Хмельницький національний університет

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

Пляцика Олександра Віталійовича

Прізвище, ім'я, по-батькові студента(ки)

на здобуття ступеня вищої освіти Бакалавра

Серверна частина системи управління взаємовідносинами з клієнтами

Назва теми

Рівень вищої освіти Перший (бакалаврський)

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

КПП.2201125.01.04.ПЗ

Виконав студент III курсу група ІПЗс-22-1 _____ Олександр ПЛЯЦИК

Підпис Ім'я, прізвище

Керівник канд. пед. наук, доцент _____ Оксана ОНИШКО

Науковий ступінь, звання Підпис Ім'я, прізвище

Нормоконтролер канд. тех. наук _____ Оксана ЯШИНА

Науковий ступінь, звання Підпис Ім'я, прізвище

До захисту допускаю:

Завідувач кафедри інженерії програмного забезпечення _____ Леонід БЕДРАТЮК

Підпис Ім'я, прізвище

10 червня 2025 р.

Хмельницький 2025

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій
Кафедра Інженерії програмного забезпечення
Рівень вищої освіти Перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри ІПЗ
Л. П. Бедратюк
2.01 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Пляцику Олександру Віталійовичу
Прізвище, ім'я, по батькові студента

1. Тема кваліфікаційної роботи Серверна частина системи управління взаємовідносинами з клієнтами

Керівник проекту (роботи) Онишко Оксана Григорівна, канд. пед. наук, доцент
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 07.02.2025 р. № 23
2. Строк подання студентом роботи на кафедру 01.06.2025 р.
3. Вихідні дані до роботи Матеріали переддипломної практики

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити) Дослідження предметної області та постановка задачі, проектування програмного забезпечення, програмна реалізація, тестування програмного забезпечення

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень) Презентаційні матеріали (слайди, 15 шт.)

6. Консультанти розділів дипломного проекту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Яшина О. М., доцент кафедри ІПЗ		
Антиплагіат	Форкун Ю. В., доцент кафедри ІПЗ		

7. Дата видачі завдання « 07 » 02 2025р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1 Ознайомлення з тематикою кваліфікаційних робіт (КвР), визначення та узгодження індивідуальної теми	01.12 – 31.12.2024	
2 Збір матеріалу за темою КвР; дослідження предметної області, в якій планується використання програмного забезпечення (ПЗ), визначення задач та вимог, розробка технічного завдання	01.01 – 20.02.2025	
3 Проектування програмного забезпечення	21.02 – 20.03.2025	
4 Програмна реалізація з використанням відповідних засобів розробки. Тестування ПЗ	21.03 – 30.04.2025	
5 Написання вступу, загальних висновків, оформлення джерел посилання та додатків. Оформлення пояснювальної записки КвР згідно вимог стандартів	01.05 – 25.05.2025	
6 Попередній захист КвР	Травень 2025	Згідно графіка
7 Перевірка КвР на плагіат, нормоконтроль, отримання відгуків, рецензій та інших супровідних документів. Брошурування (зшиття) пояснювальної записки.	26.05 – 30.05.2025	
8 Здача КвР на кафедру; підготовка КвР для розміщення у репозитарії ХНУ; підготовка до захисту та захист КвР	з 01.06.2025	

Студент _____ Олександр ПЛЯЦИК
Підпис Ім'я, прізвище

Керівник роботи _____ Оксана ОНИШКО
Підпис Ім'я, прізвище

АНОТАЦІЯ

Кваліфікаційна робота на тему: «Серверна частина системи управління взаємовідносинами з клієнтами».

Автор проєкту: Пляцик Олександр Віталійович.

Керівник проєкту: Онишко Оксана Григорівна.

Пояснювальна записка: 76 ст., 33 рис., 3 дод., 40 джерел.

Графічна частина: 15 слайдів.

На сьогоднішній день багатьом керівникам компаній, підприємств потрібно тримати інформацію в голові, якщо компанія невелика. Як тільки кількість співробітників почанає перевищувати 20 осіб, потрібно скористатитсь додатковим застосунком, проте, зазвичай, функціоналу не вистачає і доводиться користуватись послугами у інших сервісів.

Дипломний проєкт був виконаний на основі фреймоврку Laravel із застосуванням мов програмування Php, Js в середовищі для розробки програмного забезпечення PhpStorm. Також було використане середовище Docker, для контейнеризації, СУБД MySQL і Composer.

В ході розробки дипломного проєкту були визначені певні етапи, такі як:

- проектування бази даних і послідуюча її нормалізація;
- кодування, реалізація патерну MVC;
- тест на виявлення помилок;
- налагодження та повторне тестування;
- оформлення пояснювальної записки.

Пояснювальна записка складається з таких розділів: загальний розділ, розділ проєктування, розробка технічного та робочого проєкту, спеціальний розділи.

Отже, це програмне забезпечення дозволяє використати його для подальшої модернізації у повноцінну платформу, інтеграції в уже існуючиу або для керування компанії будь-якої спеціалізації, тощо.

Швидкодія, внаслідок використання Laravel, проста підтримка, внаслідок перевикористання базових класів є головними перевагами цієї програми.

10.06.25
Дата

Підпис

ВІДОМІСТЬ ДОКУМЕНТІВ

№ рядка	Формат	Позначення документа	Найменування документа	К-сть аркушів	№ екз.	Примітка
			Текстові документи			
1	А4	КПП.2201125.01.04.ПЗ	Пояснювальна записка	76		
2	А4		Завдання на кваліфікаційну роботу	1		
3	А4		Анотація	1		
			Графічні документи			
4	А4		Презентаційні матеріали	15		

Змн.	Арк.	№ докум.	Підпис	Дата	КПП.2201125.01.04.ВД		
Розробив		Пляцик О.В.		10.06	Серверна частина системи управління взаємовідносинами з клієнтами Відомість документів	Літ.	Арк. 1 / Аркушів 1
Керівник		Онишко О.Г.		10.06			
Н.контр.		Яшина О.М.		10.06		ХНУ.ІПЗс-22-1	
Зав. Каф.		Бедратюк Л.П.		10.06			

ЗМІСТ

Змін.	Аркуш	№ докум.	Підпис	Дата	КПІ.2201125.01.04.ПЗ		
Розробив		Пляцик О.В.		10.06	Серверна частина системи управління взаємовідносинами з клієнтами	Літ	Аркуш 1 / Аркушів 76
Керівник		Онишко О.Г.		10.06		Н	
Н.контр.		Яшина О.М.		10.06		ХНУ.ІПЗс-22-1	
Зав. Каф.		Бедратюк Л.П.		10.06			

ВСТУП

У сучасних компаніях, де кількість співробітників постійно зростає, особливо гостро постає питання централізованого зберігання інформації та автоматизації бізнес-процесів. Використання CRM-систем у таких умовах дозволяє не лише зберігати важливі дані про клієнтів і співробітників у єдиній базі, а й забезпечує ефективний контроль над взаємодією між різними відділами підприємства. Це сприяє підвищенню прозорості процесів, зменшенню людського фактору та зниженню ризиків помилок при обробці інформації.

Сучасні інформаційні технології вимагають від підприємств впровадження ефективних інструментів для управління персоналом і бізнес-процесами, що сприяють підвищенню конкурентоспроможності на ринку. Вебзастосунки CRM стають невід'ємною частиною цифрової трансформації компаній, забезпечуючи можливість швидкого доступу до актуальної інформації, аналізу даних і оперативного прийняття управлінських рішень.

Вибір серверної частини та її правильна архітектура відіграють ключову роль у забезпеченні стабільної роботи всієї системи. Сервер відповідає за обробку великого обсягу запитів, збереження конфіденційних даних та підтримку інтеграції з іншими бізнес-додатками. Тому оптимальний вибір технологій і методів розробки є критично важливим для успішної реалізації проєкту та подальшої його експлуатації.

Дана кваліфікаційна робота присвячена розробці серверної частини вебзастосунку CRM, спрямованої на управління персоналом та організацію бізнес-процесів в IT-компанії. Серверна частина виступає центральним елементом системи, відповідальним за зберігання, обробку, валідацію та безпечну передачу даних між фронтендом і базою даних. Від правильного вибору архітектури та технологій для серверної частини залежить загальна стабільність, масштабованість і продуктивність вебзастосунку.

Основною метою цього проєкту є створення надійного, безпечного та масштабованого серверного застосунку, який би максимально відповідав

вимогам сучасного бізнесу та забезпечував зручність у користуванні для кінцевих користувачів. Важливою складовою цієї мети є використання найбільш актуальних і перевірених фреймворків та інструментів, що дозволяють ефективно підтримувати подальший розвиток і вдосконалення системи.

Для досягнення поставленої мети необхідно виконати низку послідовних завдань, серед яких:

- ґрунтовний аналіз предметної області з визначенням усіх функціональних і нефункціональних вимог до серверної частини;
- проєктування архітектури вебзастосунку з урахуванням особливостей бізнес-процесів компанії та потреб користувачів;
- вибір найбільш оптимальних технологій, фреймворків та бібліотек для реалізації серверної логіки;
- розробка безпечної та ефективної серверної частини CRM-системи;
- проведення всебічного тестування програмного забезпечення, виявлення та усунення помилок, а також оптимізація продуктивності.

Реалізація серверної частини надасть вебзастосунку необхідну надійність та швидкодію, дозволить підтримувати велику кількість одночасних користувачів без втрати якості роботи системи. Крім того, правильно спроєктована серверна логіка забезпечить інтеграцію з іншими сервісами, розширюваність функціоналу та гнучкість у налаштуваннях.

Завдяки комплексному підходу до розробки, система дозволить автоматизувати ключові процеси управління персоналом, такі як облік робочого часу, планування відпусток, контроль за проектами і подіями, а також організацію внутрішніх комунікацій через систему сповіщень і зустрічей. Це підвищить загальну продуктивність працівників та покращить якість управлінських рішень у компанії.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Змістовний аналіз та опис предметної області, її структурних та функціональних особливостей

Перш ніж заглиблюватися в деталі, варто чітко визначити, що таке CRM-система (Customer Relationship Management) і яку роль вона відіграє в сучасному бізнесі. CRM – це спеціалізоване програмне забезпечення, створене для автоматизації і оптимізації взаємодії компанії з її замовниками. Основна мета CRM-системи полягає в підвищенні ефективності управління процесами продажів, маркетинговими кампаніями, підтримкою клієнтів та іншими бізнес-процесами, які безпосередньо впливають на відносини з клієнтською базою.

По суті, CRM-система – це потужний інструмент, що дозволяє зберігати і структурувати інформацію про контакти і компанії, сегментувати клієнтів за різними критеріями, контролювати хід угод, формувати і відстежувати воронку продажів, а також генерувати аналітичні звіти для оцінки роботи команди і планування розвитку бізнесу. Головна ідея полягає в тому, щоб упорядкувати бізнес-процеси, стандартизувати роботу з базою клієнтів, спростити виконання щоденних завдань менеджерів, а також прискорити обробку заявок і оптимізувати взаємодію між різними відділами компанії.

Сучасні CRM-системи можуть бути реалізовані у вигляді локальних додатків, які встановлюються на робочих станціях користувачів, або ж як хмарні сервіси, що надають доступ до інформації через інтернет з будь-якого пристрою, включаючи мобільні телефони. Це забезпечує гнучкість у роботі співробітників і дозволяє їм ефективно керувати клієнтською базою незалежно від місцезнаходження.

Для бізнесу, який орієнтований на роботу з клієнтами, критично важливо контролювати кожен етап взаємодії – від реєстрації заявок, ведення історії комунікацій, контролю перебігу угод, планування зустрічей і дзвінків, до розподілу завдань між співробітниками. Зазвичай у бізнес-процесах

використовуються різноманітні інструменти, такі як електронна пошта, месенджери, таблиці, календарі і таск-менеджери. Однак CRM-система об’єднує всі ці функції в єдине робоче середовище, усуваючи необхідність перемикатися між різними додатками, що значно підвищує ефективність роботи.

Крім організаційних функцій, CRM-система також автоматизує багато рутинних процесів, зменшуючи навантаження на менеджерів з продажу. Наприклад, автоматичне створення карток клієнтів, внесення їх даних, запуск розсилок, оновлення статусів угод у воронці продажів – все це дозволяє сконцентрувати увагу співробітників на виконанні стратегічних завдань.

Важливою структурною характеристикою CRM є інтеграція різних функціональних модулів, які разом забезпечують повний цикл роботи з клієнтом. Це охоплює збір та сегментацію контактної інформації, ведення історії комунікацій, управління угодами, автоматизацію маркетингу, підтримку клієнтів через систему заявок, а також збір аналітики і формування звітів для оцінки результативності роботи компанії. Кожен із модулів виконує чітко визначені функції, а їхня взаємодія організована через єдиний, зручний інтерфейс користувача.

Структура CRM-системи зазвичай базується на багаторівневій архітектурі, що дозволяє чітко розділити відповідальність між компонентами: інтерфейсом користувача, бізнес-логікою та системою управління базою даних. Такий підхід забезпечує високу гнучкість і масштабованість, а також можливість інтеграції із зовнішніми сервісами та додатками.

Однією з важливих складових є система ролей і прав доступу, яка гарантує безпеку інформації і запобігає несанкціонованому доступу. Користувачі мають доступ лише до тієї інформації і функцій, які потрібні для виконання їх професійних обов’язків. Це підвищує рівень безпеки і захищає дані від випадкового чи навмисного порушення.

В умовах сучасного ринку CRM-системи все частіше реалізуються як вебзастосунки з клієнт-серверною архітектурою, що забезпечує мобільність і доступність для співробітників з будь-яких пристроїв. Це робить систему

зручною для масштабування і дозволяє швидко адаптуватися до змін у бізнес-процесах.

Отже, предметна область CRM охоплює широкий спектр бізнес-процесів, які можуть бути ефективно автоматизовані для підвищення продуктивності компанії, покращення якості обслуговування клієнтів і, як наслідок, збільшення прибутковості та конкурентоспроможності на ринку.

Загалом, CRM-система виступає ключовим інструментом для сучасного бізнесу, що дозволяє оптимізувати роботу команди, стандартизувати процеси взаємодії з клієнтами і підвищити якість надання послуг. Вона є невід'ємною частиною стратегії управління відносинами з клієнтами, що сприяє довгостроковому успіху і стабільному розвитку підприємства.

1.2 Аналіз існуючих методів та програмно-технічного забезпечення предметної області

Everhour [1] (рисунок 1.1) – це хмарна система для керування проєктами. Вона дозволяє компаніям відстежувати робочий час співробітників, розраховувати бюджети, управляти завданнями та виставляти рахунки. Everhour підходить для маркетингових і рекламних команд, некомерційних організацій, консалтингових фірм та інших галузей бізнесу. Завдяки інтеграціям із зовнішніми хмарними сервісами можна розширити функціональність системи.

Everhour пропонує користувачам кілька ключових можливостей: управління проєктами, контроль часу, розрахунок бюджету та формування рахунків. Проєкти можна переглядати у форматі списку завдань або канбан-дошки. У завданнях можна визначати пріоритети, встановлювати статуси, додавати теги для швидкого пошуку та фіксувати витрачений час. Для планування діяльності керівники можуть використовувати інтерактивний часовий графік, що відображає завантаженість працівників, їхній розклад і плани. Графіки містять фільтри для пошуку необхідної інформації, перегляду розподілу ресурсів та позначення вихідних чи відгулів.

Облік робочого часу здійснюється через браузерне розширення та мобільний застосунок. Співробітники можуть вести логування часу безпосередньо в завданнях, фіксувати витрати й користуватися таймером для автоматичного підрахунку робочого часу.

