіграє важливу роль у процесі проєктування структури веб-застосунку. Карта сайту є схематичним зображенням, що демонструє логічну структуру системи, відображаючи всі сторінки, до яких може мати доступ користувач, а також маршрути переходів між цими сторінками. Така візуалізація особливо корисна під час розробки навігаційної логіки, оскільки дозволяє врахувати всі потенційні сценарії взаємодії з інтерфейсом, уникнути надлишкової складності й забезпечити інтуїтивність користування. З метою точнішого уявлення про особливості взаємодії з системою для різних типів користувачів було розроблено три окремі варіанти карти сайту: для клієнта та адміністратора.

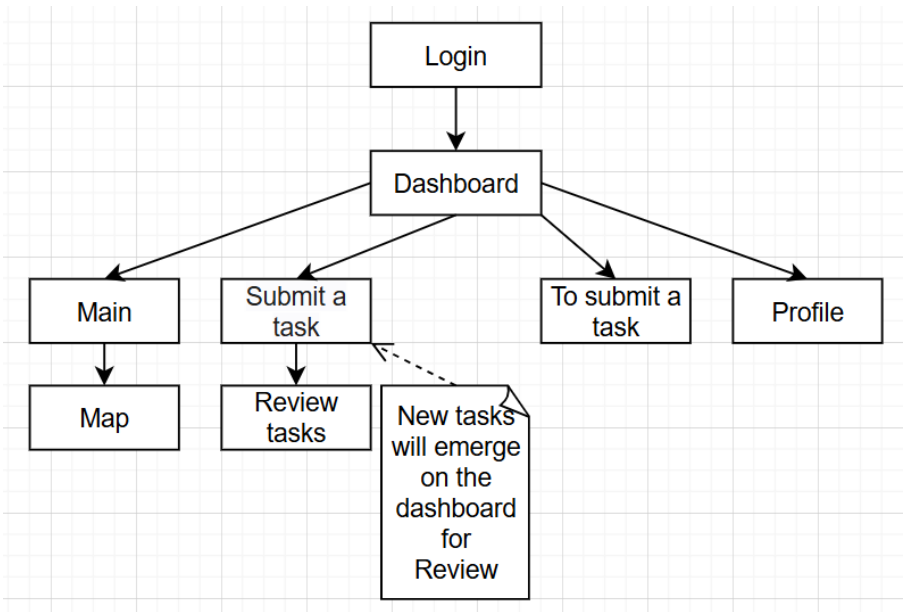

Рисунок 2.2 – Карта сайту клієнта

Першою сторінкою є сторінка авторизації. Після успішної авторизації користувач потрапляє на панель керування, та звідки він може переходити на інші сторінки, такі як профіль, головну сторінку з картою, сторінка вже створеними завданнями або на сторінку подачі завдання. Такий поетапний доступ дозволяє направляти увагу користувача на виконання конкретної дії у потрібний момент, уникаючи зайвої плутанини та створюючи максимально комфортний користувацький досвід. Інтерфейс стає зрозумілим навіть для нових користувачів, оскільки кожен елемент з’являється саме тоді, коли він дійсно необхідний для подальшого взаємодії з системою. Водночас така структура підтримує принципи інтуїтивності та логічної послідовності у побудові користувацького шляху.

Більш детальний опис функціональних можливостей сторінок, а також зв’язків між ними, наведено у таблиці 2.2, яка демонструє об’єкти карти сайту та специфіку їхньої взаємодії.

Таблиця 2.2 – Опис об’єктів карти сайту

Назва об’єкта	Опис
Login	Сторінка авторизації користувача є основним пунктом входу, яка забезпечує захист платформи та визначає рівень доступу, щоб надати користувачу відповідні повноваження та функціональні можливості
Dashboard	Сторінка виконує роль центрального вузла, звідки користувач може перейти переходити на інші сторінки, такі як профіль, головну сторінку з картою, сторінка вже створеними завданнями або на сторінку подачі завдання
Main	Головна сторінка, з можливістю зразу подати задання працівникам, частково переглянути їх список та побачити місце розташування головного офісу
Map	Сторінка з можливістю надати маршрут до головного офісу фірми, за допомогою Google API
Profile	Сторінка з інформацією про користувача. Є можливість вносити зміни, наприклад, назву, пароль, пошту
Submit a task	Сторінка зі списком завдань, переданих працівникам
Review tasks	Сторінка з інформацією про уже створені завдання
To submit a task	Сторінка для створення завдань для працівників

Наступна карта сайту, що розроблена для адміністратора, представлена на рисунку 2.3.

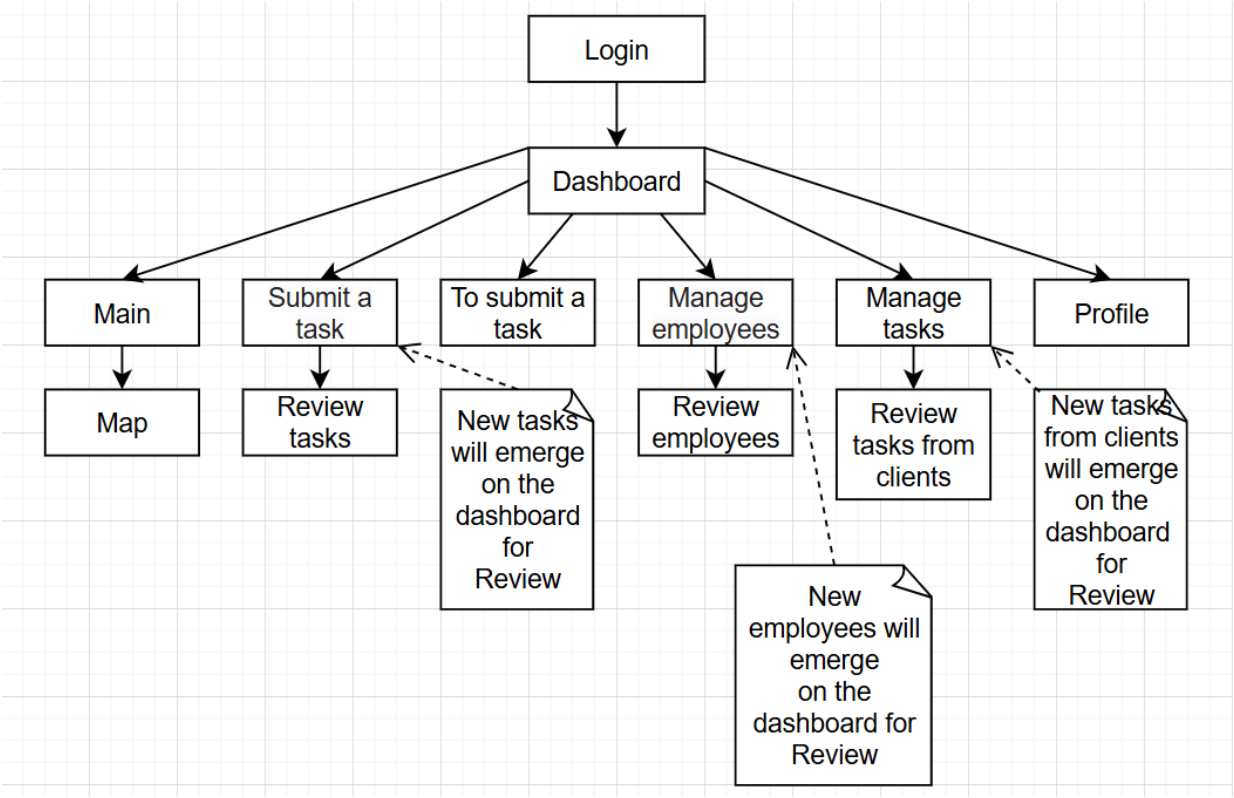

Рисунок 2.3 – Карта сайту адміністратора

Набір сторінок схожий до карти сайту користувача, але крім доступу до звичайних сторінок користувача, адміністратор має можливість відвідати сторінку додавати працівників, а також їх видаляти. Також доступна сторінка для корегування завдання для працівників. У таблиці 2.3 наведений опис об’єктів з рисунку 2.3.

Таблиця 2.3 – Опис об’єктів карти сайту адміністратора

Назва об’єкта	Опис
Login	Сторінка авторизації користувача є основним пунктом входу, яка забезпечує захист платформи та визначає рівень доступу, щоб надати користувачу відповідні повноваження та функціональні можливості
Dashboard	Сторінка виконує роль центрального вузла, звідки користувач може перейти переходити на інші сторінки, такі як профіль, головну сторінку з картою, сторінка вже створеними завданнями або на сторінку подачі завдання
Main	Головна сторінка, з можливістю зразу подати задання працівникам, частково переглянути їх список та побачити місце розташування головного офісу
Map	Сторінка з можливістю надати маршрут до головного офісу фірми, за допомогою Google API
Profile	Сторінка з інформацією про користувача. Є можливість вносити зміни, наприклад, назву, пароль, пошту

Продовження таблиці 2.3

Назва об’єкта	Опис
Submit a task	Сторінка зі списком завдань, переданих працівникам
Review tasks	Сторінка з інформацією про уже створені завдання
To submit a task	Сторінка для створення завдань для працівників
Manage employees	Сторінка для створення працівників
Review employees	Сторінка з інформацією про створених працівників
Manage tasks	Сторінка для редагування завдань клієнтів
Review tasks from clients	Сторінка з інформацією про уже створені завдання від клієнтів до адміністратора

Після детального аналізу функціональних та інформаційних потреб платформи було встановлено, що структура бази даних повинна охоплювати всі ключові аспекти діяльності системи. Зокрема, до бази даних необхідно включити персональні дані користувачів, облікові записи для забезпечення процесу автентифікації, інформацію про завдання, які формуються користувачами для виконання працівниками, а також детальні дані про самих співробітників, що беруть участь у виконанні робіт. Крім того, передбачено збереження структурованих списків завдань, виконавців, статусів завдань, історії їх змін та інших супутніх даних, які є важливими для повноцінного функціонування платформи.

На основі зібраних вимог було виокремлено основні логічні сутності, що репрезентуються в базі даних окремими таблицями. На поточному етапі виділено три ключові таблиці: «Користувачі», яка зберігає загальну інформацію про зареєстрованих осіб; «Адміністратори», відповідальні за модерацію, погодження та ціноутворення завдань; та «Завдання», що містить інформацію про конкретні роботи, їх статуси, виконавців, строки виконання та інше.

Ураховуючи всі можливі сценарії взаємодії між об’єктами, можна сформувати наступні логічні зв’язки: «Користувачі – Працівники» (зв’язок типу один до одного, якщо працівник теж є користувачем системи), «Користувач – Завдання» (зв’язок один до багатьох, оскільки один користувач може створювати багато завдань), та «Працівники – Завдання» (також один до багатьох, оскільки один працівник може виконувати декілька завдань). Усі ці взаємозв’язки між сутностями наочно відображені на рисунку 2.4.

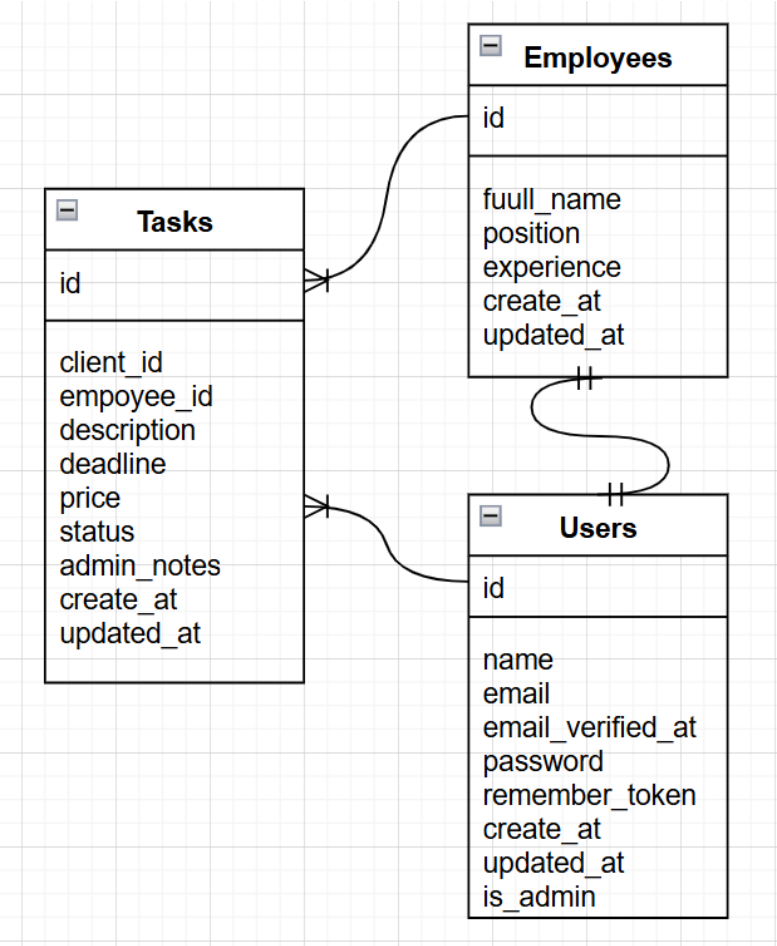

Рисунок 2.4 – Діаграма зв’язків між сутностями в базі даних

2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання

При розробці CRM-системи з функціями комунікації та моніторингу ефективності працівників було прийнято рішення використовувати сучасні та надійні інструменти, що відповідають актуальним вимогам до розробки корпоративного ПЗ. Основним фреймворком обрано Laravel, написаний мовою PHP. Цей вибір обумовлений його модульною архітектурою, наявністю шаблонізатора Blade, системою маршрутизації, ORM Eloquent, підтримкою RESTful API, автентифікації та механізмів захисту. У таблиці 2.4 наведено порівняння трьох популярних фреймворків: Laravel, Symfony та CodeIgniter.

Таблиця 2.4 – Порівняння PHP-фреймворків

Фреймворк	Функціонал	Позитивні сторони	Негативні сторони
Laravel	Сучасний PHP-фреймворк із підтримкою MVC, шаблонізатора Blade, ORM Eloquent, системою маршрутизації, RESTful API, автентифікацією та middleware	Велика спільнота, активна документація, багато вбудованих інструментів, зручність у розробці, підтримка Docker	Може бути повільнішим порівняно з CodeIgniter, потребує більше ресурсів, складніший для початківців
Symfony	Потужний модульний фреймворк, орієнтований на розширення та масштабованість. Використовується для створення складних проєктів	Надійна архітектура, висока гнучкість, використовується у багатьох корпоративних проєктах, підтримка стандартів PSR, повторне використання компонентів	Високий поріг входу, складна структура, більше кодування «вручну», повільніший стартовий проєкт
CodeIgniter	Легкий фреймворк, орієнтований на швидку розробку. Має просту структуру, підходить для невеликих веб-застосунків	Висока продуктивність, мінімальні вимоги до налаштувань, низький поріг входу, швидке розгортання	Менша спільнота, обмежений набір вбудованих функцій, менш сучасний стек, складніше масштабувати для великих проєктів

Для роботи з програмним кодом використовувалося потужне й водночас легке у використанні середовище розробки Visual Studio Code. Воно є одним із найпопулярніших інструментів серед розробників завдяки своїй відкритості, гнучкості й великій кількості доступних розширень. Зокрема, під час розробки CRM-системи були встановлені плагіни для підтримки PHP, шаблонізатора Blade, контейнеризації через Docker, системи контролю версій Git, а також розширення для інтеграції з терміналом і базами даних. Завдяки цьому середовище стало повноцінною платформою для написання, тестування та відлагодження серверного коду, роботи з базою даних, управління версіями й навіть локального запуску контейнеризованих сервісів.

Особливу роль відіграє можливість одночасної роботи над кількома модулями системи, підтримка підсвічування синтаксису, автодоповнення коду, перевірка помилок у реальному часі, а також інтеграція з зовнішніми сервісами.

Використання Visual Studio Code значно прискорило цикл розробки, дозволяючи гнучко реагувати на зміни вимог, проводити швидке тестування функціоналу й покращувати зручність роботи в команді завдяки Git-інтеграції.

У таблиці 2.5 наведено порівняння трьох популярних рішень для розробки веб-застосунків: Visual Studio Code**,** PhpStorm та Sublime Text. Порівняння охоплює такі параметри, як зручність налаштування, продуктивність, підтримка розширень, інтеграція з системами контролю версій і вартість. Це дозволяє обґрунтовано оцінити переваги обраного інструменту в контексті задач, які ставились під час розробки системи

Таблиця 2.5 – Порівняння середовищ розробки

Середовище розробки	Функціонал	Позитивні сторони	Негативні сторони
Visual Studio Code	Легке, розширюване середовище з відкритим кодом, підтримує розширення для PHP, Blade, Docker, Git, інтегрований термінал	Безкоштовне, гнучке, велика кількість розширень, активна спільнота, підтримка багатьох мов і технологій	Менш потужне автодоповнення для PHP у порівнянні з PhpStorm, потребує налаштування для повноцінної PHP-роботи
PhpStorm	Комерційне IDE для PHP від JetBrains, з глибокою інтеграцією з фреймворками, базами даних, тестуванням, дебагером та версткою	Найкраще автодоповнення для PHP, інтегрований дебагер, інспекції коду, підтримка Laravel, Symfony тощо	Платне (є безкоштовна версія для студентів), потребує більше ресурсів, складніше у налаштуванні для новачків
Sublime Text	Легкий та швидкий текстовий редактор із базовою підтримкою PHP, можливістю розширення через плагіни	Надзвичайно швидкий, простий інтерфейс, підтримка багатьох мов, мінімальне навантаження на систему	Обмежений функціонал для PHP із коробки, багато речей потрібно встановлювати вручну, умовно-безкоштовна ліцензія

У якості СУБД використано PostgreSQL – об’єктно-реляційну СУБД, що відзначається високою продуктивністю, підтримкою транзакцій і масштабованістю. Для адміністрування бази застосовувався клієнт DBeaver, що дозволяє створювати структуру БД, переглядати зв’язки й виконувати запити. У таблиці 2.6 наведено порівняльний аналіз трьох популярних СУБД: PostgreSQL,

MySQL та SQLite, з урахуванням їх функціональних можливостей, сильних і слабких сторін.

Таблиця 2.6 – Порівняння систем управління базами даних

СУБД	Функціонал	Позитивні сторони	Негативні сторони
PostgreSQL	Потужна об’єктно-реляційна СУБД із підтримкою складних запитів, транзакцій, JSON, розширень, повнотекстового пошуку	Висока надійність, розширюваність, відповідність стандартам SQL, чудова підтримка транзакцій та складних зв’язків	Потребує більше ресурсів, складніше у налаштуванні для новачків
MySQL	Широко використовувана реляційна СУБД, яка підходить для багатьох веб-застосунків і підтримує реплікацію, індекси, транзакції	Швидкодія, простота встановлення, підтримка великої кількості хостингів, активна спільнота	Менш суворе дотримання SQL-стандартів, обмежена робота з JSON, слабші можливості складних транзакцій
SQLite	Вбудована легка СУБД, яка не потребує окремого сервера; ідеально підходить для невеликих проєктів, мобільних або десктопних додатків	Простота використання, мінімальні ресурси, немає потреби в установці сервера, чудова для локального тестування та прототипів	Не підходить для великих навантажень, відсутність багатокористувацького доступу, обмежена масштабованість

Для забезпечення однакового середовища розробки та продакшну використовувався Docker, а для керування – Docker Desktop. Завдяки цьому стало можливим створення ізольованого середовища для всіх компонентів (PHP, PostgreSQL, веб-сервер), що значно спростило розгортання, масштабування та тестування У таблиці 2.7 наведено порівняння Docker, Vagrant та LXC за функціональними можливостями, перевагами та недоліками.

Таблиця 2.7 – Порівняння засобів контейнеризації

Інструмент	Функціонал	Позитивні сторони	Негативні сторони
Docker	Платформа для автоматизації розгортання застосунків у контейнерах з підтримкою мереж, томів, Dockerfile, образів та оркестрації	Швидкий запуск контейнерів, велика екосистема, підтримка Docker Hub, легка масштабованість, проста інтеграція у CI/CD	Вимагає знання архітектури контейнерів, потребує додаткового захисту для продакшн

Продовження таблиці 2.7

Інструмент	Функціонал	Позитивні сторони	Негативні сторони
Vagrant	Засіб для створення й керування віртуальними машинами (переважно через VirtualBox, VMware, тощо), орієнтований на середовище розробки	Простий у використанні для відтворення повноцінного середовища, підтримка різних провайдерів, зручний для спільної розробки	Повільніше за Docker, потребує більше ресурсів, запускає повноцінні ОС замість легких контейнерів
LXC	Linux Containers – низькорівнева технологія контейнеризації, яка забезпечує ізоляцію процесів через ядро Linux	Більш гнучкий контроль над ресурсами, ближче до традиційної віртуалізації, ефективна робота з системами Linux	Складніший у налаштуванні та керуванні, слабша підтримка кросплатформеності, менша екосистема ніж у Docker

Для реалізації геолокаційної функціональності в системі було обрано Google Maps API – потужний інструмент, який надає широкий спектр можливостей для інтеграції картографічних сервісів у вебзастосунки. Завдяки використанню цього API стало можливим не лише відображати будівельні об’єкти на інтерактивній карті, а й точно визначати їхні координати, створювати мітки з додатковою інформацією, прокладати маршрути доставки матеріалів, а також візуалізувати логістику та етапи планування будівельних робіт. Це значно покращує комунікацію між працівниками, оптимізує пересування техніки на будівельних майданчиках та сприяє загальному підвищенню ефективності проєктного менеджменту.

З метою обґрунтування вибору саме Google Maps API, було проведено порівняльний аналіз трьох найбільш популярних рішень у сфері картографії: Google Maps API, Leaflet.js та OpenStreetMap. У таблиці 2.8 наведено порівняння цих технологій за основними функціональними характеристиками, перевагами, недоліками, умовами використання та адаптивністю до потреб будівельної галузі. Аналіз враховує такі критерії, як точність геолокації, підтримка мобільних пристроїв, гнучкість у налаштуванні, вартість використання, а також простота інтеграції в середовище розробки, що застосовується в проєкті.