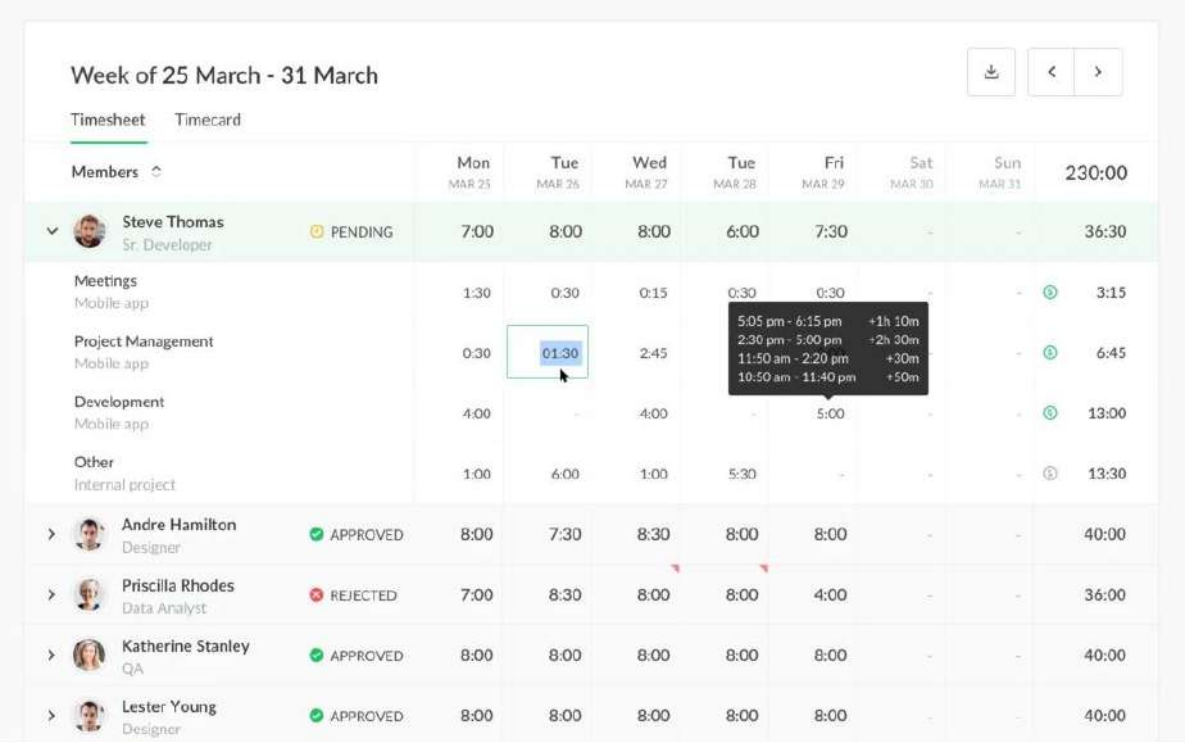


Рисунок 1.1 – Вигляд сервісу Everhour

ActiveCollab [2] (рисунок 1.2) – це вебплатформа для керування проєктами та організації командної роботи. Вона дозволяє централізовано зберігати всі дані щодо проєкту, координувати бізнес-процеси та контролювати хід виконання завдань. Систему можна встановити на власний сервер або використовувати в хмарному середовищі для аналізу ефективності проєктів та розподілу робочого часу. ActiveCollab допомагає оцінювати продуктивність співробітників, переглядати їхні профілі та аналізувати витрати на виконання завдань.

Система пропонує широкий набір функцій для керування проєктами, бізнес-процесами та робочими завданнями. Вона дозволяє оцінювати продуктивність команди, контролювати часові витрати та визначати реальну вартість проєкту.

Основні можливості ActiveCollab:

– централізована платформа для спільної роботи над проєктами;
– інтеграція з електронною поштою;
– взаємодія між членами команди;
– керування проєктами та завданнями;
– календар для планування;
– перегляд і аналіз профілів користувачів;
– масштабовані тарифні плани;
– підтримка багатокористувацького доступу;
– щомісячна та щорічна система оплат;
– мобільні застосунки;
– інструменти для оцінки витрат і контролю робочого часу.

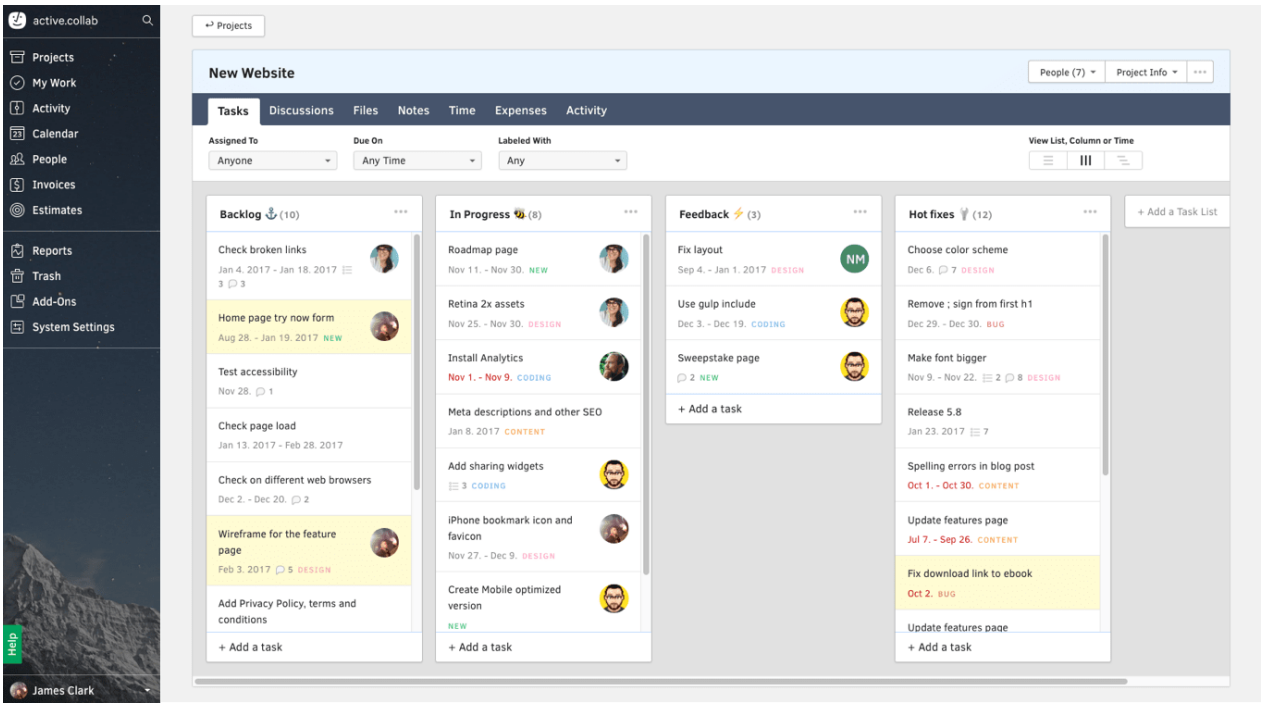


Рисунок 1.2 – Вигляд вебзастосунку ActiveCollab

Clockify [3] (рисунок 1.3) – це сервіс для контролю робочого часу, який допомагає компаніям відстежувати продуктивність співробітників, їхню присутність на роботі та оплачувані години. Основні можливості платформи включають аудит часу, експорт даних, нагадування, налаштовувані звіти та управління доступом.

Інтерфейс Clockify складається з чотирьох основних панелей: таймер, розклад, дашборд і звіти. У розділі «Таймер» користувач може створювати власні проєкти, додавати до них завдання, встановлювати теги та вартість, а також запускати таймер для фіксації робочого процесу. «Розклад» представлений у вигляді календаря, де відображаються виконані завдання та витрачений на них час.

Панель «Дашборд» містить інформацію про продуктивність користувачів, показуючи, скільки часу витрачено та на що саме. Clockify також пропонує три типи звітів: підсумковий, детальний і тижневий.

Ключові особливості Clockify:

– система запрошень і керування персоналом (керівник може додавати співробітників через запрошення на електронну пошту);
– можливість експорту звітів;
– логування робочого процесу;
– створення приватного робочого простору;
– розширений пошук із використанням тегів і фільтрів.

У майбутньому можуть з’явитися аналоги деяких функцій цих інструментів, адже вони корисні та дозволяють централізовано зберігати інформацію. Водночас слід зазначити, що ці системи використовуються великими компаніями й доведені до високого рівня ефективності. Тому вибір подібних рішень, як і інших платформ для управління персоналом, дозволяє зосередитися на найважливіших функціях, адже деякі з них можуть залишатися невикористаними, наприклад, оцінка часу та витрат у ActiveCollab.

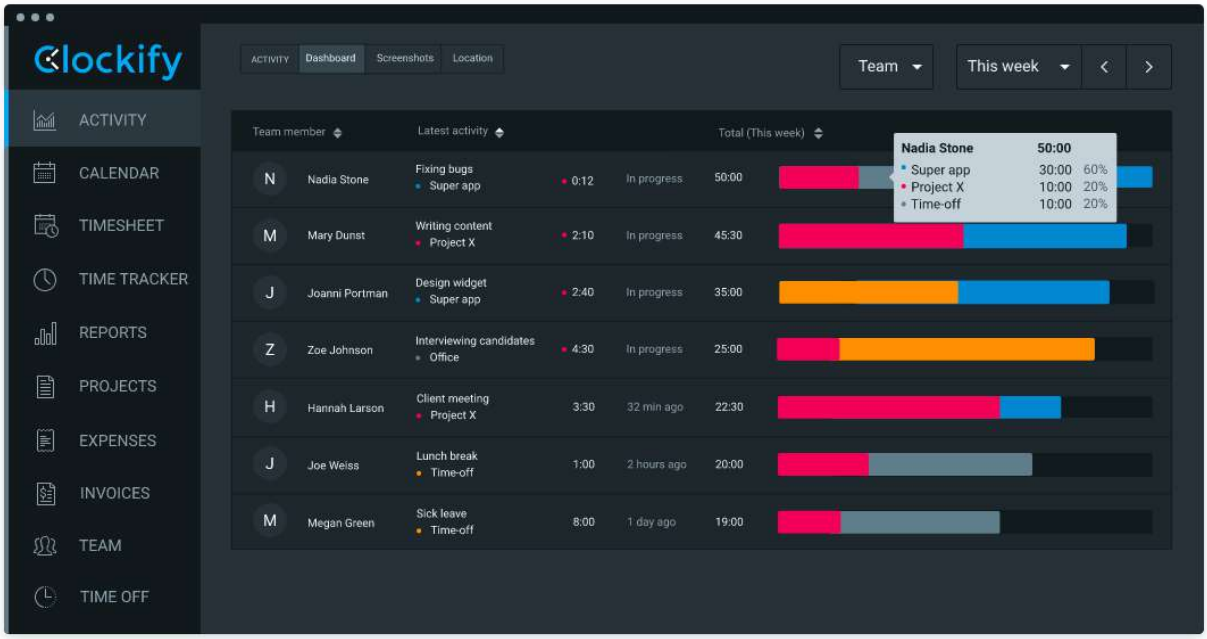


Рисунок 1.3 – Вигляд платформи Clockify

В інших варіантах програм, є багато недоліків та прорахувань, або вебзастосунки є занадто дорогими, з недостатнім функціоналом.

Враховуючи всі недоліки потрібно створити програму із зручним інтерфейсом та достатньою кількістю функцій для роботи.

1.3 Визначення функціональних та нефункціональних вимог до програмного забезпечення

Вхідні дані в системі будуть формуватися на основі інформації користувача, до якої в подальшому прив’язуватимуться інші сутності. Наприклад, «Події» – це сутність, що міститиме всі корпоративні заходи компанії; «Проєкти» – об’єкт, який зберігатиме інформацію про активні проєкти та користувачів, що в них задіяні; «Особливі дні» – блок, який дозволить працівникам позначати дні відпустки, лікарняні або роботу з дому; а «Зустрічі» – це сутність, яка міститиме дані про заплановані наради та обговорення.

Таблиця 1.1 – Вхідні дані в систему

Сутність	Опис
Події	Важливі корпоративні події, які створюються адміністраторами вручну. Це можуть бути державні вихідні, корпоративи, річниці або дні народження працівників. Вся ця інформація відображатиметься у спеціальному календарі, де кожен день може містити кілька подій або жодної.
Користувачі	Керування працівниками: створення, редагування та видалення профілів. Включає детальну інформацію, таку як сімейний стан, кваліфікація, вік, спеціальні навички тощо. Персонал ділиться на звичайних користувачів, адміністраторів та супер-адмінів.
Проєкти	Додавання та управління проєктами, включаючи закріплення відповідальних користувачів та визначення їхніх ролей. У майбутньому планується інтеграція чату для спілкування в межах кожного проєкту.
Особливі дні	Дні, які відрізняються від стандартних робочих (відпустка, лікарняний, дистанційна робота, відрядження, неоплачувана відпустка). Система дозволяє налаштовувати доступну кількість днів відпустки, які будуть автоматично нараховуватися щомісяця, а також конфігурувати ліміти на лікарняні та дистанційні дні. Користувачі можуть подавати запити на їх використання для погодження адміністратором.
Зустрічі	Адміністратор може створювати зустрічі (персональні або групові) для обговорення важливих питань. Наприклад, зустрічі можуть бути призначені для співбесід, обговорення підвищень, оцінки кваліфікації або вирішення нагальних робочих питань.

Уся вхідна інформація безпосередньо пов’язана з користувачем, оскільки саме ця сутність є ключовою в системі. На основі даних про користувача формуються всі інші сутності, тому важливо забезпечити можливість їхнього збереження для подальшого використання.

Результуюча (вихідна) інформація – це дані, які користувач отримує в результаті роботи з системою та які зберігаються в базі даних. У проєкті «Customer Relationship and Management» основною вихідною інформацією буде вебсторінка, яка відображатиме всі необхідні дані для роботи користувача.

Після заповнення інформації (що зберігатиметься у відповідних таблицях БД) користувач зможе:

– створювати запити на погодження особливих днів,
– переглядати інформацію про інших співробітників,
– слідкувати за подіями та проєктами,
– контролювати свої заплановані зустрічі.

Кожна вкладка вебзастосунку надаватиме доступ до відповідної інформації:

– «Особливі дні» – відображатиме користувачів та їхні запити на вихідні або віддалену роботу.
– «Події» – міститиме календар корпоративних і державних свят, доступний для перегляду.
– «Користувачі» – надаватиме доступ до інформації про співробітників, включаючи перелік проєктів, у яких вони задіяні.
– «Зустрічі» – дозволятиме переглядати заплановані зустрічі з фільтрацією за місяцями.

Таким чином, система забезпечить користувачам доступ до всієї необхідної інформації, що спростить управління робочими процесами та комунікацію між співробітниками.

Крім базових сутностей, у системі передбачено розширені механізми взаємодії між ними, що дозволяють ефективно управляти інформаційними потоками та забезпечують гнучкість роботи. Наприклад, система підтримує

комплексне управління ролями та правами доступу, що дає змогу розмежовувати повноваження користувачів залежно від їх посад та обов'язків. Це сприяє підвищенню рівня безпеки і знижує ризики випадкових або умисних помилок у роботі з даними.

Інтеграція з календарем та системою сповіщень забезпечує автоматичне нагадування про важливі події, зустрічі або строки погоджень, що допомагає організувати робочий процес і покращує комунікацію між працівниками. Система також підтримує історію змін для ключових сутностей, що дозволяє відслідковувати будь-які модифікації, аналізувати їхній вплив і, у разі потреби, відновлювати попередні версії даних. Це особливо важливо для підтримки цілісності інформації і проведення аудитів.

Важливою складовою є модульна архітектура системи, яка розділяє функціональні блоки на незалежні компоненти. Кожен модуль відповідає за окремий аспект роботи – наприклад, управління подіями, контроль за проектами чи обробку запитів на особливі дні. Такий підхід не лише підвищує масштабованість і гнучкість системи, але й значно спрощує її підтримку і подальший розвиток. Модульність також сприяє повторному використанню коду в інших проектах або нових версіях системи.