Таблиця 2.8 – Порівняння картографічних сервісів

Інструмент	Функціонал	Позитивні сторони	Негативні сторони
Google Maps API	Пропрієтарний сервіс для інтеграції карт, маршрутів, геокодування та інших геолокаційних функцій у веб- та мобільні додатки	Висока точність, широкий функціонал (маршрути, геокодування, Street View), стабільність, підтримка, велика кількість прикладів	Платний після перевищення ліміту безкоштовних запитів, залежність від стороннього сервісу Google
Leaflet.js	Легка та гнучка JavaScript-бібліотека для відображення інтерактивних карт на основі тайлів (найчастіше з OpenStreetMap)	Відкритий код, легка інтеграція, хороша продуктивність, підтримка багатьох плагінів, необмежене використання	Менш функціональний без сторонніх плагінів, вимагає самостійної реалізації частини логіки (напр. геокодування)
OpenStreetMap	Відкрита база географічних даних, яку можна використовувати як джерело карт та для візуалізації з допомогою Leaflet, Mapbox тощо	Безкоштовність, постійно оновлювана спільнотою, можливість використання офлайн, гнучке налаштування вигляду карти	Не має готового API рівня Google Maps, потребує додаткових засобів для інтеграції та сервісів геокодування

Вибір інструментів для розробки CRM-системи був продиктований необхідністю забезпечення надійності, масштабованості та зручності в обслуговуванні програмного продукту. Застосування Laravel, PostgreSQL, Docker, Visual Studio Code та Google Maps API дозволило реалізувати функціонально насичене, сучасне та гнучке рішення. Проведене порівняння альтернативних технологій у відповідних таблицях підтверджує доцільність обраних рішень, оскільки вони найкраще відповідають вимогам до корпоративних систем, зокрема щодо безпеки, розширюваності, швидкодії та інтеграційних можливостей.

Висновки до розділу 2

У результаті проведеного аналізу було сформовано чітке бачення структури, функціональності та візуального оформлення веб-застосунку для будівельної фірми. Визначені вимоги охоплюють як функціональні, так і

нефункціональні аспекти, що забезпечує комплексний підхід до розробки системи. Продумана взаємодія між користувачем, працівником і адміністратором дозволяє оптимізувати процес замовлення послуг, зробити його прозорим і контрольованим. Ретельне проектування інтерфейсу відповідно до сучасних UX/UI принципів, а також розробка діаграм і карти сайту, забезпечують логічну та інтуїтивно зрозумілу навігацію. Усе це створює надійну основу для реалізації зручного, ефективного та сучасного веб-застосунку, що орієнтований на потреби кінцевого користувача.

РОЗДІЛ 3
РОЗРОБКА CRM-СИСТЕМИ З ФУНКЦІЯМИ КОМУНІКАЦІЇ ТА МОНІТОРИНГУ ЕФЕКТИВНОСТІ ПРАЦІВНИКІВ З ВИКОРИСТАННЯМ LARAVEL ТА ІНТЕГРАЦІЄЮ GOOGLE API

3.1 Практична реалізація об’єкта проектування

Розробка практичної частини проекту включала створення як серверного, так і клієнтського компонентів системи. На першому етапі було розпочато роботу над серверною частиною. Серверна розробка традиційно починається з проектування моделей даних, які відповідають за взаємодію з базою даних. Моделі було створено на основі ER-діаграми, що відображає зв’язки між сутностями в БД. На рисунку 3.1 показано організацію файлової структури, де зберігаються всі моделі даних.

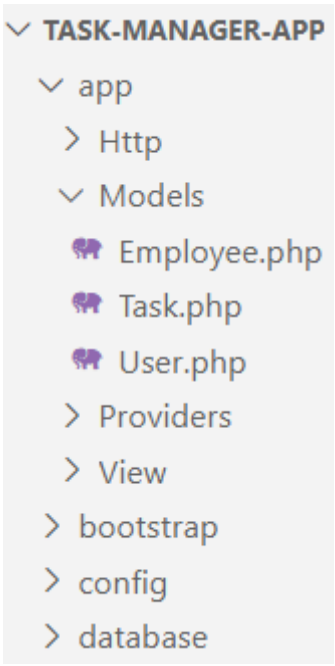

Рисунок 3.1 – Структура папок файлів моделей

Як приклад моделі даних, розглянемо модель Employee (ліст. 3.1), яка представляє сутність «Співробітник» у системі. Модель успадковує клас Illuminate\Database\Eloquent\Model фреймворку Laravel, що надає зручні методи для роботи з відповідною таблицею в базі даних (за замовчуванням, це

таблиця employees). Властивість $fillable визначає, які атрибути моделі можуть бути масово присвоєні, що є важливим для безпеки при обробці даних, що надходять з форм.

Лістинг 3.1 – Код моделі Employee.php

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Employee extends Model
{
use HasFactory;

/**
* The attributes that are mass assignable.
*
* @var array<int, string>
*/
protected $fillable = [
'full_name',
'position',
'experience',
// Інші поля, якщо є
];

// Тут можуть бути визначені зв'язки з іншими моделями,
// наприклад, зв'язок "один до багатьох" із завданнями:
// public function tasks()
// {
//     return $this->hasMany(Task::class);
// }
}
```

кінець лістингу 3.1

Після визначення моделей даних наступним кроком стала розробка контролерів. Контролери є ключовим елементом архітектури MVC (Model-View-Controller) і відповідають за обробку HTTP-запитів, що надходять від клієнта. Вони взаємодіють з моделями для отримання або модифікації даних, реалізують основну бізнес-логіку додатку та повертають відповідь. Як приклад, розглянемо

фрагмент контролера WelcomeController, який відповідає за відображення головної сторінки та передачу даних про співробітників у відповідний шаблон (ліст. 3.2). Метод index цього контролера отримує всіх співробітників з бази даних за допомогою моделі Employee (Employee::all()) та передає їх у шаблон welcome через функцію view().

Лістинг 3.2 – Фрагмент коду контролера WelcomeController.php

```
<?php

namespace App\Http\Controllers;

use App\Models\Employee; // Імпорт моделі Employee
use Illuminate\Http\Request;
use Illuminate\View\View; // Для типізації поверненого значення

class WelcomeController extends Controller
{
/**
* Display the welcome page with a list of employees.
*
* @return \Illuminate\View\View
*/
public function index(): View
{
$employees = Employee::orderBy('full_name')->get(); // Отримати всіх співробітників, відсортованих за іменем
return view('welcome', compact('employees')); // Передати змінну $employees у шаблон welcome
}
}
```

кінець лістингу 3.2

Паралельно з розробкою контролерів визначалися маршрути (routes). Маршрутизація в веб-додатку встановлює відповідність між URL-адресами та конкретними методами контролерів. У фреймворку Laravel файли маршрутів містять декларації всіх доступних ендпоінтів. Наприклад, маршрут для головної сторінки, який викликає метод index контролера WelcomeController, може виглядати так, як показано в лістингу 3.3.

Лістинг 3.3 – Приклад визначення маршруту в routes/web.php

```
<?php

use Illuminate\Support\Facades\Route;
use App\Http\Controllers\WelcomeController; // Імпорт WelcomeController

/*
|--------------------------------------------------------------------------
| Web Routes
|--------------------------------------------------------------------------
|
| Here is where you can register web routes for your application. These
| routes are loaded by the RouteServiceProvider and all of them will
| be assigned to the "web" middleware group. Make something great!
|
*/

Route::get('/', [WelcomeController::class, 'index'])->name('welcome');

// Інші маршрути додатку...
// Auth::routes(); // Маршрути для автентифікації, якщо використовуються стандартні
//  Route::resource('tasks',  TaskController::class)->middleware('auth');
// Приклад маршруту для завдань
```

кінець лістингу 3.3

Для управління структурою бази даних та її еволюцією в часі активно використовувалися міграції. Кожна міграція являє собою PHP-клас, що описує зміни, які необхідно внести до БД. Лістинг 3.4 демонструє приклад міграції для створення таблиці employees, яка відповідає моделі Employee. Метод up описує створення таблиці з необхідними колонками (id, full_name, position, experience та часові мітки timestamps), а метод down – її видалення при відкаті міграції.

Лістинг 3.4 – Приклад файлу міграції для таблиці employees

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;
```

```
return new class extends Migration
{
/**
Run the migrations.
*/
public function up(): void
{
Schema::create('employees', function (Blueprint $table) {
$table->id(); // Первинний ключ, автоінкремент
$table->string('full_name');
$table->string('position');
$table->integer('experience')->default(0); // Роки досвіду
$table->timestamps(); // Колонки created_at та updated_at
});
}

/**
* Reverse the migrations.
*/
public function down(): void
{
Schema::dropIfExists('employees');
}
};
```

кінець лістингу 3.4

Після того, як основні компоненти серверної частини були реалізовані, розпочалася розробка клієнтського інтерфейсу. Для створення користувацького інтерфейсу було використано шаблонізатор Blade. Стилізація сторінок здійснювалася за допомогою CSS, зокрема з використанням Tailwind CSS. Для компіляції та управління клієнтськими ресурсами використовувався інструмент Vite. Як приклад практичної реалізації клієнтської частини, розглянемо код головної сторінки системи (welcome.blade.php), представлений у додатку А. Ця сторінка відображає вітальне повідомлення, список доступних співробітників та надає користувачеві можливість переходу до створення нового завдання або до сторінок автентифікації.

Представлений лістинг демонструє структуру типової сторінки веб-додатку, розробленої з використанням шаблонізатора Blade у фреймворку Laravel. Документ починається зі стандартного оголошення <!DOCTYPE html> та тегу <html> з динамічним визначенням мови. У секції <head> визначені

метатеги, динамічний заголовок сторінки, а також відбувається підключення шрифтів та скомпільованих CSS і JavaScript файлів за допомогою директиви @vite.

Тіло сторінки (<body>) включає загальний навігаційний блок, який підключається директивою @include('layouts.navigation'). Далі йде блок <header>, що містить основний заголовок сторінки, текст якого локалізований. Основний контент сторінки розміщений у тезі <main>. Тут відбувається відображення списку співробітників: спочатку виводиться заголовок секції, а потім за допомогою умовної логіки @if($employees->isEmpty()) перевіряється наявність співробітників. Якщо їх немає, виводиться відповідне повідомлення. В іншому випадку, цикл @foreach ($employees as $employee) ітерує по колекції співробітників (переданої з контролера) і для кожного виводить його ім’я, посаду та досвід роботи у вигляді стилізованих карток. Нижче розташовані елементи керування для користувача, відображення яких залежить від стану автентифікації (@auth та @else). Авторизованим користувачам пропонується створити нове завдання, а неавторизованим – увійти або зареєструватися, з використанням функції route() для генерації відповідних URL. На завершення, у футері (<footer>) сторінки вбудовано карту Google Maps.

Такий підхід до структурування шаблонів з використанням Blade дозволяє створювати гнучкі, динамічні та легко підтримувані користувацькі інтерфейси.

3.2 Розробка бази даних

Розробка структурованої та ефективної бази даних є ключовим елементом для успішного функціонування інформаційно-комп’ютерної системи управління завданнями. Цей процес охоплює вибір системи управління базами даних (СУБД), проектування логічної та фізичної структури даних, а також реалізацію розробленої схеми за допомогою інструментів моделювання та мови SQL.

У межах цього проєкту було обрано реляційну СУБД MySQL. Серед її основних переваг – висока швидкодія, надійність, гнучкість, активна спільнота підтримки та чудова сумісність з фреймворком Laravel, який використовується для реалізації серверної частини системи. MySQL є ефективним рішенням для веб-застосунків, що вимагають стабільної роботи з великими обсягами даних та підтримки складних запитів.

Процес реалізації бази даних розпочався з проектування її структури, зокрема створення концептуальної моделі та побудови фізичних зв’язків між сутностями. Для цього використовувалися спеціалізовані інструменти. MySQL Workbench слугував основним середовищем для побудови ER-діаграм, у яких візуально створювалися таблиці, задавалися їхні поля, типи даних, первинні та зовнішні ключі. Цей інструмент дозволив автоматично генерувати SQL-скрипти, що зручно для подальшого впровадження структури. Для адміністрування бази даних використовувався phpMyAdmin, який надає зручний веб-інтерфейс для перегляду, редагування, тестування запитів і керування привілеями користувачів.

На основі побудованої моделі було створено таблиці, зокрема users, tasks та employees, які є ключовими в системі. Нижче наведено приклад SQL-запиту для створення таблиці users (ліст. 3.5).

Лістинг 3.5 – Приклад модульного тесту для моделі Task

```
CREATE TABLE IF NOT EXISTS `users` (
`id` BIGINT UNSIGNED NOT NULL AUTO_INCREMENT,
`name` VARCHAR(255) NOT NULL,
`email` VARCHAR(255) NOT NULL UNIQUE,
`email_verified_at` TIMESTAMP NULL DEFAULT NULL,
`password` VARCHAR(255) NOT NULL,
`remember_token` VARCHAR(100) NULL DEFAULT NULL,
`created_at` TIMESTAMP NULL DEFAULT CURRENT_TIMESTAMP,
`updated_at`  TIMESTAMP  NULL  DEFAULT  CURRENT_TIMESTAMP  ON  UPDATE
CURRENT_TIMESTAMP,
PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

кінець лістингу 3.5

Аналогічним чином формувалися таблиці для зберігання завдань (ліст. 3.6).

Лістинг 3.6 – Приклад модульного тесту для моделі Task

```
CREATE TABLE IF NOT EXISTS `tasks` (
`id` BIGINT UNSIGNED NOT NULL AUTO_INCREMENT,
`title` VARCHAR(255) NOT NULL,
`description` TEXT NULL,
`status` VARCHAR(50) NOT NULL DEFAULT 'pending', -- наприклад, 'pending',
'in_progress', 'completed'
`user_id` BIGINT UNSIGNED NOT NULL,
`due_date` DATE NULL,
`created_at` TIMESTAMP NULL DEFAULT CURRENT_TIMESTAMP,
`updated_at` TIMESTAMP NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
PRIMARY KEY (`id`),
FOREIGN KEY (`user_id`) REFERENCES `users`(`id`) ON DELETE CASCADE ON
UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

кінець лістингу 3.6

І таблиця для зберігання інформації про працівників, яка також відображається на головній сторінці системи (ліст. 3.7).

Лістинг 3.7 – Приклад модульного тесту для моделі Task

```
CREATE TABLE IF NOT EXISTS `employees` (
`id` BIGINT UNSIGNED NOT NULL AUTO_INCREMENT,
`full_name` VARCHAR(255) NOT NULL,
`position` VARCHAR(255) NOT NULL,
`experience` INT UNSIGNED NOT NULL,
`created_at` TIMESTAMP NULL DEFAULT CURRENT_TIMESTAMP,
`updated_at` TIMESTAMP NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

кінець лістингу 3.7

Після створення структури бази даних почалася активна робота з даними. Додавання, оновлення, видалення та вибірка інформації реалізовувалися за допомогою SQL DML-запитів. Хоча ручне написання SQL-запитів є основою

взаємодії з базою даних, у межах Laravel активно використовуються міграції та ORM Eloquent. Міграції дозволяють автоматизувати створення таблиць у різних середовищах розгортання, а ORM забезпечує об’єктно-орієнтований спосіб доступу до даних. Це значно спрощує підтримку проєкту, однак знання SQL залишається критично важливим для оптимізації запитів, діагностики помилок та вирішення складних завдань, пов’язаних із взаємозв’язками в таблицях.

3.3 Захист інформаційно-комп’ютерної системи

Забезпечення надійного захисту інформаційно-комп’ютерної системи є критично важливим аспектом розробки будь-якого програмного продукту, і система управління завданнями не є винятком. Метою даного розділу є опис комплексу заходів, спрямованих на захист даних користувачів, забезпечення стабільності роботи платформи та запобігання несанкціонованому доступу. Основними потенційними загрозами для системи визначено несанкціонований доступ до облікових записів та даних, витік або модифікація конфіденційної інформації, атаки типу «відмова в обслуговуванні» (DDoS), а також поширені веб-вразливості, такі як SQL-ін’єкції та міжсайтовий скриптинг (XSS). Відповідно, ключовими цілями безпеки є забезпечення конфіденційності, цілісності та доступності даних користувачів та системи в цілому.

Фундаментальним елементом захисту є надійна система аутентифікації та авторизації. Для доступу до системи користувачі повинні пройти процедуру аутентифікації, надавши унікальний логін та пароль. При цьому, для зберігання паролів у базі даних використовується сучасний криптографічний підхід – хешування за допомогою алгоритму bcrypt, що є стандартною практикою у фреймворку Laravel і унеможливлює відновлення оригінального пароля навіть у випадку компрометації бази даних. Після успішної аутентифікації система авторизації визначає рівень доступу користувача до функціоналу та даних. Якщо в системі передбачені різні ролі (наприклад, адміністратор та звичайний

користувач), то розмежування прав доступу реалізовано на основі механізму Role-Based Access Control (RBAC), що гарантує, що кожен користувач може виконувати лише дозволені йому дії та мати доступ лише до відповідної інформації.

Для захисту даних від поширених веб-вразливостей вжито низку заходів. Зокрема, для запобігання SQL-ін’єкціям система активно використовує Eloquent ORM, що вбудований у Laravel, який автоматично параметризує запити до бази даних, унеможливлюючи впровадження шкідливого SQL-коду. Захист від міжсайтового скриптингу (XSS) забезпечується завдяки автоматичному екрануванню всіх даних, що виводяться у шаблонах Blade, що є стандартною поведінкою Laravel і запобігає виконанню довільного JavaScript-коду в браузері користувача. Також система захищена від міжсайтової підробки запитів (CSRF) шляхом використання CSRF-токенів, які Laravel автоматично генерує та перевіряє для всіх POST, PUT, PATCH, DELETE запитів, що ускладнює проведення атак від імені аутентифікованого користувача. Для забезпечення конфіденційності даних під час їх передачі між клієнтським браузером та сервером рекомендується та підтримується використання протоколу HTTPS, який шифрує весь трафік. Щодо збереження даних, розроблена стратегія резервного копіювання, що передбачає регулярне створення копій бази даних та файлів системи, а також процедури для їх оперативного відновлення у випадку збоїв або втрати даних.

Безпека на рівні сервера та інфраструктури також відіграє важливу роль. Це включає регулярне оновлення всього програмного забезпечення, що використовується: операційної системи сервера, веб-сервера (наприклад, Nginx або Apache), інтерпретатора PHP, системи управління базами даних та всіх залежних бібліотек для усунення відомих вразливостей. Для контролю та фільтрації мережевого трафіку використовується брандмауер. Важливим елементом є також система моніторингу та логування подій, яка дозволяє відстежувати стан системи, фіксувати потенційно небезпечні дії та оперативно реагувати на інциденти безпеки.

У підсумку, впроваджений комплекс заходів безпеки, що охоплює надійну аутентифікацію та авторизацію, захист від поширених веб-вразливостей, шифрування даних, регулярне резервне копіювання, а також заходи безпеки на рівні серверної інфраструктури, створює міцну основу для захисту інформаційно-комп'ютерної системи управління завданнями. Ці заходи спрямовані на мінімізацію ризиків та забезпечення безпечного середовища для користувачів платформи, хоча постійний моніторинг та оновлення політик безпеки залишаються важливими для підтримки належного рівня захищеності в майбутньому.

3.4 Тестування та налагодження інформаційно-комп'ютерної системи

Етап тестування та налагодження є невід'ємною та критично важливою частиною життєвого циклу розробки інформаційно-комп'ютерної системи, зокрема платформи управління завданнями. Основна мета цього етапу полягала у виявленні та усуненні помилок, недоліків та вузьких місць у функціональності, продуктивності та безпеці системи, а також у забезпеченні відповідності розробленого продукту початковим вимогам та очікуванням користувачів. Якісне тестування гарантує надійність, стабільність та коректність роботи системи перед її впровадженням та експлуатацією. Процес тестування був багатоетапним та включав різні види перевірок для всебічного охоплення системи. Насамперед проводилося модульне тестування (unit testing), під час якого кожен окремий компонент системи, такий як класи моделей, сервіси, контролери та окремі функції, перевірявся на коректність роботи в ізоляції. Для цього використовувалися стандартні інструменти, такі як PHPUnit, що дозволило автоматизувати перевірку логіки окремих модулів та швидко виявляти помилки на ранніх стадіях розробки. Це забезпечило впевненість у правильності функціонування базових елементів системи. Як приклад модульного тесту, розглянемо тест для перевірки створення об'єкта моделі Task. Такий тест міг би виглядати наступним чином (ліст. 3.8).

Лістинг 3.8 – Приклад модульного тесту для моделі Task

```
<?php

namespace Tests\Unit;

use App\Models\Task; // Припускаємо, що модель Task існує
use App\Models\User; // Припускаємо, що для завдання потрібен користувач
use Illuminate\Foundation\Testing\RefreshDatabase;
use Tests\TestCase;

class TaskTest extends TestCase
{
use RefreshDatabase; // Очищує базу даних перед кожним тестом

/**
* Тест перевіряє можливість створення завдання.
*
* @return void
*/
public function test_task_can_be_created()
{
// Створюємо користувача, якому буде належати завдання
 $user = User::factory()->create();

        $taskData = [
'title' => 'Тестове завдання',
'description' => 'Опис тестового завдання',
'status' => 'pending', // Припустимий статус
'user_id' => $user->id, // Пов'язуємо завдання з користувачем
// Можуть бути інші необхідні поля
];

// Створюємо завдання через модель
$task = Task::create($taskData);