З огляду на сучасні тенденції, система передбачає можливість розгортання як локально, так і у хмарному середовищі, що розширює варіанти її використання і забезпечує мобільність співробітників. Можливість доступу через вебінтерфейс із будь-якого пристрою дозволяє працівникам бути завжди в курсі подій і оперативно реагувати на зміни.

Враховуючи великий обсяг даних, що обробляється, система використовує ефективні механізми кешування та оптимізації запитів до бази даних, що знижує час відгуку і підвищує загальну продуктивність.Всі ці технічні рішення спрямовані на створення зручного, швидкого та надійного інструменту для комплексного управління взаємовідносинами з клієнтами і персоналом.

1.4 Висновки до першого розділу. Постановка задачі

Тема дипломного проєкту – вебзастосунок «Серверна частина системи управління взаємовідносинами з клієнтами».

У сучасному світі багато керівників компаній використовують різні сервіси для збору та обробки інформації про своїх співробітників. Щоб уникнути необхідності особистого збору даних кожного працівника, доцільно створити платформу, де працівники самостійно вносять необхідну інформацію без втручання керівника.

Мета проєкту – створення системи управління персоналом, що функціонує в режимі реального часу.

У межах проєкту передбачено:

– управління персоналом;
– контроль особливих днів;
– календар важливих подій;
– управління проєктами;
– система сповіщень;
– система зустрічей;
– ролі адміністратора та звичайного користувача.

Отже, вся необхідна інформація буде централізовано оброблятися в одній системі, що дозволить уникнути використання сторонніх сервісів.

Вимоги до вебзастосунку:

– можливість легкого розширення функціоналу;
– гнучкість завдяки конфігурації параметрів;
– зручний користувацький інтерфейс;
– перегляд результатів роботи.

Дані зберігатимуться у реляційній базі даних MySQL, яка буде приведена до третьої нормальної форми.

Повна назва проєкту – «Серверна частина системи управління взаємовідносинами з клієнтами». Робоча назва – «CRaM» («Управління та менеджмент персоналу»).

Програмний продукт має забезпечити:

– управління персоналом та проєктами компанії;
– контроль важливих подій та особливих дат;
– функціонування системи сповіщень і зустрічей;
– можливість створення нотаток та чат для комунікації;
– відстеження робочого часу для подальшого аналізу ефективності роботи компанії.

Реалізація проєкту включає наступні етапи:

– вибір середовища програмування для написання коду;
– визначення системи управління базами даних;
– організація вхідних і вихідних даних.

Проєкт буде розроблений на PHP-фреймворку Laravel для серверної частини та React.js [19] для фронтенду з використанням інтегрованого середовища розробки PHPStorm.

PHPStorm обраний через зручний редактор MySQL з графічним інтерфейсом, а також підтримку додаткових плагінів для роботи з кодом. MySQL як система управління базами даних є надійною та використовується у популярних вебзастосунках.

З огляду на можливі оновлення технологій під час розробки, проєкт буде створений із урахуванням сучасних тенденцій, щоб уникнути застарівання ще до запуску.

Вимоги до програмного продукту:

– нормалізована база даних;
– підтримка великої кількості користувачів;
– зручна навігація по сторінках;
– можливість перегляду результатів роботи;
– сучасний та адаптивний інтерфейс.

Загальна схема розробки програмного забезпечення:

– аналіз і збір інформації;
– формування вимог;
– проєктування архітектури;
– реалізація;
– тестування;
– запуск і написання документації;
– подальший супровід.

На етапі аналізу необхідно зібрати інформацію про майбутній продукт, вивчити предметну область та існуючі рішення на ринку, визначити основні проблеми та шляхи їх вирішення. Останнім часом IT-компанії активно запитують автоматизовані CRM-системи для збору інформації про персонал.

Формування вимог передбачає створення концептуальної моделі додатку, опис сценаріїв взаємодії користувача із системою, визначення ключових функцій та технологій.

На етапі проєктування необхідно деталізувати технічні рішення для реалізації вимог. Важливу роль у цьому процесі відіграє використання UML для оптимізації процесу розробки та ефективного обміну інформацією між розробниками.

Етап реалізації включає написання програмного коду, поетапне тестування окремих компонентів і поступову інтеграцію всієї системи. Це складний процес, який потребує глибокого врахування специфіки вибраного середовища розробки та обраної мови програмування. Після завершення основної розробки система проходить повне тестування, яке охоплює всі можливі варіанти її використання, включаючи граничні ситуації. Виявлені помилки ретельно аналізуються і усуваються. Лише після цього система передається в експлуатацію. Окрім того, готуються інструкції та довідкові матеріали, які допоможуть кінцевим користувачам ефективно працювати з продуктом. На завершальному етапі запуску проєкту важливо забезпечити його технічну підтримку та супровід для гарантії стабільної й безперебійної роботи.

Вимоги до тестування:

– перевірка повноти реалізованих функцій;
– аналіз коректності взаємодії користувача із системою;
– формалізація вимог у форматі «вхід-вихід», «подія-наслідок», «умова-відповідь».

Таким чином, вебзастосунок «Серверна частина системи управління взаємовідносинами з клієнтами» стане ефективним інструментом для автоматизації процесів ІТ-компаній, забезпечуючи централізований контроль за всіма аспектами роботи персоналу та проєктів.

Важливим аспектом розробки вебзастосунку є забезпечення інтуїтивно зрозумілого інтерфейсу користувача. Зручність і простота навігації безпосередньо впливають на ефективність роботи співробітників, що користуються системою щодня. Для цього у проєкті застосовуються сучасні підходи до дизайну користувацького досвіду (UX/UI [36]), які передбачають адаптивність інтерфейсу для різних пристроїв і різних категорій користувачів. Впровадження таких рішень сприяє швидкому освоєнню системи, зменшенню кількості помилок при роботі та підвищенню загальної продуктивності команди.

Ще однією суттєвою перевагою вебзастосунку є можливість його інтеграції з іншими сервісами та платформами, що вже використовуються в організації. Це дозволяє побудувати єдине інформаційне середовище, в якому дані автоматично синхронізуються між різними системами, знижуючи навантаження на користувачів і мінімізуючи ризик втрати або дублювання інформації. Використання відкритих стандартів обміну даними, таких як REST API [25], відкриває широкі можливості для подальшого розвитку системи та інтеграції з популярними сервісами аналітики, комунікаційними платформами та хмарними сховищами.

Важливо також відзначити, що впровадження вебзастосунку такого рівня складності вимагає системного підходу до організації процесу розробки та впровадження. Саме тому кожен з етапів – від аналізу предметної області до супроводу – був ретельно спланований і виконаний із застосуванням сучасних

методологій управління проектами, таких як Agile [37] та Scrum [38]. Це дозволяє не лише оперативно реагувати на зміни вимог замовника, а й забезпечує постійну комунікацію між членами команди, що підвищує якість кінцевого продукту.

Особливу увагу приділено адаптивності системи, яка є критичною у сучасному динамічному бізнес-середовищі. Вебзастосунок спроектовано так, щоб він міг безболісно масштабуватися відповідно до зростання компанії, розширюватися за рахунок нових функцій і інтегруватися з іншими корпоративними системами через стандартизовані API [40]. Це робить проєкт не лише корисним у поточному стані, а й перспективним інструментом для подальшого розвитку бізнесу.

У процесі розробки особливо враховувалися вимоги до безпеки інформації, адже управління персональними та корпоративними даними вимагає дотримання законодавчих норм і найкращих практик у сфері інформаційної безпеки. Для цього застосовувалися сучасні підходи, включаючи шифрування, автентифікацію, рольовий доступ та аудит дій користувачів. Такий підхід забезпечує надійний захист від несанкціонованого доступу та гарантує збереження конфіденційності даних.

Загалом, створення системи керування взаємовідносинами з клієнтами у вигляді вебзастосунку є актуальним завданням, яке відповідає потребам сучасного бізнесу. Завдяки широкому функціоналу, зручності у використанні і високій надійності, розроблений продукт покликаний значно підвищити ефективність управління персоналом та покращити внутрішні бізнес-процеси компанії.

2 ПРОЕКТУВАННЯ ТА ВИЗНАЧЕННЯ МЕТОДІВ І ЗАСОБІВ РЕАЛІЗАЦІЇ

2.1 Аналіз та вибір архітектури вебзастосунку

Архітектура розробки програмного забезпечення – це фундаментальна структура програми, яка визначає взаємодію її компонентів, зокрема, зв'язок між інтерфейсною частиною та внутрішніми процесами. По суті, це концепція, яка регламентує відповідальність кожного модуля та способи їхньої взаємодії.

Розглядаючи архітектуру програмного забезпечення для кваліфікаційної роботи «Серверна частина системи управління взаємовідносинами з клієнтами», необхідно вибрати оптимальну модель взаємодії між клієнтом і сервером. З огляду на специфіку завдання, система буде складатися із серверної та клієнтської частини. Для реалізації такої взаємодії можна використати кілька основних підходів, зокрема файлово-серверну та клієнт-серверну архітектури.

Файлово-серверна архітектура передбачає, що всі дані зберігаються у вигляді файлів на окремому файловому сервері. Клієнти надсилають запити на отримання файлів, після чого виконують обробку даних на власних пристроях. Це означає, що кожен клієнт самостійно виконує всі необхідні зміни, оновлення індексів і роботу з базою даних. Такий підхід має кілька значних недоліків: високе навантаження на мережу через передачу великих обсягів даних, проблеми з узгодженістю даних і цілісністю бази, а також потенційне зниження продуктивності при збільшенні кількості користувачів.

Натомість клієнт-серверна архітектура значно ефективніша, оскільки сервер бази даних не лише зберігає інформацію, а й виконує всі операції з її обробки. Клієнти надсилають серверу SQL-запити на отримання або зміну даних, а сервер самостійно виконує всі необхідні обчислення, контролює цілісність бази та повертає лише результати у вигляді набору записів або коду відповіді. Це знижує навантаження на мережу, оскільки передається значно менший обсяг даних, і підвищує продуктивність системи, оскільки всі критично важливі обчислення виконуються централізовано на сервері.

До переваг клієнт-серверної архітектури належать:

– централізоване управління даними, що підвищує їхню узгодженість і безпеку;
– ефективний розподіл обчислювального навантаження між сервером і клієнтами;
– вища масштабованість, оскільки сервер може обробляти велику кількість запитів одночасно;
– покращена продуктивність системи завдяки зменшенню обсягу передаваних даних.

Враховуючи всі ці фактори, для реалізації даної кваліфікаційної роботи буде використана архітектура клієнт-сервер, яка забезпечить стабільність, високу продуктивність та ефективне управління даними.

При виборі архітектури вебзастосунку особливу увагу слід звернути на забезпечення розділення відповідальностей між компонентами системи. Наприклад, реалізація патерну MVC [10] дозволяє виділити окремо логіку представлення, логіку обробки даних та контролери. Це забезпечує зручність підтримки, масштабованість і повторне використання коду.

У сучасній практиці розробки перевага надається мікросервісній архітектурі [35], проте для даної кваліфікаційної роботи обґрунтованим вибором є моноліт із чітко виокремленими модулями. Такий підхід дозволяє забезпечити керованість проєкту на етапі розробки, скоротити кількість залежностей та уникнути зайвого перевантаження інфраструктури.

Крім того, враховувалося забезпечення RESTful API для взаємодії з фронтендом, що дозволяє легко масштабувати застосунок і підключати зовнішні клієнтські рішення, наприклад, мобільні застосунки або сторонні сервіси. Стандартизовані маршрути дозволяють реалізовувати CRUD-операції над сутностями з мінімальними зусиллями.

Ключовим аспектом створення будь-якої сучасної програмної системи є розробка надійної, гнучкої та масштабованої архітектури. Вона не лише визначає

структуру майбутнього застосунку, але й безпосередньо впливає на ефективність процесу розробки, підтримки та подальшого розвитку системи.

У рамках даної кваліфікаційної роботи архітектура системи була спроєктована відповідно до принципів модульності, ізольованості функцій і можливості подальшої масштабованості. Це означає, що кожен компонент виконує окрему, чітко визначену функцію, а його взаємодія з іншими частинами системи побудована через інтерфейси або API. Такий підхід значно полегшує внесення змін, дає змогу перетестовувати окремі частини без впливу на решту, а також полегшує впровадження нових функцій.

Обрана клієнт-серверна архітектура побудована з урахуванням багаторівневої структури, що охоплює рівень представлення (frontend), рівень обробки запитів (контролери та сервісна логіка), рівень бізнес-логіки та рівень доступу до даних (репозиторії, моделі, бази даних). Завдяки цьому забезпечується висока ступінь ізоляції між модулями, що дозволяє незалежно змінювати або оновлювати частини системи без ризику порушення її цілісності.

На рівні серверної частини реалізовано використання фреймворку Laravel, який ґрунтується на шаблоні проєктування MVC (Model-View-Controller). Цей патерн дозволяє чітко розмежувати відповідальність кожного шару: моделі відповідають за взаємодію з базою даних, контролери – за логіку обробки запитів, а представлення – за відображення інформації у відповідному форматі. Така структура спрощує супровід коду, сприяє його повторному використанню та забезпечує кращу структурованість.

Для підвищення гнучкості обрано модульний підхід до організації функціоналу. Кожен функціональний блок (автентифікація, робота з клієнтами, управління знижками тощо) реалізується у вигляді окремого модуля або пакета, який можна легко вбудовувати, вимикати або розширювати. Цей підхід дозволяє реалізувати масштабування не лише вертикальне, тобто додавання ресурсів, а й горизонтальне – додавання нових функцій через нові модулі без зміни базової структури.

У майбутньому така архітектура відкриває шлях до поступового переходу на мікросервісну модель, яка передбачає розділення системи на автономні сервіси, що взаємодіють через API. Це особливо актуально для розширення масштабів платформи, підключення сторонніх сервісів або збільшення кількості одночасних користувачів.

Також особливу увагу приділено забезпеченню RESTful архітектурного стилю для побудови API. Це дозволяє гарантувати однаковий підхід до реалізації ендпоінтів, забезпечити ієрархічну структуру URL, використовувати стандартизовані методи HTTP (GET, POST, PUT, DELETE), а також забезпечити легкість у підключенні сторонніх клієнтів, таких як мобільні застосунки, SPA-фреймворки або навіть сторонні API-агенти.

Крім того, важливим аспектом архітектурного проєктування стало врахування вимог до продуктивності та безпеки. Було впроваджено розділення доступу до ресурсів, застосовано middleware для контролю авторизації, а також реалізовано кешування результатів частих запитів за допомогою вбудованих засобів Laravel. Це дозволяє досягти стабільної роботи системи навіть при високих навантаженнях.

Окремо варто згадати, що середовище виконання програми організовано з використанням XAMPP – кросплатформеного середовища, яке містить Apache, MySQL і PHP. Це забезпечує локальну розробку, тестування, зокрема, застосовуватиметься інструмент Postman [23] для перевірки REST-запитів та налагодження проєкту без необхідності зовнішнього хостингу на перших етапах.

Таким чином, запропонована архітектура програмного забезпечення повністю відповідає сучасним вимогам до розробки, забезпечує високу масштабованість, зручність підтримки та можливість інтеграції з іншими системами. Вона оптимально підходить для реалізації функціоналу системи керування взаємовідносинами з клієнтами, підтримуючи її стабільність, гнучкість і ефективність.