// Перевіряємо, що об'єкт завдання було успішно створено
$this->assertInstanceOf(Task::class, $task);
// Перевіряємо, що назва завдання відповідає наданій
$this->assertEquals($taskData['title'], $task->title);
// Перевіряємо наявність відповідного запису в базі даних
$this->assertDatabaseHas('tasks', [
'title' => 'Тестове завдання',
'user_id' => $user->id
]);
}
}
```

кінець лістингу 3.8

У наведеному лістингу 3.8 клас TaskTest успадковується від базового тестового класу Laravel TestCase. Треит RefreshDatabase забезпечує очищення бази даних перед кожним тестом, що гарантує незалежність тестів один від одного. Метод test_task_can_be_created імітує створення нового завдання, надаючи необхідні дані. За допомогою тверджень (assertions) assertInstanceOf, assertEquals та assertDatabaseHas перевіряється, що об’єкт завдання дійсно створений, його властивості відповідають очікуваним, і відповідний запис з’явився в базі даних. Такі тести дозволяють автоматично контролювати коректність роботи ключових моделей системи. Наступним кроком було інтеграційне тестування, спрямоване на перевірку взаємодії між різними модулями та компонентами системи. На цьому етапі перевірялося, наскільки коректно різні частини програми, такі як взаємодія контролерів з сервісами та моделями, обмін даними з базою даних, та робота API-ендпоінтів, функціонують разом. Це дозволило виявити проблеми, що виникають на стиках компонентів, та забезпечити їх узгоджену роботу.

Ключовим етапом стало функціональне тестування (end-to-end testing), яке проводилося з точки зору кінцевого користувача. Метою було переконатися, що всі функції системи працюють відповідно до специфікацій та бізнес-логіки. Тестувалися основні сценарії використання платформи, включаючи реєстрацію та автентифікацію користувачів, створення, редагування, призначення та відстеження статусу завдань, а також перегляд списку співробітників, як це передбачено на головній сторінці (welcome.blade.php). Частково це тестування проводилося вручну, а також розглядалася можливість використання інструментів автоматизації, таких як Laravel Dusk, для перевірки користувацьких інтерфейсів та потоків. Окрім функціональних аспектів, увага приділялася тестуванню зручності використання (usability testing). Здійснювалася оцінка інтуїтивності навігації, зрозумілості інтерфейсу та загального користувацького досвіду. Збиралися відгуки для виявлення потенційних проблем у взаємодії користувача з системою та вносилися відповідні корективи для покращення ергономіки платформи. Також, хоча

основні аспекти безпеки були розглянуті в попередньому розділі, під час тестування проводилися перевірки на стійкість до базових веб-вразливостей та коректність роботи механізмів авторизації та контролю доступу.

Процес налагодження (debugging) супроводжував усі етапи розробки та тестування. Для ідентифікації та виправлення помилок використовувалися вбудовані інструменти фреймворку Laravel, такі як функція dd() для швидкого виведення та аналізу змінних, фасад Log для ведення журналів подій та помилок, а також інструменти розробника в браузері для налагодження клієнтської частини. У випадках складних помилок застосовувався покроковий налагоджувач, наприклад Xdebug. Кожна виявлена помилка ретельно аналізувалася для визначення її причини, після чого вносилися необхідні виправлення, і проводилося повторне тестування (регресійне тестування) для гарантії, що виправлення не спричинило нових проблем. Тестування проводилося в локальному середовищі розробки, яке максимально наближене до конфігурації робочого сервера.

За результатами проведеного комплексного тестування та налагодження було виявлено та усунуто низку помилок різного ступеня критичності, оптимізовано роботу окремих модулів та покращено загальну стабільність і надійність інформаційно-комп’ютерної системи управління завданнями. Ітеративний підхід до тестування, коли після кожного виправлення чи додавання нового функціоналу проводився повторний цикл перевірок, дозволив досягти високої якості кінцевого продукту та підготувати його до експлуатації. Таким чином, етап тестування та налагодження підтвердив працездатність системи та її відповідність поставленим функціональним вимогам.

3.5 Розробка документації для програмного продукту

Належна документація є невід’ємною частиною будь-якого програмного продукту, оскільки вона забезпечує користувачів та розробників необхідною інформацією для ефективного використання, обслуговування та подальшого

розвитку системи. У рамках даного проекту було розроблено комплект документації, що включає опис системних вимог, довідкові матеріали для користувачів та інструкції з встановлення та розгортання програмного продукту.

Для стабільної та ефективної роботи розробленої системи управління завданнями необхідно забезпечити відповідність певним мінімальним системним вимогам. Щодо серверної частини, потрібна сучасна серверна операційна система, наприклад, Linux дистрибутиви як Ubuntu Server або CentOS, або Windows Server. Веб-сервер має бути Apache 2.4 або новіший, або Nginx 1.18 або новіший, з підтримкою модулів для обробки PHP та перезапису URL. Необхідна версія PHP – 8.1 або новіша, з розширеннями OpenSSL, PDO, Mbstring, Tokenizer, XML, Ctype, JSON, BCMath та Fileinfo. Як система управління базами даних (СУБД) може використовуватися MySQL 5.7 (або MariaDB 10.3) або новіша, або PostgreSQL 10 або новіша; Laravel також підтримує SQLite та SQL Server. Для встановлення та оновлення бібліотек проекту знадобиться остання стабільна версія Composer, менеджера залежностей для PHP. Мінімальний обсяг оперативної пам’яті (RAM) становить 2 ГБ, хоча рекомендовано 4 ГБ або більше для оптимальної продуктивності. Також необхідно щонайменше 1 ГБ вільного дискового простору для файлів додатку, бази даних та журналів.

Вимоги до клієнтської частини, тобто для кінцевих користувачів, включають наявність будь-якого сучасного веб-браузера з підтримкою HTML5, CSS3 та JavaScript, такого як Google Chrome, Mozilla Firefox, Safari або Microsoft Edge останніх версій, а також стабільне підключення до мережі Інтернет. Для середовища розробки, додатково до серверних вимог, знадобиться Node.js версії 16.x або новішої для управління frontend залежностями та компіляції ресурсів за допомогою Vite, а також npm (Node Package Manager) або Yarn для управління JavaScript пакетами та система контролю версій Git для роботи з кодовою базою проекту.

Для забезпечення комфортної роботи користувачів з системою було підготовлено довідкові матеріали, які описують основний функціонал та процеси

взаємодії. Ці матеріали плануються до інтеграції безпосередньо в систему у вигляді онлайн-довідки та підказок, а також можуть бути надані як окремий документ. Основні розділи довідкових матеріалів охоплюють вступ із загальним описом системи та її призначенням. Далі йде розділ про початок роботи, що включає процес реєстрації, входу до системи та огляд головного інтерфейсу. Для авторизованих користувачів детально описано управління завданнями: створення, перегляд, редагування, зміна статусу та видалення завдань. Для всіх користувачів надається інформація про перегляд даних співробітників. Також є розділ, присвячений профілю користувача, де описано редагування особистих даних та зміну паролю, якщо це передбачено функціоналом. На завершення, передбачено розділ з частими питаннями (FAQ) та рекомендаціями щодо вирішення можливих проблем.

Процес встановлення та розгортання системи управління завданнями на сервері є багатоетапним. Спочатку необхідно підготувати сервер, переконавшись, що він відповідає мінімальним системним вимогам та має встановлене все необхідне програмне забезпечення, таке як веб-сервер, PHP, СУБД, Composer, Node.js та npm/yarn. Наступним кроком є отримання коду додатку шляхом клонування репозиторію проекту з системи контролю версій Git. Після цього встановлюються залежності: PHP залежності за допомогою команди composer install --optimize-autoloader --no-dev та JavaScript залежності з наступною компіляцією frontend ресурсів командами npm install та npm run build.

Далі відбувається конфігурація середовища. Потрібно скопіювати файл конфігурації .env.example в .env, згенерувати унікальний ключ додатку командою php artisan key:generate та відредагувати файл .env, вказавши назву додатку, тип середовища (production), вимкнувши режим налагодження (APP_DEBUG=false), вказавши повну URL-адресу додатку та налаштування підключення до бази даних (тип, хост, порт, назва БД, ім’я користувача та пароль), а також, за потреби, налаштування поштового сервера.

Після конфігурації середовища виконуються міграції бази даних командою php artisan migrate --force для створення структури таблиць. За

бажанням, можна заповнити базу даних початковими даними за допомогою сідера. Для коректного доступу до файлів зі сховища створюється символьне посилання командою php artisan storage:link.

Важливим етапом є конфігурація веб-сервера (Apache або Nginx) таким чином, щоб кореневою директорією була папка public проекту Laravel, та надання веб-серверу прав на запис до директорій storage та bootstrap/cache. Наприклад, для Apache це налаштовується у файлі віртуального хоста, вказуючи DocumentRoot та дозволяючи AllowOverride All для директорії public. Для Nginx конфігурація сервера включає визначення root та налаштування обробки PHP-запитів через fastcgi_pass.

Для підвищення продуктивності у продуктивному середовищі рекомендується кешувати конфігурацію, маршрути та шаблони за допомогою команд php artisan config:cache, php artisan route:cache та php artisan view:cache. Якщо додаток використовує заплановані завдання Laravel, необхідно додати відповідний Cron запис на сервер, який буде щохвилини запускати планувальник командою php artisan schedule:run. У випадку використання черг для відкладеного виконання завдань, потрібно налаштувати та запустити воркери черг, наприклад, за допомогою php artisan queue:work, та забезпечити їхню безперервну роботу через інструменти типу Supervisor. Після виконання всіх цих кроків розроблена система буде розгорнута та готова до використання. Регулярне створення резервних копій бази даних та файлів додатку є критично важливим для запобігання втрати даних.

3.6 Розробка дизайну сайту

Під час розробки інтерфейсу веб-застосунку особливу увагу було приділено дизайну, оскільки саме зорове сприйняття сторінки визначає перше враження користувача та його готовність взаємодіяти з системою. Основним принципом, що ліг в основу дизайну, стала концепція мінімалізму, яка передбачає використання простих форм, чіткої структури, великої кількості

білого простору та відсутності зайвих декоративних елементів. Такий підхід дозволяє максимально зосередити увагу користувача на функціональних можливостях додатку та спростити процес навігації.

Візуальне оформлення інтерфейсу було виконано у світлих тонах із домінуванням білого кольору, що забезпечує чистоту і легкість сприйняття сторінки. Білий фон також дозволяє легко поєднувати елементи з різною кольоровою гамою, акцентуючи увагу на найважливіших ділянках, зокрема, кнопках дій, повідомленнях, формах заповнення тощо. Завдяки використанню контрастних акцентів було досягнуто високої видимості основних елементів керування, що є критично важливим для швидкої орієнтації користувача на сторінці.

У дизайні були враховані сучасні принципи UX/UI, які базуються на створенні інтерфейсу, орієнтованого на користувача. Було забезпечено логічну послідовність переходів між сторінками, інтуїтивно зрозуміле розташування функціональних блоків, єдину стилістику шрифтів і кнопок. Головна сторінка дозволяє ще до авторизації ознайомитися з головним працівником фірми, що створює відчуття відкритості та прозорості сервісу Такий елемент структури водночас виконує інформаційну та навігаційну функцію, підштовхуючи користувача до подальших дій.

Особливої уваги було надано формам введення інформації. Було реалізовано зручну та стислу форму для заповнення завдання, де користувач може зазначити опис проблеми або потреби, вказати бажаний дедлайн та обрати працівника. Усі поля мають підказки, щоб навіть новий користувач без досвіду міг легко зрозуміти, яку саме інформацію потрібно ввести. Дизайн цих форм також витриманий у мінімалістичному стилі – відсутність зайвих графічних елементів дозволяє користувачу зосередитися на суті.

Загалом дизайн веб-застосунку є невіддільною складовою його загальної архітектури. Він забезпечує простоту, швидкість, зручність та задоволення від використання сервісу. Завдяки продуманому поєднанню візуальних рішень та логічної структури було досягнуто головної мети – створення інтерфейсу, який

не потребує додаткових пояснень, є інтуїтивно зрозумілим і привабливим для кінцевого користувача. Створений дизайн сприяє формуванню позитивного іміджу компанії та довіри з боку потенційних клієнтів, що є надзвичайно важливим у сфері обслуговування.

Першою з розроблених у рамках проєкту сторінок, для якої було створено повноцінний дизайн, стала сторінка авторизації користувача (рис. 3.2). Ця сторінка виконує ключову роль у процесі входу в систему, тому її оформлення було ретельно продумане з урахуванням зручності та зрозумілості для кінцевого користувача. Вона містить стандартні поля для введення електронної пошти та пароля, які є обов’язковими для автентифікації.

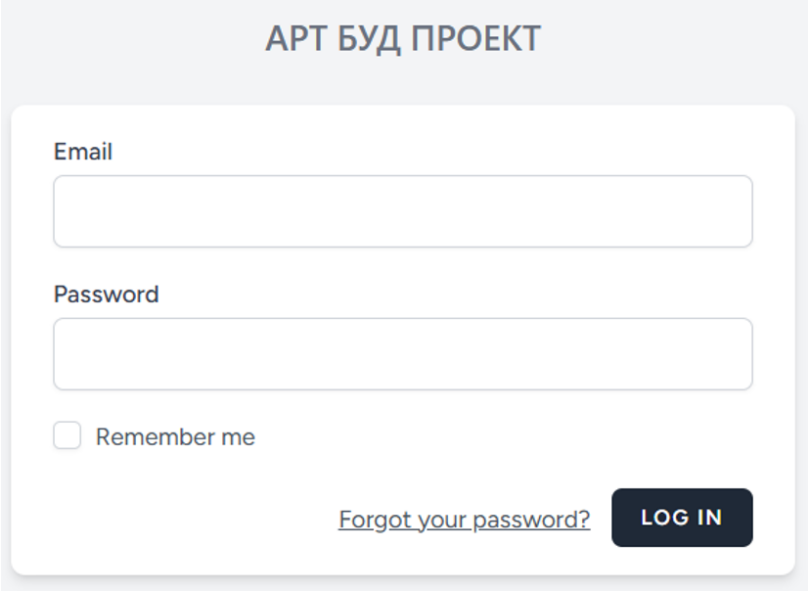

Рисунок 3.2 – Дизайн сторінки авторизації

Однією з ключових сторінок, опрацьованих під час етапу створення дизайну, стала сторінка реєстрації користувача (рис. 3.3). Її оформлення витримано у загальному стилі додатку, що забезпечує візуальну цілісність та єдність інтерфейсу. Водночас, у порівнянні зі сторінкою авторизації, вона має низку особливостей, продиктованих її функціональним призначенням.

Дизайн сторінки реєстрації передбачає чітке розмежування від сторінки входу: були змінені текстові написи, заголовки та підказки у формах, що

допомагає користувачеві точно зрозуміти, де саме він знаходиться. Для уникнення плутанини були вилучені непотрібні елементи, а замість них додано посилання на вхід для тих, хто вже має обліковий запис. Такий підхід зробив інтерфейс більш лаконічним, зрозумілим і зручним для користувачів.

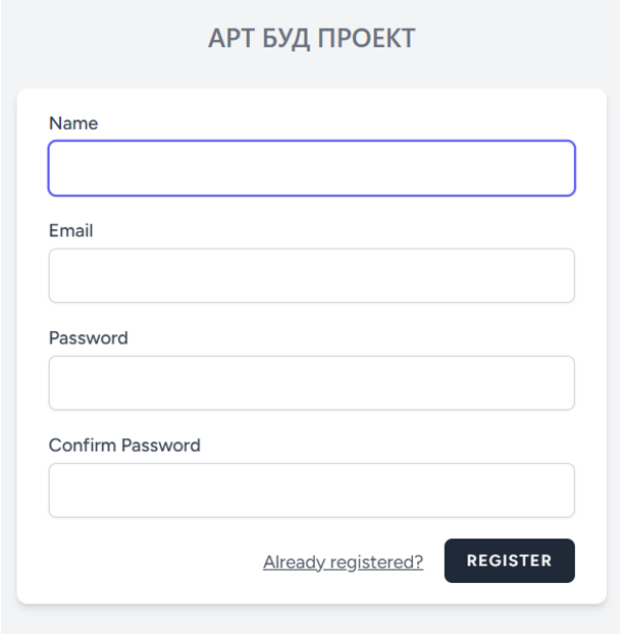

Рисунок 3.3 – Дизайн сторінки реєстрації

Ще однією важливою сторінкою, для якої було створено індивідуальний дизайн, є головна сторінка додатку, що виконує функцію пошуку створення завдання та показу карти до головного офісу (рис. 3.4). Основною метою її створення було забезпечення зручного, швидкого та інтуїтивно зрозумілого способу ознайомлення з частковим списком працівників фірми, тому особливу увагу було приділено структуруванню елементів інтерфейсу.

У верхній частині сторінки було додано заголовок із навігаційними посиланнями, які дозволяють користувачеві перемикатися між іншими сторінками додатку. Крім того, у нижній частині сторінки розміщена карта, де показано розташування головного офісу. Такий підхід забезпечує зручність навігації та комфортного досвіду.

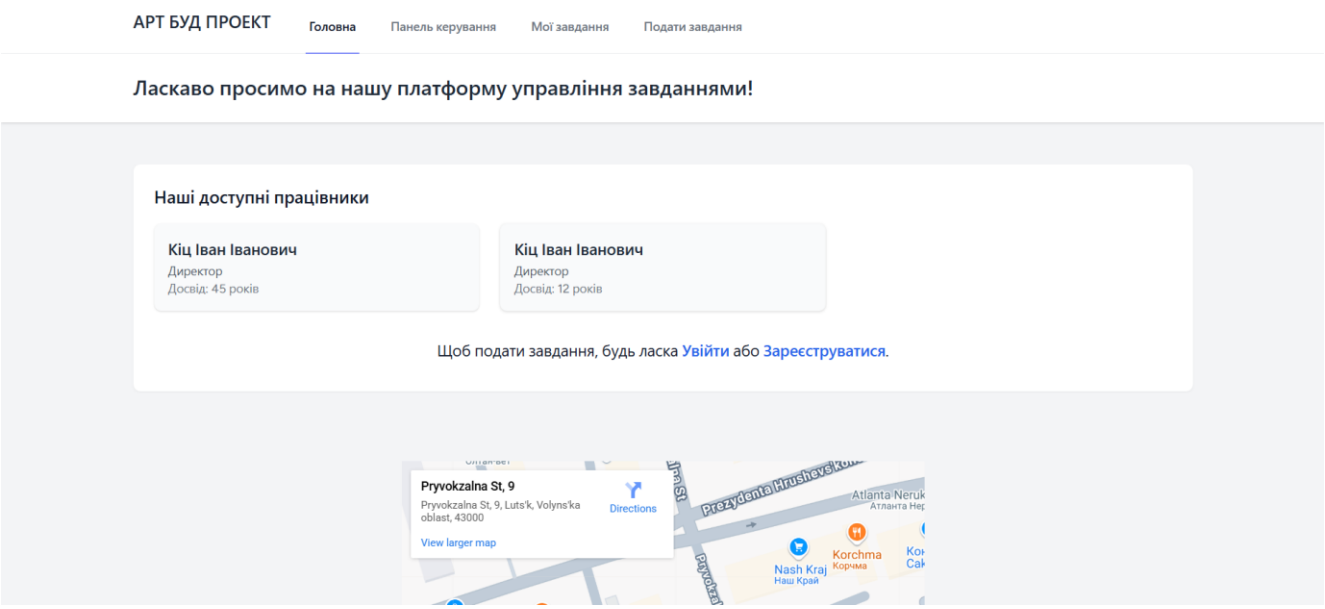

Рисунок 3.4 – Дизайн головної сторінки

Ще однією важливою сторінкою, для якої було розроблено дизайн, є сторінка «Мої завдання». Вона оформлена відповідно до загального стилю веб-додатку, з акцентом на візуальну впорядкованість та зручність сприйняття інформації. Основне завдання дизайну полягало у створенні зрозумілої структури для відображення переліку завдань, яку легко читати та використовувати.

У верхній частині сторінки розташовано інтуїтивно зрозуміле меню навігації, яке дає змогу швидко перемикатися між основними розділами сайту. Завдяки активній вкладці «Мої завдання» користувач завжди розуміє, в якому розділі перебуває.

Для візуального акценту в інтерфейсі застосовано яскраву кнопку створення нового завдання, що вирізняється кольором і займає помітне місце під заголовком сторінки. Основна інформація представлена у вигляді таблиці з чіткою сіткою, де застосовано нейтральне тло для загального вигляду та контрастні кольори для виділення важливих елементів, таких як статус завдання. Такий підхід дозволяє користувачу швидко орієнтуватися в даних і зменшує когнітивне навантаження при взаємодії зі сторінкою (рис. 3.5).

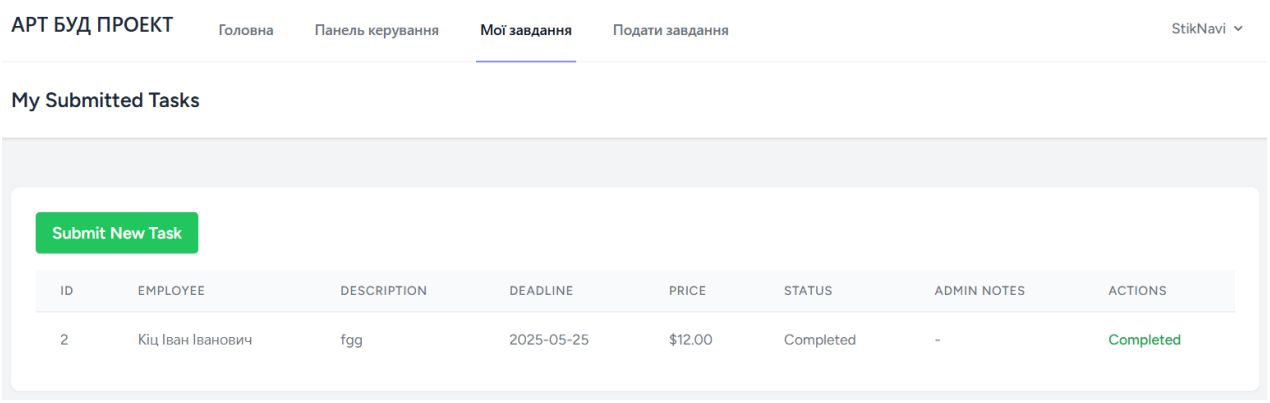

Рисунок 3.5 – Дизайн сторінки зі списком поданих завдань

Серед численних сторінок, які було розроблено в межах інтерфейсу додатку, сторінка створення завдання відіграє одну з ключових ролей, оскільки саме через неї користувачі взаємодіють із основним функціоналом системи. Особливу увагу під час її проєктування було приділено простоті, зрозумілості та ергономіці, що є визначальними чинниками зручної взаємодії користувача з інтерфейсом. Дизайн сторінки створено таким чином, щоб користувач, незалежно від технічної підготовки, міг швидко та безпомилково створити нове завдання, не витрачаючи час на додаткові пояснення чи інструкції.

Інтерфейс сторінки побудований на принципі послідовного заповнення форми, де кожен елемент логічно пов’язаний із попереднім. Всі поля форми мають зрозумілі підписи та структуровані відповідно до загальноприйнятих правил інтерфейсного дизайну: спочатку вводиться назва завдання, далі – опис, термін виконання, виконавець тощо. Це не лише полегшує користувачу орієнтацію, але й забезпечує логічну послідовність дій, зменшуючи когнітивне навантаження.

Важливою особливістю є продумане розташування елементів. Розмітка дозволяє одразу охопити поглядом усі необхідні поля, не змушуючи користувача шукати потрібну інформацію. Візуальний стиль базується на принципах мінімалізму: відсутні зайві декоративні елементи, кольори використовуються стримано, шрифти – читабельні та однакові по всьому інтерфейсу. Це сприяє

зосередженню користувача на основному завданні – введенні необхідної інформації без відволікань.

Крім того, дотримано єдину стилістику з іншими сторінками додатку, що сприяє збереженню цілісного користувацького досвіду. Верхнє меню навігації, яке залишилося незмінним, дозволяє користувачу легко повертатися до інших розділів системи. Колірна гама та типографіка узгоджені з дизайном усього додатку, забезпечуючи візуальну гармонію.