2.2 Аналіз та вибір технологій і засобів реалізації системи

Порівняння серверних фреймворків

При виборі серверного фреймворку для розробки CRM-системи важливо враховувати різні аспекти: продуктивність, безпеку, простоту використання, підтримку спільноти та можливість масштабування. Кожен фреймворк має свої переваги і недоліки, а також різні сфери застосування. Деякі фреймворки орієнтовані на швидкість розробки, інші – на продуктивність або безпеку. Тому перед прийняттям остаточного рішення варто проаналізувати найпопулярніші серверні технології, їхні особливості та обмеження.

У таблиці нижче наведено порівняння найпопулярніших серверних фреймворків, що використовуються для створення вебзастосунків. Основними критеріями оцінки є мова програмування, переваги та недоліки кожного фреймворку.

Для реалізації серверної частини CRM-системи управління IT-компанією розглянуто кілька сучасних серверних фреймворків: Laravel (PHP), Django (Python), Express.js (Node.js), Spring Boot (Java), ASP.NET Core (C#) та Ruby on Rails (Ruby) (таблиця 2.1).

Таблиця 2.1 – Порівняння популярних фреймворків

1	2	3	4
Laravel	PHP	Велика спільнота, висока продуктивність, підтримка MVC, вбудовані засоби безпеки, зручна ORM (Eloquent), гнучкість та масштабованість	Відносно невисока продуктивність порівняно з Express.js та Java

Продовження таблиці 2.1

1	2	3	4
Django	Python	Високий рівень безпеки, швидкість розробки, вбудована ORM, автоматичне адміністрування	Складність налаштування та розгортання, низька продуктивність для високонавантажених систем
Express.js	JavaScript	Висока продуктивність, асинхронна обробка запитів, велика кількість бібліотек	Відсутність вбудованих засобів безпеки, необхідність додаткових модулів для реалізації MVC
Spring Boot	Java	Висока продуктивність, масштабованість, багатий набір функціоналу, підтримка мікросервісів	Висока складність налаштування, значні ресурси для роботи
ASP.NET Core	C#	Висока продуктивність, підтримка кросплатформності, потужні засоби безпеки, інтеграція з Azure	Високий поріг входу, складність налаштування, потребує ресурсів сервера
Ruby on Rails	Ruby	Прискорений цикл розробки, зручна ORM Active Record, конвенції замість конфігурацій	Низька продуктивність, менша популярність у корпоративному секторі

Серверна частина будь-якого сучасного вебзастосунку – це серце системи, яке відповідає за обробку запитів, збереження та зміну даних, забезпечення бізнес-логіки, авторизації, а також безпеку. Тому вибір фреймворку для реалізації бекенд-частини є критично важливим. Під час аналізу можливих варіантів було розглянуто як легковагові фреймворки, що надають розробнику максимальну свободу, так і фреймворки з чіткою структурою, що дозволяють створювати масштабовані та надійні системи у стислі терміни.

Laravel є одним з найкращих виборів для реалізації серверної частини CRM-системи з наступних причин:

– Легкість розробки – Laravel має зручний синтаксис, що значно пришвидшує розробку системи;
– Безпека – фреймворк включає засоби для захисту від SQL-ін'єкцій, CSRF-атак, XSS-атак та інших загроз;
– ORM Eloquent [33] – спрощує роботу з базою даних MySQL, що важливо для реалізації нормалізованої структури;
– Масштабованість – Laravel підтримує модульну архітектуру, що дозволяє легко розширювати функціонал;
– Підтримка реального часу – завдяки WebSockets та інтеграції з Laravel Echo, можна легко реалізувати систему сповіщень та чат;
– Популярність – велика спільнота та документація, що спрощує розв'язання можливих проблем;
– Екосистема – Laravel має широкий вибір офіційних та сторонніх пакетів (наприклад, Laravel Horizon, Laravel Nova, Laravel Sanctum [21]), що полегшує впровадження багатьох функцій;
– Тестування – вбудована підтримка PHPUnit [27], що дозволяє легко писати та запускати тести;
– Підтримка API – Laravel забезпечує зручне створення REST API завдяки Laravel Passport або Sanctum.

Laravel продовжує активно розвиватися, що гарантує його відповідність сучасним вимогам веброзробки. Завдяки своїй архітектурі, він дозволяє не тільки

швидко реалізовувати проєкти, а й легко підтримувати їх у майбутньому. Фреймворк має чітку структуру, що спрощує роботу як для досвідчених розробників, так і для новачків.

Крім того, Laravel пропонує розширену підтримку автоматизації, що допомагає оптимізувати процес розгортання додатків та управління залежностями. Важливим аспектом є також наявність зручної документації та безлічі онлайн-курсів, що значно полегшує навчання та освоєння фреймворку.

Laravel, як один з найпопулярніших PHP-фреймворків у світі, вирізняється своєю гнучкістю, зрозумілим синтаксисом, добре структурованою архітектурою та потужною екосистемою. Однією з головних причин вибору саме Laravel стала його комплексність: він надає не лише зручні інструменти для реалізації CRUD-операцій, а й готові рішення для таких задач як маршрутизація, автентифікація, робота з чергами, планування завдань (scheduler), кешування, управління сесіями, взаємодія з базами даних тощо.

Крім того, Laravel дотримується архітектурної моделі MVC (Model-View-Controller), яка сприяє кращій організації коду, його підтримуваності та розширюваності. Це особливо важливо у випадку CRM-систем, де кожен компонент повинен бути чітко розмежованим, а логіка взаємодії – легко зрозумілою як для розробника, так і для тестувальника.

Окремої уваги заслуговує ORM Eloquent – власна реалізація об’єктно-реляційного відображення у Laravel, яка забезпечує зручну, читабельну і водночас потужну роботу з базами даних. Завдяки Eloquent, створення, оновлення або видалення записів у таблицях MySQL стає інтуїтивно зрозумілим і максимально наближеним до логіки роботи з об’єктами в PHP.

Laravel має глибоку інтеграцію з Composer [22], що дозволяє легко підключати сторонні пакети та бібліотеки. Крім того, фреймворк має власну CLI-утиліту – Artisan, яка дозволяє швидко генерувати шаблони моделей, контролерів, міграцій, запускати тести та планувати регулярні задачі. Це суттєво прискорює процес розробки і зменшує ймовірність помилок при створенні нових елементів архітектури.

Ще одним вагомим аргументом на користь Laravel є активна підтримка спільноти: тисячі статей, офіційна документація, регулярні оновлення, велика кількість навчальних курсів. Це дозволяє як швидко навчатися новим аспектам фреймворку, так і оперативно вирішувати проблеми, що виникають у процесі розробки.

Також, завдяки підтримці Laravel Echo, Pusher та WebSockets, легко реалізується функціонал реального часу – наприклад, сповіщення про події або повідомлення у чатах. Це особливо корисно для CRM, де оперативне інформування користувачів є критично важливим.

Laravel забезпечує високий рівень безпеки: захист від CSRF (підробки міжсайтових запитів), XSS (міжсайтового скриптингу), SQL-ін'єкцій, підтримка хешування паролів, перевірка прав доступу. Це дозволяє без зайвих зусиль реалізувати надійні механізми автентифікації та авторизації на рівні API або вебінтерфейсу.

Окремо слід зазначити, що Laravel чудово підходить як для реалізації REST API, так і для повноцінних вебзастосунків із серверним рендерингом. Це забезпечує гнучкість в подальшій еволюції проєкту – наприклад, можливість переходу до SPA (Single Page Application) або PWA (Progressive Web App) з мінімальними змінами в бекенді.

У цьому проєкті було використано також JWT [31] для реалізації механізму токенної автентифікації, що забезпечує безпечну взаємодію між сервером і клієнтською частиною, зокрема, у мобільних чи фронтенд-додатках, які працюють через API.

Таким чином, у процесі всебічного аналізу, Laravel показав себе як універсальний, потужний, але при цьому зручний для освоєння фреймворк, що дозволяє створювати сучасні вебсистеми, які відповідають вимогам до швидкодії, безпеки, підтримки та масштабованості. Laravel поєднує в собі простоту, гнучкість, продуктивність і широкий набір функцій, що робить його ідеальним вибором для створення масштабованих і надійних вебзастосунків. Саме ці характеристики стали вирішальними під час вибору інструментів

реалізації серверної частини CRM-системи у межах даної кваліфікаційної роботи.

У процесі реалізації важливо враховувати сумісність технологій. Вибраний стек – PHP, Laravel, MySQL, JavaScript, HTML, CSS – забезпечує гнучкість, високу продуктивність і велику кількість документації та підтримки. Laravel, як сучасний фреймворк, має вбудовані механізми автентифікації, маршрутизації, міграцій, що значно скорочує час розробки.

IDE PhpStorm дає змогу ефективно працювати з кодом, завдяки розширеним інструментам підсвітки, автодоповнення та інтеграції з системами контролю версій. Використання Composer як менеджера пакетів дозволяє підключати додаткові бібліотеки, а Laravel Artisan – автоматизувати типові завдання.

Уся система буде розроблятись з урахуванням принципів DRY (Don’t Repeat Yourself) та KISS (Keep It Simple, Stupid), що підвищує читабельність та якість коду. Розгортання на локальному сервері XAMPP дозволить здійснити всі етапи тестування до повного запуску системи в продуктивне середовище.

2.3 Проектування структути даних та моделі бази даних

Оскільки ми використовуємо PHP-фреймворк Laravel, у ньому реалізовано архітектурний патерн MVC (Model-View-Controller). Це дозволяє чітко розділити бізнес-логіку, роботу з базою даних і відображення даних у вебзастосунку.

Застосування архітектурного патерну MVC у поєднанні з модульним підходом дає змогу реалізувати масштабовану та добре структуровану архітектуру системи, де кожен функціональний блок є самодостатнім. Модульність забезпечує високий рівень ізоляції: кожен модуль відповідає лише за одну область відповідальності і може розвиватися незалежно від решти системи. Такий підхід не тільки знижує зв’язаність коду, але й підвищує зручність супроводу та модифікації програмного забезпечення.

У межах даного проєкту реалізовано окремі модулі для таких блоків системи, як управління користувачами, управління подіями, проєктами,

зустрічами, статистикою активності, а також спеціальними днями (відпустки, лікарняні, святкові дні). Кожен модуль має власний набір моделей, контролерів, сервісів, маршрутів, шаблонів представлення, що логічно розділяє бізнес-логіку та дозволяє уникати дублювання коду. При потребі, такі модулі можна повторно використати або виносити в окремі пакети.

Особливо важливою є логічна взаємодія між модулями. Наприклад, модуль керування користувачами інтегрується з модулем ролей і прав доступу, що дає змогу гнучко керувати дозволами в системі. Така гнучкість дає змогу впроваджувати різноманітні сценарії доступу без необхідності переписувати код ядра системи. Всі перевірки ролей та прав виконуються централізовано за допомогою middleware, що покращує безпеку.

Що стосується проєктування бази даних, то важливу роль відіграє початкове планування та нормалізація. Для цього було створено інфологічну модель, що описує сутності, зв’язки між ними, типи атрибутів та обмеження. Наприклад, таблиця користувачів users пов’язана з таблицями завдань, зустрічей, подій та днів відпусток. Це дозволяє формувати глибоку аналітику щодо активності працівників, планування навантаження та управління робочим часом.

Всі сутності пройшли етап нормалізації до третьої нормальної форми (3НФ), що дозволяє уникнути надмірності даних і забезпечує узгодженість. Було враховано важливі принципи проектування: забезпечення унікальності даних, мінімізація дублювання, підтримка цілісності за допомогою зовнішніх ключів, реалізація каскадних дій під час оновлення та видалення записів. Відповідно, наприклад, поле user_id в таблиці special_days є зовнішнім ключем до таблиці users, а поле approved_by_user_id дозволяє відстежувати, хто саме затвердив кожен запис.

З метою підвищення безпеки та відстеження змін у системі реалізовано систему логування. Таблиці logs та audit_trails містять інформацію про дії користувачів, із вказанням дати, часу, типу операції та залучених сутностей. Це дозволяє у майбутньому швидко виявляти підозрілі дії, відстежувати зміни та забезпечувати відповідність політикам безпеки.

Окрему увагу приділено обробці HTTP-запитів. Laravel пропонує зручну маршрутизацію, що дозволяє легко організувати обробку REST-запитів. Всі маршрути згруповані за модулями та мають зрозумілу семантику (/users, /events, /projects). Використання middleware дає змогу додатково перевіряти автентичність користувача, його роль, токени доступу та інші умови перед обробкою запиту.

На рівні контролерів реалізовано делегування бізнес-логіки до окремих сервісних класів. Такий підхід дозволяє спростити контролери, зробити їх більш зрозумілими та тестованими. Сервіси, у свою чергу, взаємодіють з репозиторіями, що реалізують доступ до бази даних через ORM Eloquent. Наприклад, сервіс створення події не лише формує новий запис, а й може викликати супутні функції – наприклад, надсилання сповіщень або оновлення статистики активності.

Усе це дозволяє досягти високого рівня масштабованості, адже з додаванням нових функцій не потрібно змінювати вже існуючі модулі. Кожна нова функціональність реалізується у вигляді окремого модуля, з чітко визначеним інтерфейсом. Такий підхід не лише відповідає сучасним принципам розробки, а й дозволяє підтримувати систему в довгостроковій перспективі з мінімальними витратами.

У межах проєктування системи важливою характеристикою є модульність. У проєкті застосовано модульний підхід, за якого кожен блок функціональності ізольований у власному просторі імен, має окремі моделі, контролери, міграції та роутинг. Такий підхід не лише підвищує читабельність та розширюваність коду, але й дозволяє в майбутньому адаптувати або повторно використовувати частини системи для інших проєктів або підсистем.

Наприклад, окремий модуль може відповідати лише за управління користувачами: створення, редагування, блокування, призначення ролей тощо. Інший – за CRM функціональність: взаємодію з контактами, створення записів активності, додавання замовлень, генерування статистики. Завдяки такому підходу значно полегшується відлагодження системи та внесення змін.

Для взаємодії з базою даних необхідно створити моделі та контролери для ключових сутностей: users, projects, special_days, meets, events. Це дозволить коректно обробляти запити до сервера та забезпечувати зв'язок між клієнтською і серверною частинами.

Проектування бази даних

База даних міститиме всю необхідну інформацію. Головною сутністю є користувач, тому першочергово створимо таблицю users зі структурою:

– id (ідентифікатор);
– name (ім'я);
– password (пароль);
– image (аватар);
– email (унікальний);
– gender (стать);
– birth_date (дата народження).

Унікальність деяких полів, наприклад email, необхідна для коректної аутентифікації та функціональності, пов’язаної з відновленням пароля.

Нормалізація бази даних

Для забезпечення ефективності створимо окремі таблиці для special_days, projects, events, meets. На прикладі special_days розглянемо процес нормалізації.

Початковий вигляд таблиці:

– id (ідентифікатор);
– from (дата початку);
– to (дата завершення);
– type (тип події);
– user_id (користувач);
– approved_by_user (хто затвердив).

Видно, що поля type, user_id і approved_by_user можуть повторюватися.

Щоб уникнути дублювання даних, винесемо їх у окремі таблиці:

– type_id – зв’язок із special_day_type;
– user_id – посилання на users;
– approved_by_user_id – посилання на users.

Такий підхід застосуємо й до інших сутностей, що забезпечить швидкодію та правильну роботу системи. Повна структура представлена на інфологічній схемі (Додаток А).

Основні сторінки вебзастосунку

На основі моделювання бази визначимо ключові сторінки інтерфейсу:

– dashboard;
– users;
– special_days;
– events;
– projects;
– meets.

Логіка взаємодії

– Аутентифікація: необхідно перевіряти, чи існує користувач при вході в систему;
– Ролі користувачів: розділення прав адміністратора і звичайного користувача. Користувач може переглядати інформацію та створювати запити, а адміністратор має повний контроль над даними.