На сторінці (рис. 3.6) також передбачено чітке позначення заголовка, що вказує на її функціональне призначення, та кнопку підтвердження створення завдання, яка спеціально виділена яскравим кольором. Це дозволяє користувачу інтуїтивно розуміти, як завершити процес заповнення форми.

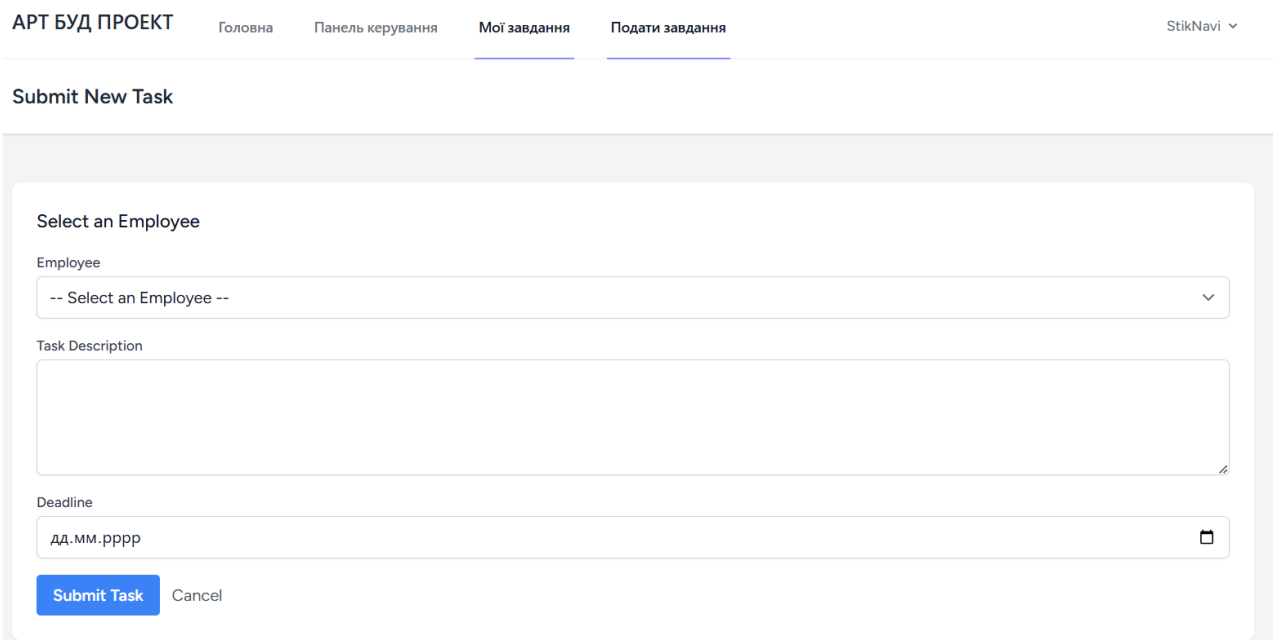

Рисунок 3.6 – Дизайн сторінки для подання завдань

Однією зі сторінок, для якої було створено дизайн у частині адміністративного інтерфейсу, є сторінка створення нового працівника (рис. 3.7). Ця сторінка є важливим інструментом для адміністратора, оскільки дозволяє швидко й зручно додавати до системи нових співробітників, які будуть виконувати завдання, створені користувачами.

Основною метою при розробці дизайну сторінки було забезпечення простоти та ефективності взаємодії адміністратора з формою введення. У верхній частині сторінки розміщено заголовок, який чітко вказує на функціональне призначення інтерфейсу – керувати працівниками. Завдяки цьому адміністратор одразу розуміє, на якій сторінці він знаходиться та яку дію виконує. Також на сторінці показаний список уже створених працівників.

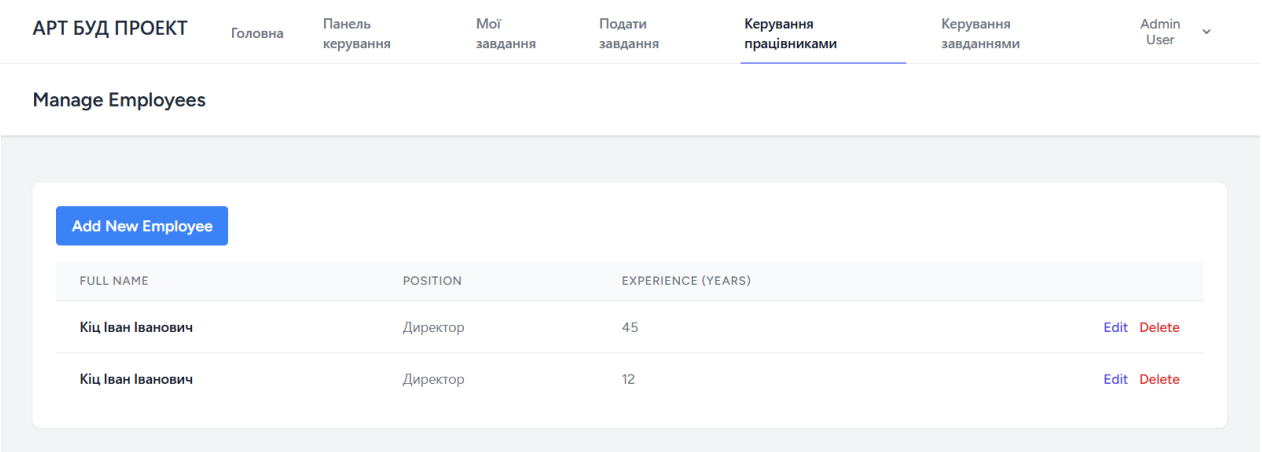

Рисунок 3.7 – Дизайн сторінки для керування працівниками

Центральна частина сторінки містить кнопку під назвою «Add New Empoyee», після натискання якої відкривається форма з полями, необхідними для додавання нового працівника до бази даних (рис. 3.8). Усі поля впорядковані логічно та супроводжуються короткими підписами, що пояснюють, яку саме інформацію потрібно ввести. Зокрема, передбачено поля для імені, прізвища та по–батькові, його позицію на фірмі та стажу його роботи. Поля введення мають стандартний вигляд і розміри, а також добре видимі рамки, що допомагають уникнути помилкового натискання або пропуску.

Кнопка підтвердження, розташована внизу форми, вирізняється контрастним кольором, привертаючи увагу до фінального етапу дії. Після натискання система виконує перевірку коректності введених даних, а в разі помилок надає підказки для їх усунення.

Дизайн сторінки створення працівників орієнтований на забезпечення зручної та зрозумілої роботи адміністратора. Простота структури, мінімалізм у

візуальних елементах та чітке функціональне розділення дозволяють адміністратору ефективно взаємодіяти зі сторінкою навіть при виконанні великого обсягу рутинної роботи.

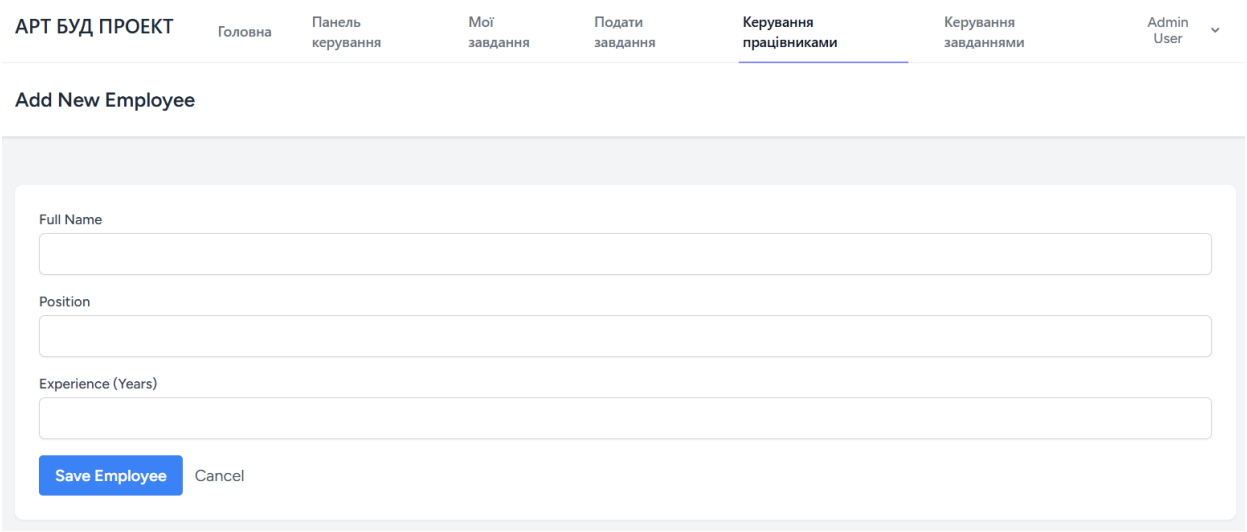

Рисунок 3.8 – Дизайн сторінки для створення працівників

Ще один дизайн сторінки, для адміністратора, є сторінка для керування завданнями (рис. 3.9). Ця сторінка призначена для відображення переліку всіх завдань, які користувач подав для виконання адміністратору. Головна мета цієї сторінки – надати адміністратору зручний спосіб переглядати статус поданих завдань, отримувати інформацію з цих завдань, з чим ця сторінка прекрасно справляється.

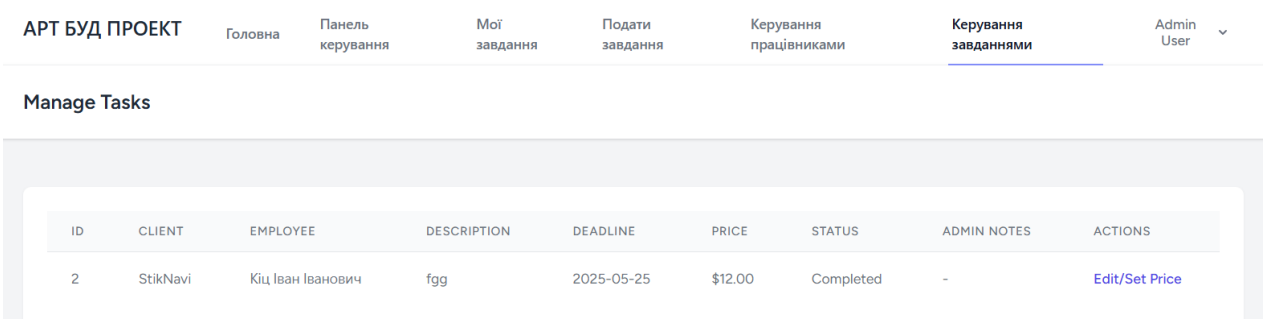

Рисунок 3.9 – Дизайн сторінки для керування працівниками

Висновки до розділу 3

У цьому розділі було представлено процес розробки CRM-системи з функціями комунікації та моніторингу ефективності працівників на основі фреймворку Laravel та інтеграції з Google API. Реалізація охопила як серверну, так і клієнтську частини проєкту. Детально розглянуто побудову моделей, контролерів, маршрутів, структури бази даних та її міграцій. Особливу увагу приділено формуванню користувацького інтерфейсу за допомогою шаблонізатора Blade і бібліотеки Tailwind CSS, що забезпечило зручність та адаптивність дизайну. Інтеграція Google Maps надала можливість візуалізувати місце розташування працівників, підвищивши функціональність системи.

Окремим етапом стала розробка бази даних на основі СУБД MySQL, що продемонструвала ефективність у зберіганні й обробці інформації про користувачів, працівників та завдання. Процес моделювання структури БД з використанням MySQL Workbench та адміністрування через phpMyAdmin дозволив забезпечити узгодженість даних та масштабованість рішення.

Загалом, розроблена система є гнучкою, надійною та зручною у використанні як для менеджменту, так і для працівників компанії, відповідаючи поставленим вимогам і цілям дослідження.

ВИСНОВКИ

У межах виконання кваліфікаційної роботи було реалізовано повний цикл створення CRM-системи для будівельної компанії з функціями комунікації та моніторингу ефективності працівників, зосереджуючись на вирішенні актуальних проблем цифрової взаємодії між клієнтами й виконавцями.

У першому етапі роботи здійснено аналіз проблем у сфері взаємодії будівельних компаній із замовниками. Було виявлено основні труднощі, зокрема відсутність інструментів для оперативного оформлення замовлень, моніторингу їх виконання та призначення виконавців, що знижує загальну ефективність роботи компаній.

На основі аналізу сформульовано вимоги до функціональності й якості майбутньої системи. Визначено основні процеси, які повинна охоплювати CRM-система, а також її здатність масштабуватися та адаптуватися до реальних умов роботи будівельної компанії.

На етапі проєктування було розроблено UML-діаграми, які відображають логіку взаємодії між користувачами, адміністраторами та працівниками. Крім того, створено карту сайту та структуру бази даних із визначенням ключових сутностей і зв’язків між ними.

Також було спроєктовано архітектуру додатку, орієнтовану на модульність, масштабованість та безпеку. Реалізовано функціонал дистанційного оформлення замовлень, призначення завдань конкретним працівникам і контроль за їх виконанням, що дозволяє ефективно організувати робочі процеси без потреби в особистому контакті між сторонами.

У практичній частині реалізовано повнофункціональну CRM-систему. Реалізовано функціонал авторизації, реєстрації, створення та призначення завдань, погодження вартості замовлення, контроль його статусу, а також перегляд інформації про працівників. Система забезпечує зручну навігацію, інтуїтивний інтерфейс та адаптивність до різних типів користувачів.

Оцінено ефективність запропонованого рішення з точки зору зручності користування та впливу на внутрішні процеси компанії. Результати впровадження свідчать про покращення комунікації між сторонами, оптимізацію процесу обробки замовлень і підвищення загальної продуктивності працівників завдяки прозорості та структурованості виконання завдань.

Проведено тестування системи з метою перевірки її працездатності, зручності користування та відповідності початковим вимогам. Особливу увагу приділено захисту даних, включаючи використання механізмів хешування паролів, перевірки прав доступу та захисту від типових веб-уразливостей.

Таким чином, усі поставлені завдання виконано в повному обсязі. Розроблена CRM-система відповідає специфіці будівельної галузі, сприяє покращенню комунікації, підвищенню ефективності роботи працівників і забезпечує цифрову трансформацію процесів компанії. Отримані результати можуть стати основою для подальшого розвитку системи або її адаптації до інших сфер діяльності.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Держстат України. Динаміка будівельного ринку. URL: https://gmk.center/ua/posts/drajverom–zrostannia–budivnytstva–v–ukraini–u–2024–rotsi–stav–nezhytlovyj–sehment/ (дата звернення: 28.02.2025).

2. Аналітичний звіт про комерційну нерухомість. (2024). URL: https://gmk.center/ua/posts/drajverom–zrostannia–budivnytstva–v–ukraini–u–2024–rotsi–stav–nezhytlovyj–sehment/ (дата звернення: 03.03.2025).

3. Google Fit. Тенденції будівельного ринку України у 2024 році. URL: https://propertytimes.com.ua/blogs/andriy_ozeychuk/tendentsiyi_budivelnogo_rinku_ukrayini_u_2024_rotsi (дата звернення: 05.03.2025).

4. Zoho CRM Features. URL: https://www.zoho.com/ crm/features.html (дата звернення: 10.03.2025).

5. Buildertrend Pricing and Features. URL: https://www.buildertrend.com/ pricing (дата звернення: 13.03.2025).

6. PlanRadar. Features Overview. URL: https://www.planradar.com/ (дата звернення: 17.03.2025).

7. Laravel – The PHP Framework for Web Artisans. URL: https://laravel.com/ docs (дата звернення: 23.03.2025).

8. Google Maps Platform Documentation. URL: https://developers.google.com/ maps/ documentation (дата звернення: 29.03.2025).

9. DBeaver Universal Database Tool. URL: https://dbeaver.io/ (дата звернення: 01.04.2025).

10. Docker Documentation. URL: https://docs.docker.com/ (дата звернення: 08.04.2025).

11. Microsoft. Visual Studio Code documentation. URL: https://code. visualstudio.com/docs (дата звернення: 14.04.2025).

12. Laravel official documentation. URL: https://laravel.com/docs (дата звернення: 19.04.2025).

13. PHP Intelephense extension for VS Code. URL: https://marketplace.visualstudio.com/items?itemName=bmewburn.vscode-intelephense-client (дата звернення: 25.04.2025).

14. Laravel Blade Spacer. Blade Spacer extension. URL: https://marketplace.visualstudio.com/items?itemName=onecentlin.laravel-blade (дата звернення: 30.04.2025).

15. Docker Extension for Visual Studio Code. URL: https://marketplace.visualstudio.com/items?itemName=ms-azuretools.vscode-docker (дата звернення: 02.05.2025).

ДОДАТКИ

Додаток А – Код файлу welcome.blade.php

```
<!DOCTYPE html>
<html lang="{{ str_replace('_', '-', app()->getLocale()) }}">
<head>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1">

<title>{{ config('app.name', 'Laravel') }} - {{ __('messages.Home') }}</title>

<!-- Fonts -->
<link rel="preconnect" href="https://fonts.bunny.net">
<link
href="https://fonts.bunny.net/css?family=figtree:400,500,600&display=swap" rel="stylesheet" />

<!-- Scripts -->
@vite(['resources/css/app.css', 'resources/js/app.js'])
</head>
<body class="font-sans antialiased">
<div class="min-h-screen bg-gray-100">
<!-- Navigation -->
@include('layouts.navigation') <!-- Assuming this is already updated -->

<!-- Page Heading -->
<header class="bg-white shadow">
<div class="max-w-7xl mx-auto py-6 px-4 sm:px-6 lg:px-8">
<h1 class="font-semibold text-2xl text-gray-800 leading-tight">
{{ __('messages.Welcome to our Task Management Platform!') }}
</h1>
</div>
</header>

<!-- Page Content -->
<main class="py-12">
<div class="max-w-7xl mx-auto sm:px-6 lg:px-8">
<div class="bg-white overflow-hidden shadow-sm sm:rounded-lg">
<div class="p-6 text-gray-900">
<h2 class="text-xl font-semibold mb-4">{{ __('messages.Our Available Employees') }}</h2>
@if($employees->isEmpty())
<p>{{ __('messages.No employees are currently listed. Please check back later.') }}</p>
@else
<div class="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-6 mb-8">
@foreach ($employees as $employee)
<div class="bg-gray-50 p-4 rounded-lg shadow hover:shadow-md transition-shadow">
<h3 class="text-lg font-semibold text-gray-800">{{ $employee->full_name }}</h3>
<p class="text-sm text-gray-600">{{ $employee->position }}</p>
```

```
<p class="text-sm text-gray-500">{{ __('messages.Experience: :years
years', ['years' => $employee->experience]) }}</p>
</div>
@endforeach
</div>
@endif

@auth
<div class="mt-8 text-center">
<a href="{{ route('tasks.create') }}" class="inline-block bg-blue-500
hover:bg-blue-700 text-white font-bold py-3 px-6 rounded-lg text-lg shadow
hover:shadow-lg transition-shadow">
{{ __('messages.Submit New Task') }}
</a>
</div>
@else
<div class="mt-8 text-center">
<p class="mb-2 text-lg">{{ __('messages.To submit a task, please') }} <a
href="{{ route('login') }}" class="text-blue-600 hover:text-blue-800 font-
semibold">{{ __('messages.Log in') }}</a> {{ __('messages.or') }} <a
href="{{ route('register') }}" class="text-blue-600 hover:text-blue-800
font-semibold">{{ __('messages.Register') }}</a>.</p>
</div>
@endauth
</div>
</div>
</div>
</main>

<footer class="py-4 text-center text-sm text-gray-500">
<div class="my-4 flex justify-center">
<iframe
src="https://www.google.com/maps/embed?pb=!1m18!1m12!1m3!1d750.3928453913
24!2d25.347656839396745!3d50.75683878294355!2m3!1f0!2f0!3f0!3m2!1i1024!2i
768!4f13.1!3m3!1m2!1s0x4725975651dbee73%3A0xf16c1b9964f09370!2sPryvokzaln
a%20St%2C%209%2C%20Luts\\'k%2C%20Volyns\\'ka%20oblast%2C%2043000!5e0!3m2!
1sen!2sua!4v1748114230509!5m2!1sen!2sua" width="600" height="450"
style="border:0;" allowfullscreen="" loading="lazy" referrerpolicy="no-
referrer-when-downgrade"></iframe>
</div>
</footer>
</div>
</body>
</html>
```