Після реалізації цих заходів:

– Неавторизований користувач не зможе отримати доступ до сервісу, оскільки механізми автентифікації перевіряють особу кожного користувача перед наданням доступу. Це значно знижує ризик несанкціонованого використання системи або спроб отримати конфіденційну інформацію;
– Користувач не матиме можливості змінювати критично важливі дані або структуру застосунку.

Процес проектування бази даних починався з побудови інфологічної моделі, яка дозволила наочно представити зв’язки між сутностями. Особливу увагу було приділено нормалізації даних до третьої нормальної форми, що дозволило уникнути надлишковості та забезпечити логічну цілісність даних.

Ключовими сутностями виступають користувачі, замовлення, задачі, проєкти, повідомлення, роли, права доступу тощо. Всі вони пов’язані реляційними зв’язками, зокрема: один до багатьох, багато до багатьох. Наприклад, один користувач може мати багато завдань, а завдання – належати кільком категоріям.

Також була передбачена можливість ведення історії змін та логування дій користувачів, що важливо з точки зору безпеки та аудиту. Для цього додано таблиці logs, audit_trails, які дозволяють зберігати дії у форматі «користувач – дія – час».

Для забезпечення надійної роботи вебзастосунку необхідно також звернути увагу на внутрішні механізми обробки запитів та відповідей у системі. У рамках проєкту буде реалізовано багаторівневу структуру контролерів та сервісних класів, що дозволяє логічно розділити відповідальність між компонентами та спростити підтримку й масштабування проєкту.

Важливою складовою є обробка запитів користувачів, що надходять через HTTP. У межах Laravel фреймворку це реалізовано за допомогою маршрутизатора, який направляє запит до відповідного контролера. Контролери, у свою чергу, взаємодіють з сервісами або репозиторіями, що забезпечують доступ до бізнес-логіки програми або безпосередньо до бази даних через Eloquent ORM.

Наприклад, при створенні нового користувача контролер передає дані на обробку у відповідний сервіс, який виконує валідацію, перевіряє унікальність email-адреси, хешує пароль, а потім передає очищені дані до моделі User, яка зберігає інформацію у базі даних. Така логіка дозволяє легко масштабувати програму, додаючи нові перевірки або розширюючи функціонал без шкоди для загальної структури.

2.4 Висновки до другого розділу

У процесі розробки системи керування персоналом було здійснено детальний аналіз різних архітектурних підходів, з метою вибору найбільш відповідної моделі для ефективної реалізації. Клієнт-серверна архітектура була визначена як оптимальна завдяки своїй здатності централізовано управляти даними, забезпечувати високу продуктивність та ефективне використання апаратних ресурсів. Такий підхід дозволяє делегувати всю відповідальність за обробку даних серверу, знижуючи навантаження на клієнтські пристрої та мережу, що є критично важливим для систем із великою кількістю користувачів.

Клієнт-серверна модель також забезпечує високу масштабованість системи – вона дає змогу додавати нових користувачів без істотного зниження швидкодії, що відповідає потребам сучасних організацій, де одночасна робота сотень або тисяч співробітників є нормою. Централізоване управління даними дозволяє підтримувати їхню цілісність та консистентність, що особливо важливо у випадках, пов’язаних із обробкою конфіденційної інформації, наприклад, персональних даних співробітників.

Однією з ключових переваг обраної архітектури та підходів у проєктуванні є забезпечення високої адаптивності системи до майбутніх змін. Використання модульної структури та чіткого розмежування відповідальностей між компонентами дає змогу зручно інтегрувати нові функціональні можливості або зовнішні сервіси без необхідності кардинальних змін у вже існуючому коді. Така гнучкість особливо важлива в контексті сучасних інформаційних систем, які потребують швидкої адаптації до мінливих бізнес-вимог та технологічних трендів.

Крім того, при проєктуванні передбачено впровадження надійних механізмів безпеки, які гарантуватимуть захист даних користувачів і контроль доступу на основі ролей. Це дозволить забезпечити безперебійну роботу системи в багатокористувацькому середовищі, зменшить ризики несанкціонованого доступу та підвищить довіру користувачів до інформаційних ресурсів.

Врахування цих аспектів на стадії проєктування закладає основу для подальшої успішної експлуатації та розвитку системи.

Паралельно було проведено глибокий огляд сучасних серверних фреймворків, які можуть бути використані для реалізації бекенд-частини системи. Вибір фреймворку — це критично важливий етап, адже він визначає подальшу ефективність розробки, безпеку, продуктивність та можливості масштабування рішення. Розглянуто такі популярні фреймворки, як Laravel (PHP), Django (Python), Express.js (Node.js), Spring Boot (Java), ASP.NET Core (C#) та Ruby on Rails (Ruby). Кожен із них має свої переваги і недоліки, зумовлені особливостями мови програмування, архітектурними рішеннями, активністю спільноти розробників, а також підтримкою сторонніх бібліотек і інструментів.

Laravel було обрано з урахуванням таких переваг:

– простота та зручність розробки – синтаксис є інтуїтивно зрозумілим і дозволяє розробникам швидко писати чистий та підтримуваний код;
– вбудовані засоби безпеки – Laravel має механізми захисту від найбільш поширених атак (SQL-ін’єкції, CSRF, XSS), що є критичним для систем, що працюють із персональними даними відповідно до рекомендацій OWASP Top-10 [24];
– потужна ORM Eloquent – дозволяє ефективно працювати з базою даних, спрощує складні запити та забезпечує високий рівень абстракції, що підвищує продуктивність розробки;
– масштабованість і модульність – фреймворк підтримує побудову додатків на основі модулів, що полегшує підтримку, тестування [28] і подальший розвиток системи;
– підтримка API та реального часу – інтеграція з WebSockets і механізмами Laravel Echo дає змогу впроваджувати сучасні функції, такі як сповіщення та чат;
– велика спільнота – активна підтримка розробників, численні офіційні і сторонні пакети, документація і онлайн-ресурси значно спрощують вирішення поточних завдань.

Крім того, Laravel відзначається високою гнучкістю, що дає змогу адаптувати систему під специфічні вимоги замовника, а також забезпечує простоту інтеграції з іншими системами та сервісами, що важливо для майбутнього масштабування і розвитку.

У межах розробки особливу увагу було приділено організації структури бази даних. Вона була спроектована відповідно до класичних принципів нормалізації до третьої нормальної форми, що дозволило уникнути надлишковості даних, забезпечити їхню узгодженість та цілісність. В процесі проектування визначено основні сутності системи: користувачі, проєкти, події, замовлення, спеціальні дні та інші. Для кожної сутності було створено окремі таблиці з чіткими зв’язками, що відображають реальні бізнес-процеси. Особлива увага була приділена захисту персональних даних, що забезпечує відповідність системи сучасним нормативним вимогам з безпеки.

Застосування архітектурного патерну MVC в рамках Laravel дозволило розділити функціональність на три основні компоненти: модель (робота з даними), представлення (інтерфейс користувача) та контролер (логіка обробки запитів). Такий підхід забезпечує високу гнучкість, полегшує тестування, супровід і масштабування системи. Це особливо важливо для комплексних систем, які мають постійно розвиватися і адаптуватися до нових бізнес-вимог.

Важливою складовою безпеки є розмежування прав доступу на основі ролей. Впровадження системи автентифікації і авторизації гарантує, що користувачі отримують доступ лише до тих ресурсів, на які мають право, що значно підвищує загальний рівень захисту системи.

Також, завдяки підтримці REST API у Laravel, забезпечується можливість легкої інтеграції із зовнішніми сервісами та мобільними додатками, що є сучасною необхідністю для корпоративних інформаційних систем. Це розширює функціональні можливості системи і робить її більш адаптивною.

Проєкт було реалізовано із дотриманням принципів чистої архітектури, з акцентом на розробку модульних і повторно використовуваних компонентів. Такий підхід сприяє підвищенню якості коду, його зручності для підтримки та

полегшує розширення системи у майбутньому без необхідності глобальних переробок.

Використання сучасних інструментів розробки, таких як середовище PhpStorm, Composer та Laravel Artisan, підвищує продуктивність команди розробників, автоматизуючи рутинні завдання та сприяючи якісній організації коду.

Перспективи розвитку системи включають впровадження мікросервісної архітектури, розширення функціоналу за рахунок інтеграції із зовнішніми API, впровадження аналітичних та звітних модулів на основі сучасних BI-рішень, а також застосування технологій штучного інтелекту для автоматизації рутинних процесів.

Загалом, реалізована система є стабільним, безпечним і масштабованим рішенням для управління персоналом, яке відповідає високим вимогам замовника і сучасним стандартам розробки вебзастосунків. Вона готова до подальшого розвитку, інтеграції нових функцій та адаптації під змінні бізнес-потреби.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Розробка програмних модулів

Цей проєкт реалізовуватиметься з використанням мови програмування PHP у зв'язці з фреймворком Laravel [6], а також з використанням JavaScript [11], HTML [12] і CSS [13]. Усі розробки здійснюватимуться в середовищі PhpStorm, яке, на мою думку, є одним із найзручніших інструментів для створення вебзастосунків: воно підтримує широкий спектр мов і надає зручний інтерфейс користувача. Контейнеризація буде здійснена за допомогою сервісу Docker [15].

Для розробки серверної частини додатку буде використано мови PHP та MySQL. PHP [5] (скорочення від PHP: Hypertext Preprocessor) – це популярна скриптова мова з відкритим кодом, яка здебільшого використовується у веброзробці. PHP дозволяє вбудовувати програмний код безпосередньо в HTML, що значно спрощує процес створення динамічних сторінок.

На відміну від таких мов, як Perl [14] чи C, де HTML генерується шляхом виведення через команди, PHP дозволяє використовувати вбудовану розмітку з програмною логікою всередині, позначену тегами <?php ... ?>.

Код PHP обробляється на сервері, а клієнту надсилається вже згенерований HTML-код, що приховує сам факт використання серверного скрипта. PHP є досить легкою для вивчення мовою, але водночас вона дозволяє створювати складні рішення завдяки своїм широким можливостям. Ця мова підтримується великою кількістю хостинг-провайдерів і активно використовується у професійному середовищі.

Кодування здійснюватиметься в PhpStorm [8] (рисунок 3.1), потужному інтегрованому середовищі розробки. Воно має вбудований редактор для MySQL з графічним інтерфейсом та підтримкою численних плагінів, що додатково полегшують процес написання коду.

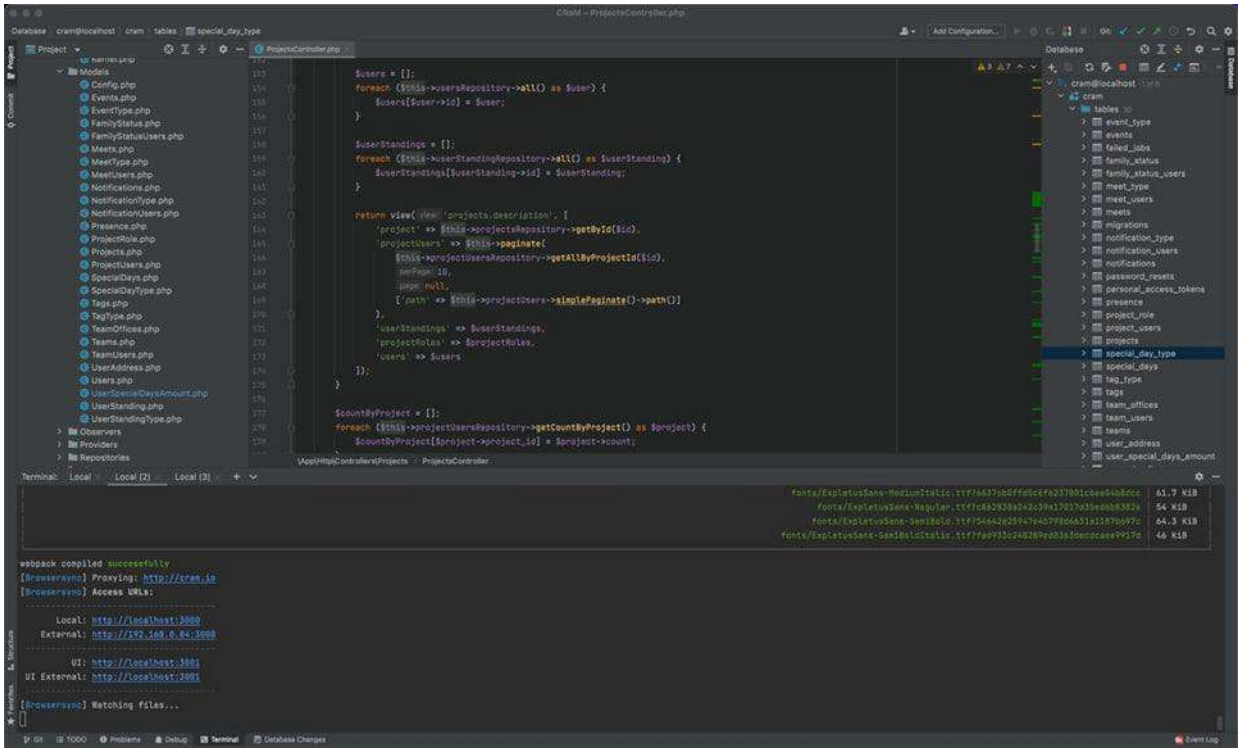


Рис. 3.1 – Інтерфейс середовища PhpStorm

Усі ці інструменти мають сприяти ефективній розробці об’єктно-орієнтованого програмного забезпечення, мінімізуючи кількість логічних помилок. Для досягнення максимальної відповідності функціоналу проєкту його цілям, доцільно також розглянути приклади аналогічних CRM-систем.

Для запуску вебзастосунку локально застосовуватиметься XAMPP, оскільки він дозволяє створювати кросплатформенні середовища розробки без додаткових налаштувань.

XAMPP [7] – це готовий набір програмного забезпечення, до складу якого входять інструменти, необхідні для роботи вебсайту: вебсервер, база даних, інтерпретатори мов програмування. Такий підхід дозволяє створювати й тестувати вебзастосунки на локальному комп’ютері без підключення до Інтернету чи наявності хостингу. XAMPP підтримується основними операційними системами: Windows [16], Linux [17], Mac OS [18].

Розшифровка абревіатури XAMPP:

– X – сумісність з усіма основними ОС;
– Apache – вебсервер із розширеними налаштуваннями;
– MySQL – популярна СУБД;
– PHP – серверна мова програмування, підтримувана більшістю хостингів;
– Perl – мова з підтримкою сторонніх модулів.

У проєкті також реалізовано підхід MVC [34] – це архітектурний шаблон, який розділяє структуру вебзастосунку на три частини: модель, представлення і контролер.

Модель / Model – являє собою об'єктну структуру певної галузі, що включає в себе дані та методи для взаємодії з ними. Вона реагує на запити з боку контролера, повертаючи необхідну інформацію або змінюючи свій стан. При цьому модель не містить відомостей про способи відображення інформації або формати їх подання, і не здійснює прямої взаємодії з користувачем. Вона виконує роль джерела даних і обробника логіки, не залежного від інтерфейсу.

Вид / View – відповідає за візуальне подання інформації. Одні й ті самі дані можуть бути показані різними способами й у різних форматах. Наприклад, набір об’єктів може бути представлений як таблиця або як список у користувацькому інтерфейсі, або експортуватися через API у форматах JSON, XML чи XLSX. Таким чином, представлення фокусується виключно на виведенні інформації, не змінюючи її зміст.

Контролер / Controller – виконує роль зв’язуючого елементу між користувачем і системою. Він здійснює обробку запитів, ініціює виклики до моделі та передає результат до представлення. Крім того, у контролері зазвичай реалізуються такі функції, як перевірка прав доступу, валідація вхідних даних, а також маршрутизація запитів. Для реалізації маршрутизації використано бібліотеку React Router [32]. Завдяки цьому контролер координує роботу всієї системи, забезпечуючи правильну реакцію на дії користувача.

Такий поділ дозволяє ізолювати бізнес-логіку від візуальної частини інтерфейсу, що спрощує підтримку й масштабування коду. Модель (Model) відповідає за дані й логіку взаємодії з ними, представлення (View) – за виведення інформації, а контролер (Controller) – за обробку дій користувача й взаємодію з іншими компонентами.

Загальна структура зображена нижче (рисунок 3.2)

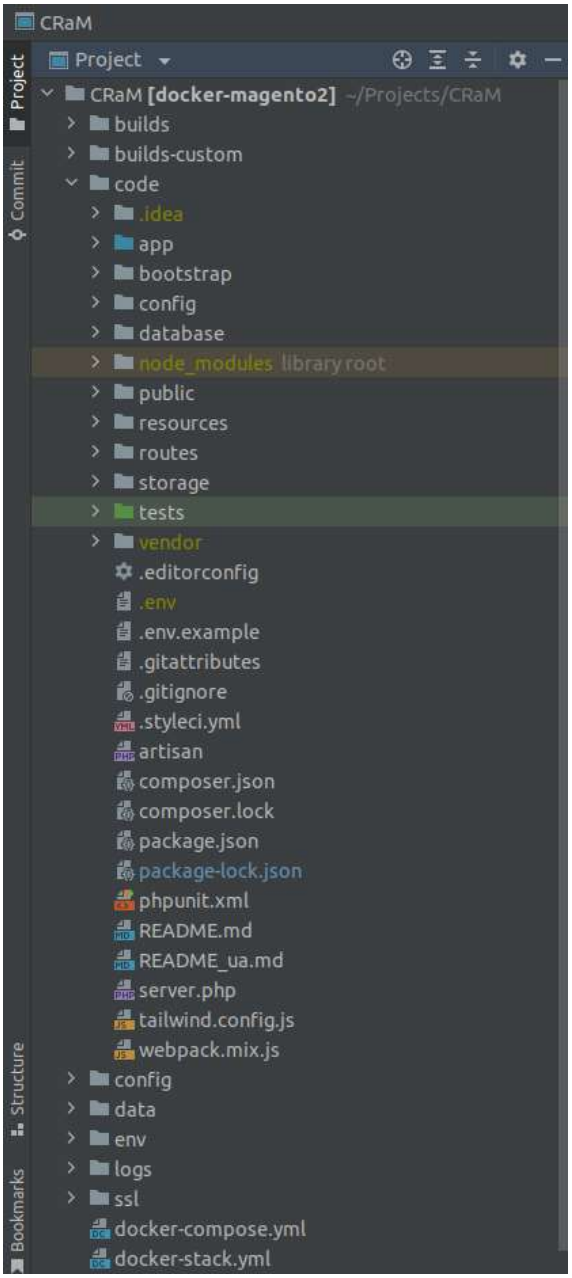


Рисунок 3.2 – Загальна структура проєкту

3.2 Розробка бази даних

MySQL [4] – це система адміністрування реляційних баз даних (RDBMS), створена компанією Oracle. Вона ґрунтується на мові SQL – мові структурованих запитів.

База даних – це впорядкований набір інформації, який може містити будь-які відомості: від простого списку покупок до великої колекції зображень або централізованого сховища об’ємних масивів даних у корпоративній IT-мережі. Реляційна база даних – це цифрове середовище, в якому інформація організована відповідно до реляційного підходу: у вигляді таблиць із чітко визначеними рядками та колонками. Зв’язки між елементами будуються за логічними правилами.

СУБД – це програмний інструментарій, який дозволяє реалізовувати структуру бази, керувати нею та взаємодіяти через запити.

MySQL входить до складу багатьох популярних технологічних стеків, які використовуються як у клієнтських вебсервісах, так і у потужних корпоративних системах типу B2B. Відкрите походження, функціональна насиченість і регулярна підтримка з боку Oracle зробили MySQL вибором таких компаній, як Facebook, Flickr, Twitter, Wikipedia і YouTube.

Серед сильних сторін MySQL – високий рівень безпеки: усі дані надійно захищені паролями, що шифруються за сучасними криптографічними алгоритмами. Для з’єднання з сервером використовуються різні механізми: TCP/IP, UNIX сокети або іменовані канали.

Одним із початкових етапів проєктування бази є нормалізація таблиць. З технічної точки зору, перед тим потрібно визначити, яка інформація буде зберігатися в базі, які поля вона включатиме, які типи даних застосовуватимуться та яка буде їхня розмірність. Але це, як правило, приймається за замовчуванням.

Нормалізація – це теоретичний підхід до структурування реляційних баз даних, що сформувався наприкінці 1970-х років. У рамках цієї концепції виокремлюють шість нормальних форм: першу, другу, третю, четверту, п’яту, а

також форму Бойса-Кодда, що знаходиться між третьою і четвертою. Базу вважають нормалізованою, якщо більшість її таблиць знаходиться принаймні у третій нормальній формі. Часто таблиці доводять до четвертої форми, але в окремих випадках навпаки проводять денормалізацію. П’ята форма в практиці зустрічається рідко.

Основною метою нормалізації є усунення надлишкових даних і повторення інформації. Ідеальним вважається варіант, коли кожне значення в базі фігурує тільки в одному екземплярі, й не може бути обчислено з інших збережених значень.

Після ознайомлення з теоретичними засадами баз даних та аналізу ключових компонентів майбутньої системи можна переходити до створення самої бази у контейнері mysql. Це забезпечить зберігання даних безпосередньо на сервері замість використання розрізнених файлів.

Далі база буде підключена до графічного середовища розробки, оскільки такий підхід значно полегшує взаємодію з інформаційною структурою системи. (рисунок 3.3)

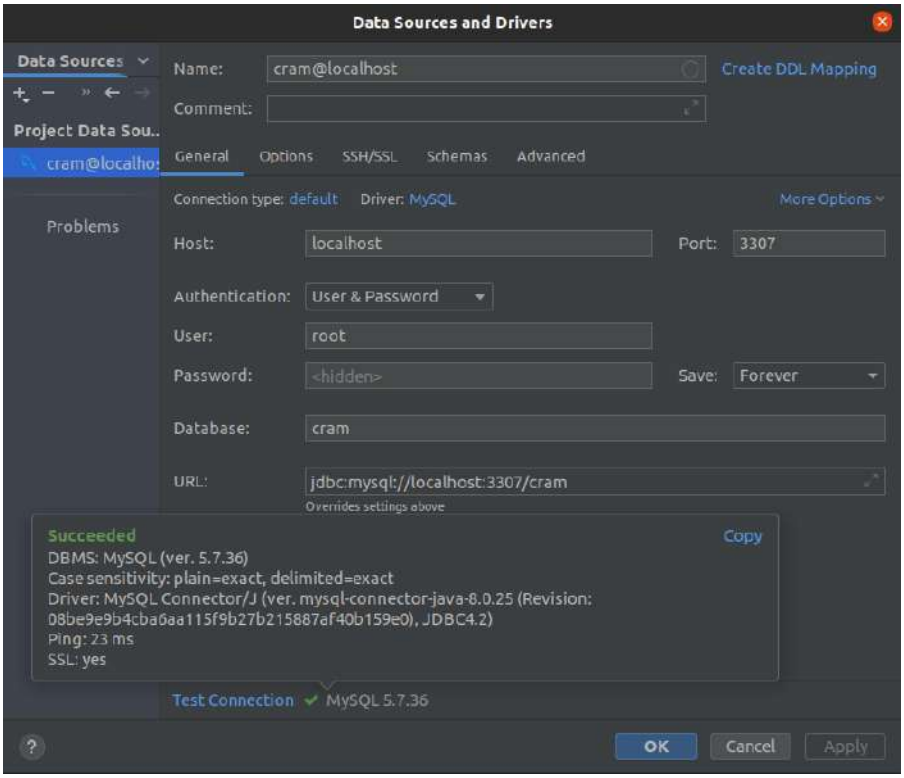


Рисунок 3.3 – Підключення бази даних до візуального серидовища розробки

3.3 Інструкція з інсталяції розробленого проєкту

Для початку роботи з програмним забезпеченням «Customer Relationships and Management» необхідно виконати кілька підготовчих кроків. Насамперед користувачу слід перейти за посиланням https://cram.io/, де розміщено вебінтерфейс системи. Однак перед тим, як відкривати сервіс у браузері, обов'язково потрібно попередньо запустити конфігураційний файл Docker. Це необхідно для активації локального серверного середовища, яке забезпечуватиме коректну роботу системи на комп'ютері користувача (рисунок 3.4).

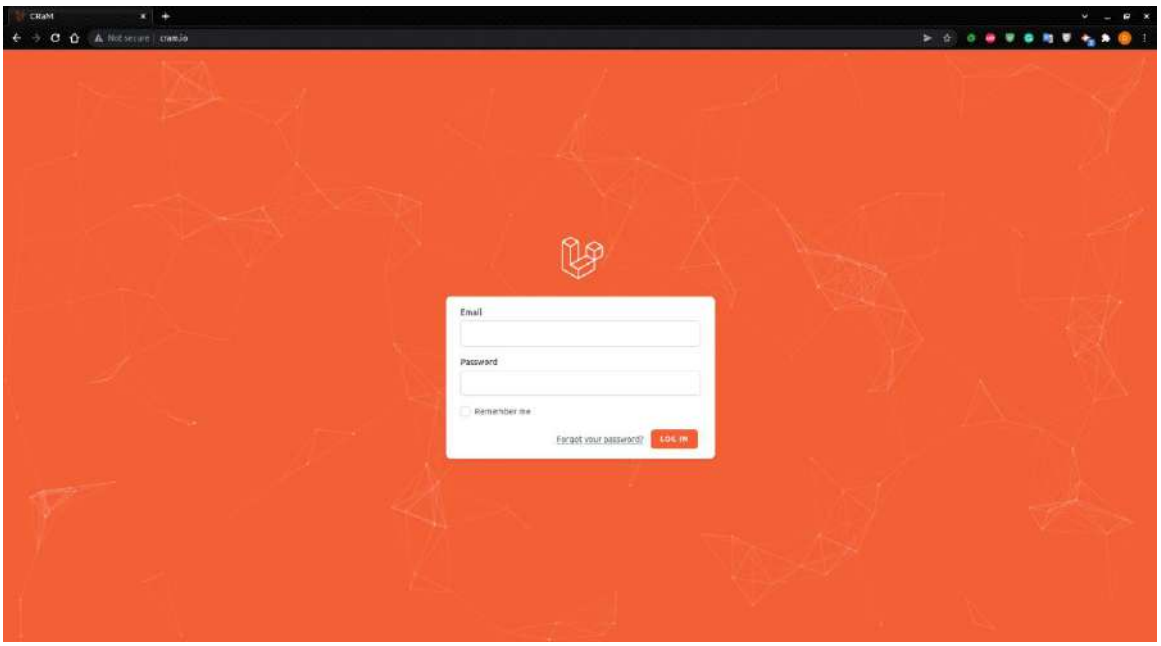


Рисунок 3.4 – Вигляд початкової сторінки CRaM

Система спроєктована таким чином, щоб бути кросплатформною, тобто сумісною з усіма основними типами операційних систем. Вона підтримує роботу на персональних комп'ютерах із встановленими Windows, Linux та MacOS, а також на мобільних пристроях, що працюють під управлінням операційних систем iOS та Android [20]. Такий підхід забезпечує гнучкість у використанні та дозволяє взаємодіяти з системою з будь-якого пристрою, незалежно від його технічних характеристик або операційної платформи.

3.4 Керівництво користувача

Після того, як був здійснений перехід за посиланням, потрібно залогінитись, адже сам користувач зареєструватись не може, оскільки потрібно, щоб спершу акаунт створили адміни. Після успішного входу перекидує на Dashboard (рисунок 3.5).

Рисунок 3.5 – Сторінка Dashboard

Далі можна починати роботу з іншими сутностями, наприклад переглянути свій профіль на сторінці користувача (рисунок 3.6).

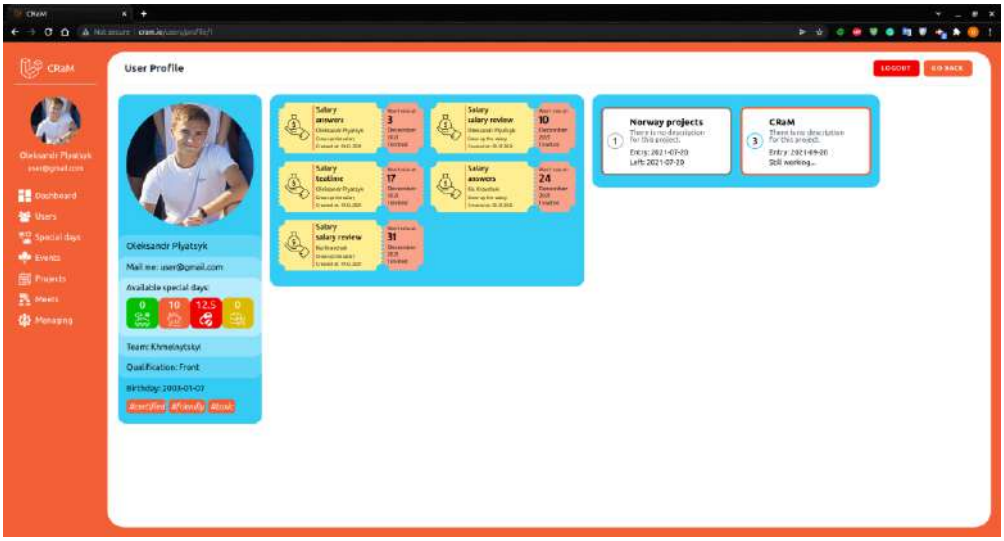


Рисунок 3.6 – Сторінка User Profile

Переглянути статистику своїх особливих днів та своїх співробітників (рисунок 3.7).

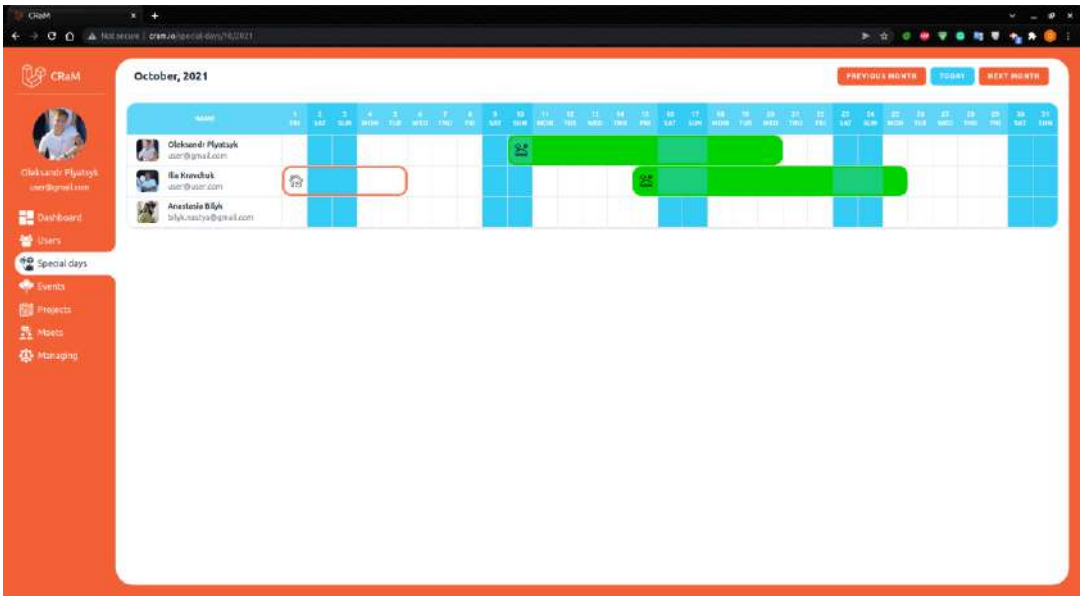


Рисунок 3.7 – Сторінка Special Days

Також переглянути стан та інформацію про проєкти, якими займається фірма (рисунок 3.8).

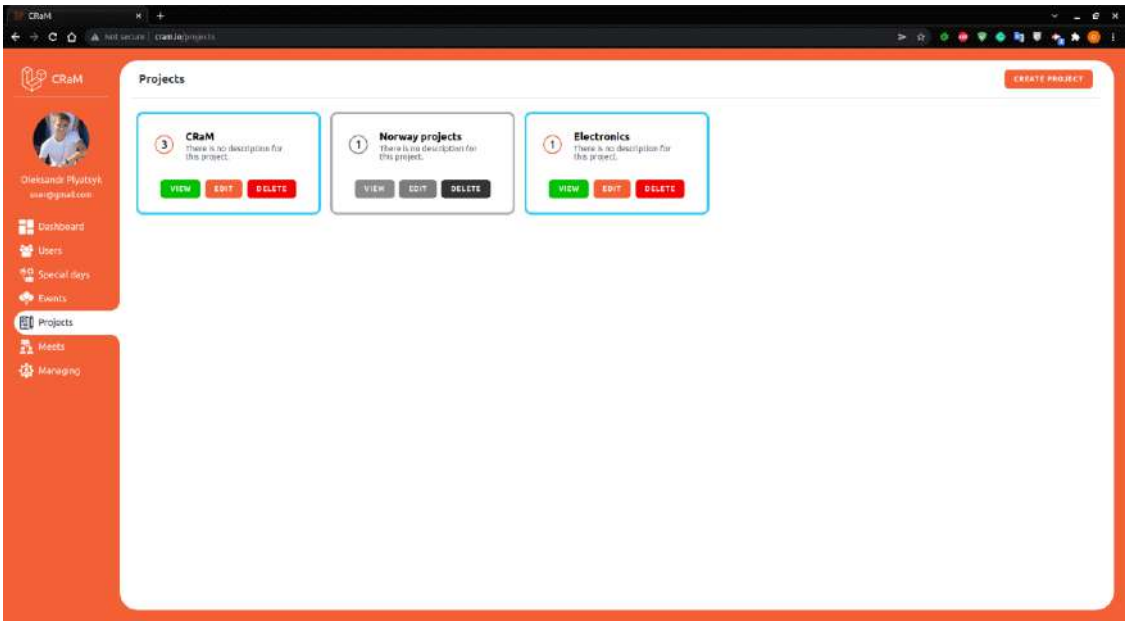


Рисунок 3.8 – Сторінка Projects

Для звичайного користувача інтерфейс буде трохи дещо відрізнятись (рисунки 3.9 – 3.11)

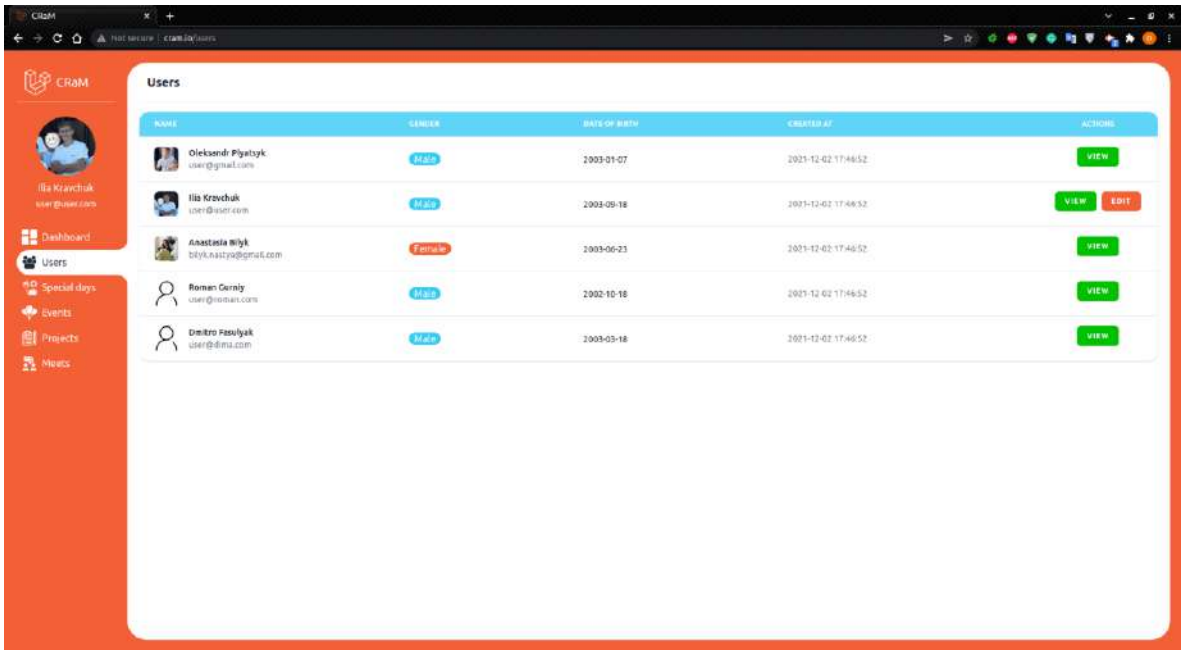


Рисунок 3.9 – Сторінка користувачів для звичайного користувача

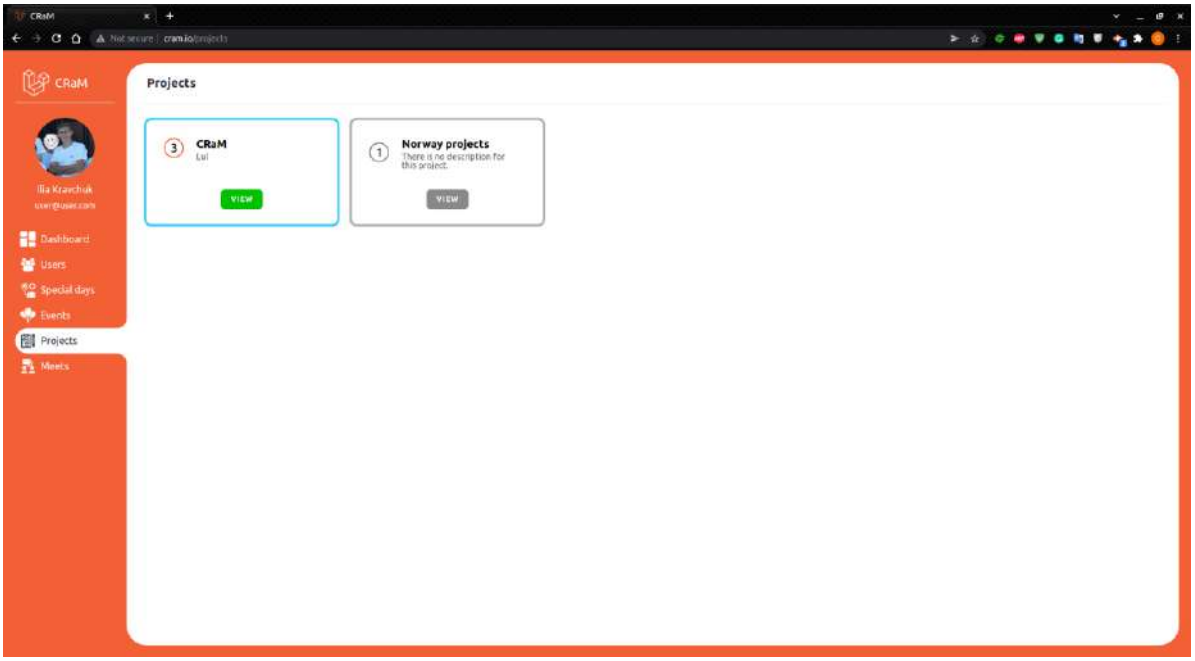


Рисунок 3.10 – Сторінка проєктів для звичайного користувача

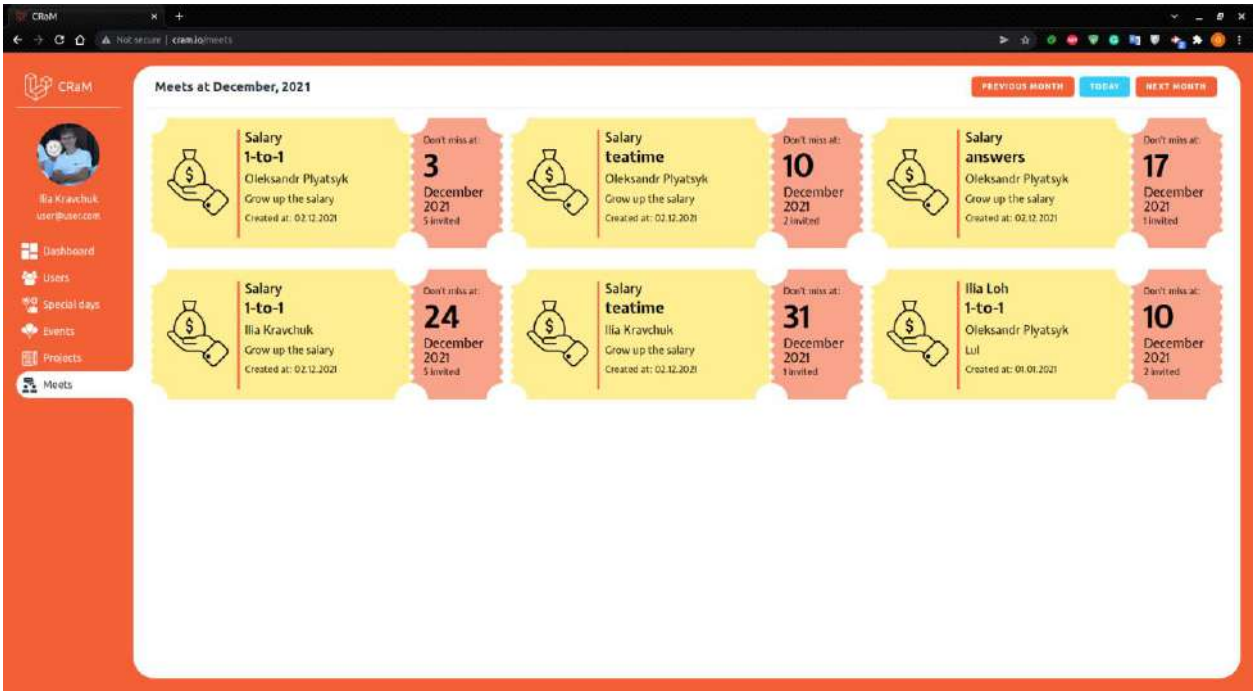


Рисунок 3.11 – Сторінка зустрічей для звичайного користувача

Проєкт створювався із застосуванням системи контролю версій Git [9]. Система контролю змін – це програмне забезпечення, яке фіксує всі редагування файлу або групи файлів протягом визначеного періоду. Вона дозволяє відкотити певні файли до попереднього стану, повернути весь застосунок до колишньої версії, переглянути історію змін, визначити, хто саме вносив правки, коли саме це відбувалося та чи викликали вони помилки або проблеми.

Завдяки використанню СКВ можна легко відновити дані у разі їх пошкодження або втрати. Крім того, така система має низькі витрати на впровадження й обслуговування. Вона також дає змогу розробникам ефективно обмінюватися кодом, перевіряти зміни одне одного та значно пришвидшувати впровадження нового функціоналу.

Оскільки розробка цього проєкту здійснювалася поетапно, кожен реліз або ревізія зберігається в репозиторії. Це дає змогу будь-кому переглянути хронологію змін, доданих можливостей та виправлення знайдених недоліків.

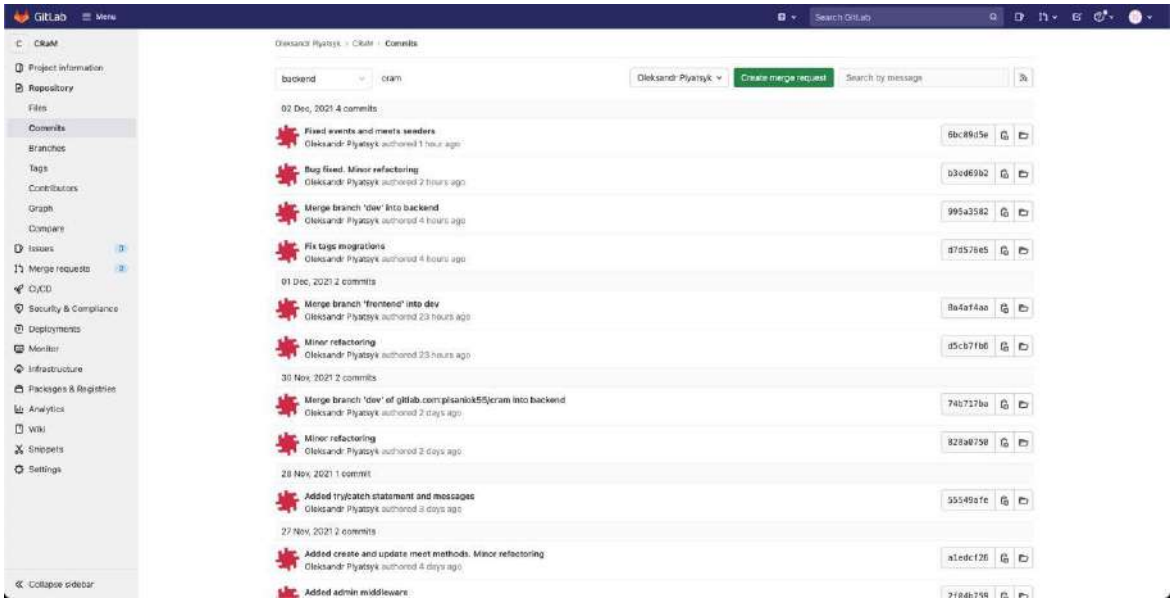


Рисунок 3.12 – Вигляд процесу розвитку програми, засобами PhpStorm

Використання цього проєкту є легким і інтуїтивно зрозумілим. Для зручного доступу до всіх наявних параметрів конфігурації був розроблений окремий лаунчер. Водночас, для правильного додавання нового функціоналу необхідне базове розуміння основних принципів архітектури самого застосунку.

3.5 Тестування вебзастосунку

Після завершення етапу проєктування настає стадія реалізації, тобто написання програмного коду. Паралельно з розробкою відбувається перевірка окремих частин системи та їх інтеграція. Це одна з найскладніших фаз, оскільки під час неї виникає чимало труднощів, пов’язаних із особливостями середовища розробки, використовуваних мов і бібліотек.

Далі виконується повноцінне тестування застосунку, що враховує всі можливі сценарії взаємодії з ним.

Функціональне тестування орієнтоване на відповідність функцій вимогам бізнесу. Його основна ціль – переконатися, що кожна частина системи виконує саме ті завдання, які були передбачені технічною документацією. У цьому підході нас цікавить кінцевий результат, а не внутрішня реалізація процесу. Це аналогія з методикою «чорного ящика», коли оцінюється лише реакція системи на вхідні дані.

Нефункціональне тестування зосереджується на додаткових характеристиках, таких як швидкодія, зручність використання, стійкість до навантажень тощо.

Обидва види перевірки є однаково важливими, оскільки дозволяють глибше зрозуміти особливості системи та гарантувати високу якість програмного продукту для кінцевого користувача.

Після завершення тестування та усунення виявлених помилок система передається у впровадження, а також створюються довідкові матеріали та інструкції для користувачів.

Підтримка проєкту продовжується і після запуску: його необхідно супроводжувати, оновлювати та вдосконалювати.

Застосування засобів перевірки вимог – це не щось незвичайне чи складне. Ці інструменти є звичними для спеціалістів, і їх використання є необхідною умовою досягнення цілей у межах проєктів.

Повнота вимог – один з головних критеріїв якості, на який спрямована більшість перевірок. Адже саме від цього залежить обсяг робіт (scope): замовник повинен бути впевнений, що отримає усе необхідне для реалізації своїх бізнес-цілей. Так само й розробник має знати, що нічого не було упущено, і згодом не виникне спірних ситуацій щодо очікуваного функціоналу.

Для ефективної перевірки вимог важливо залучати фахівців із різних напрямів. Також документацію API створено з використанням специфікації OpenAPI (Swagger [26]).

Одним із підходів є аналіз поведінки системи, коли вимоги формалізуються у вигляді сценаріїв: «вхід – результат», «подія – реакція», «умова – дія». Найчастіше це оформлюється у вигляді тест-кейсів (створюються тестувальниками) або юз-кейсів (розробляються аналітиками).

Цей підхід має свої переваги й недоліки.

Плюси:

– добре перевіряє повноту, чіткість і однозначність вимог;
– дозволяє виявити «підводні камені» та неоднозначності ще до початку кодування;
– на основі таких сценаріїв легко програмувати та тестувати функціонал.

Мінуси:

– підготовка сценаріїв вимагає багато часу та спеціальних навичок;
– не кожен співробітник зможе ефективно працювати з таким форматом вимог без попередньої підготовки.

Таким чином, якісне тестування – важлива частина життєвого циклу розробки, яка забезпечує надійність, коректність і відповідність програмного забезпечення очікуванням користувачів.

На подальших рисунках буде зображено тестування вебзастосунку.

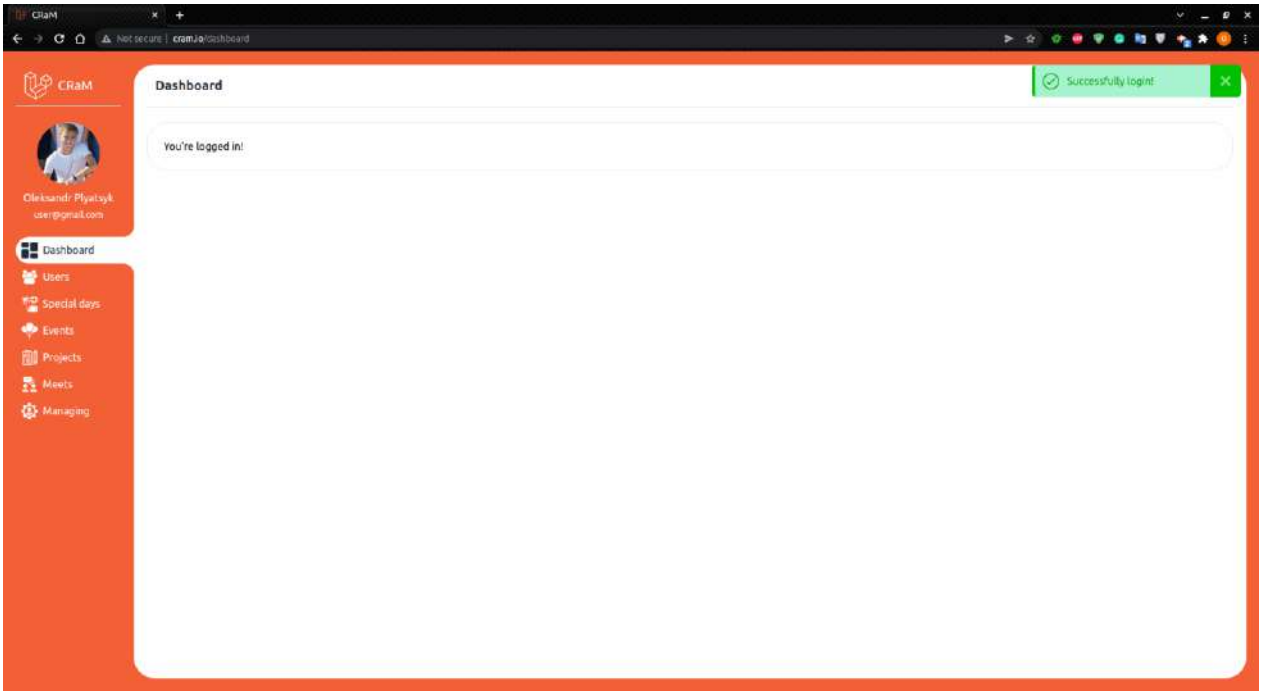


Рисунок 3.13 – Початкова сторінка з повідомленням про успішну авторизацію

Робота з сторінкою Special days:

1. Створення стуності «Особливі дні» (рисунок 3.14):

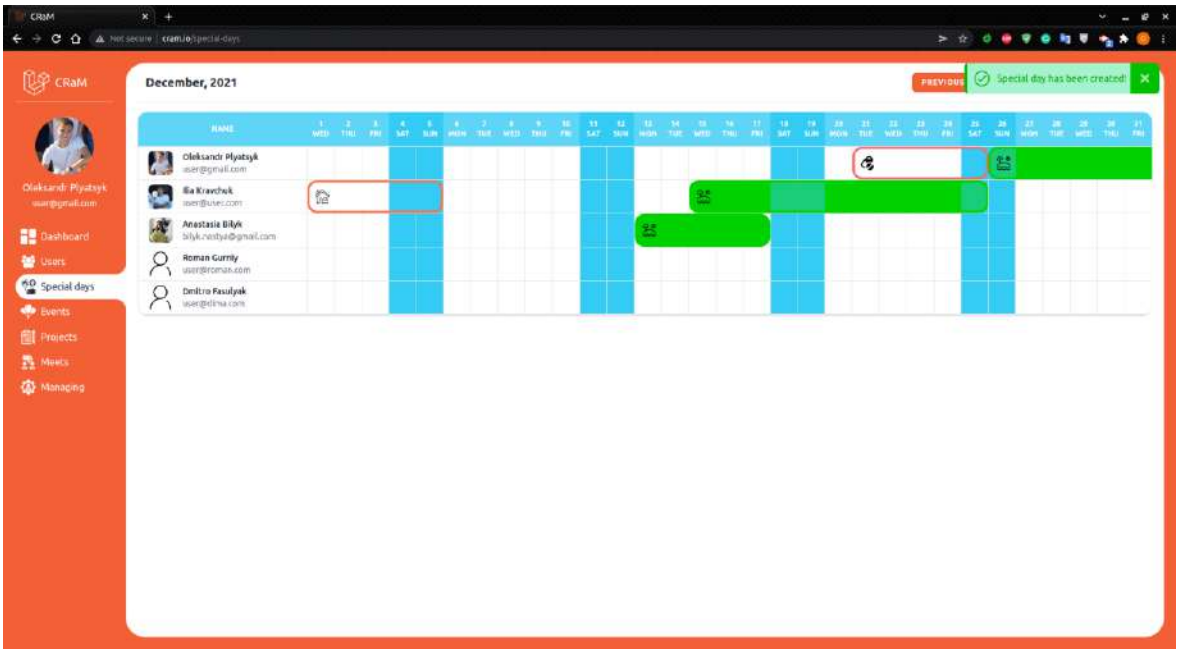


Рисунок 3.14 – Сторінка особливих днів з повідомленням про успішне створення відрізку

2. Попередження про неможливість створення стуності «Особливі дні» (рисунок 3.15):

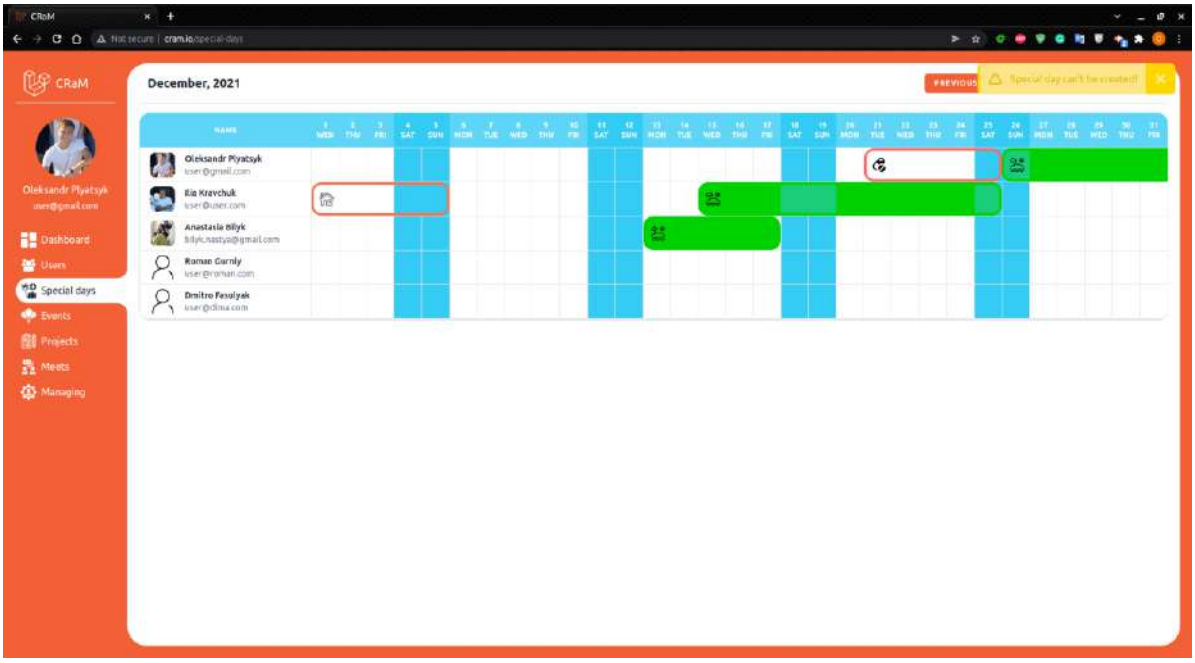


Рисунок 3.15 – Сторінка особливих днів з повідомленням про неможливість створення відрізку, оскільки на ньому вже є інший відрізок.

3. Оновлення стуності «Особливі дні» (рисунок 3.16):

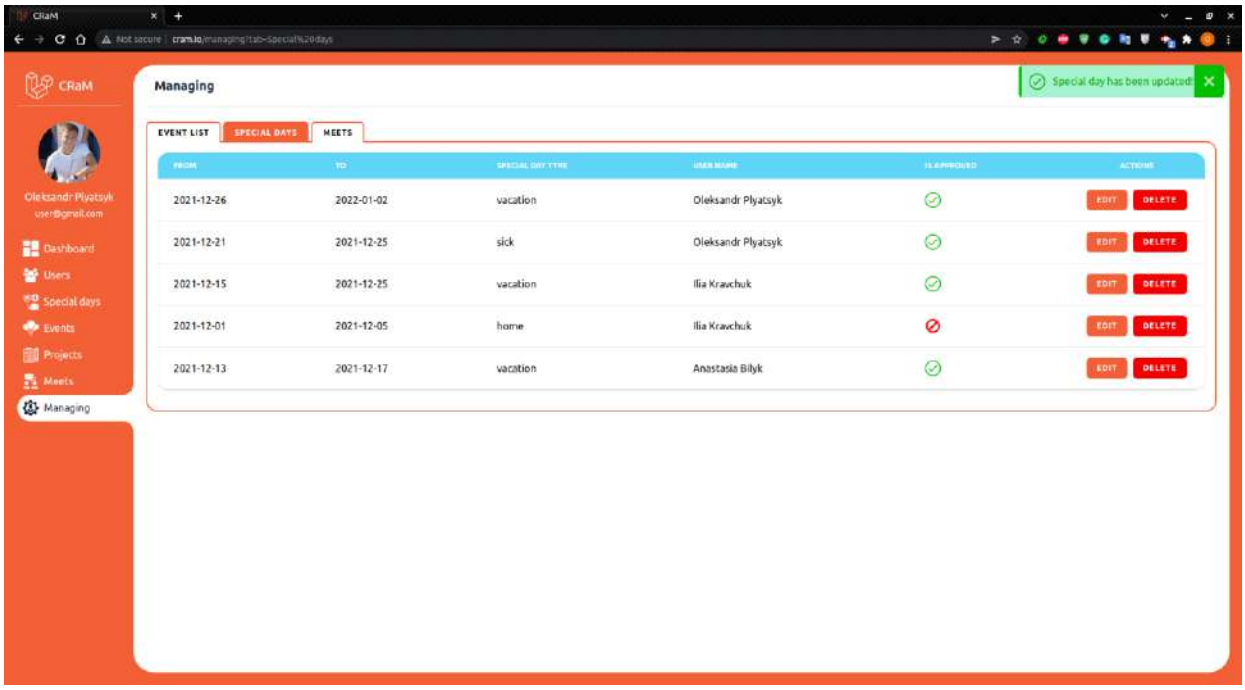


Рисунок 3.16 – Сторінка особливих днів з повідомленням про успішне оновлення відрізку

4. Видалення стуності «Особливі дні» (рисунок 3.17):

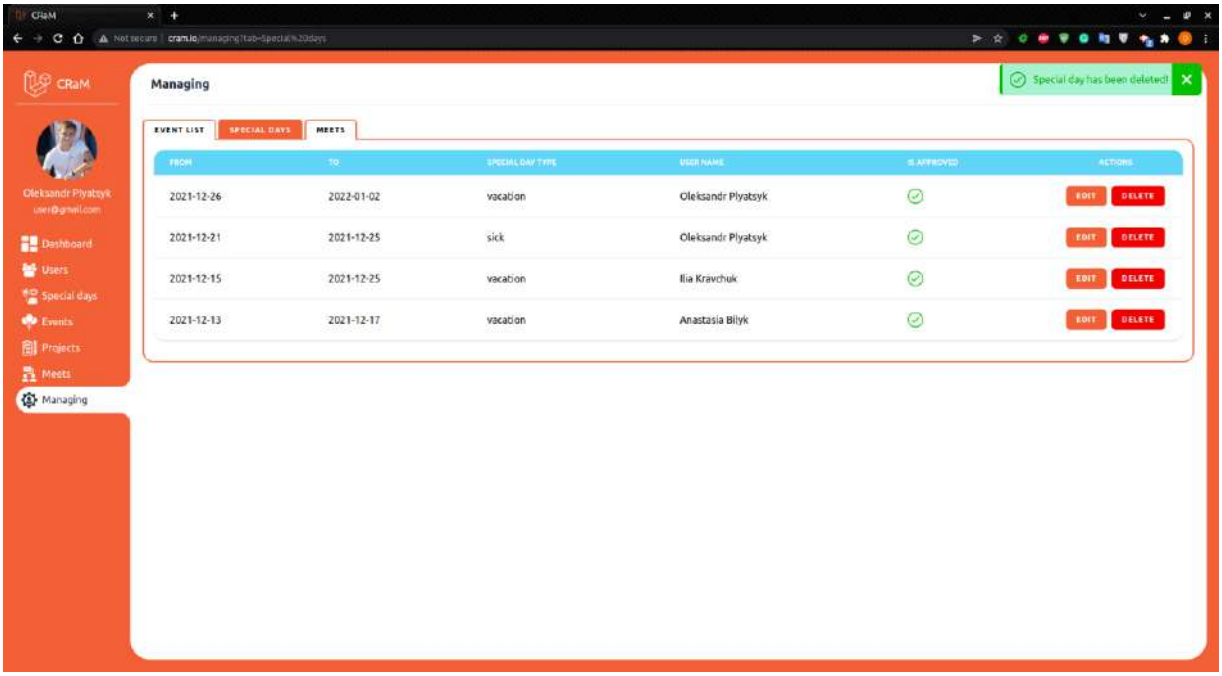


Рисунок 3.17 – Сторінка особливих днів з повідомленням про успішне видалення відрізку

Робота з сторінкою Events:

1. Створення сутності «Події» (рисунок 3.18):

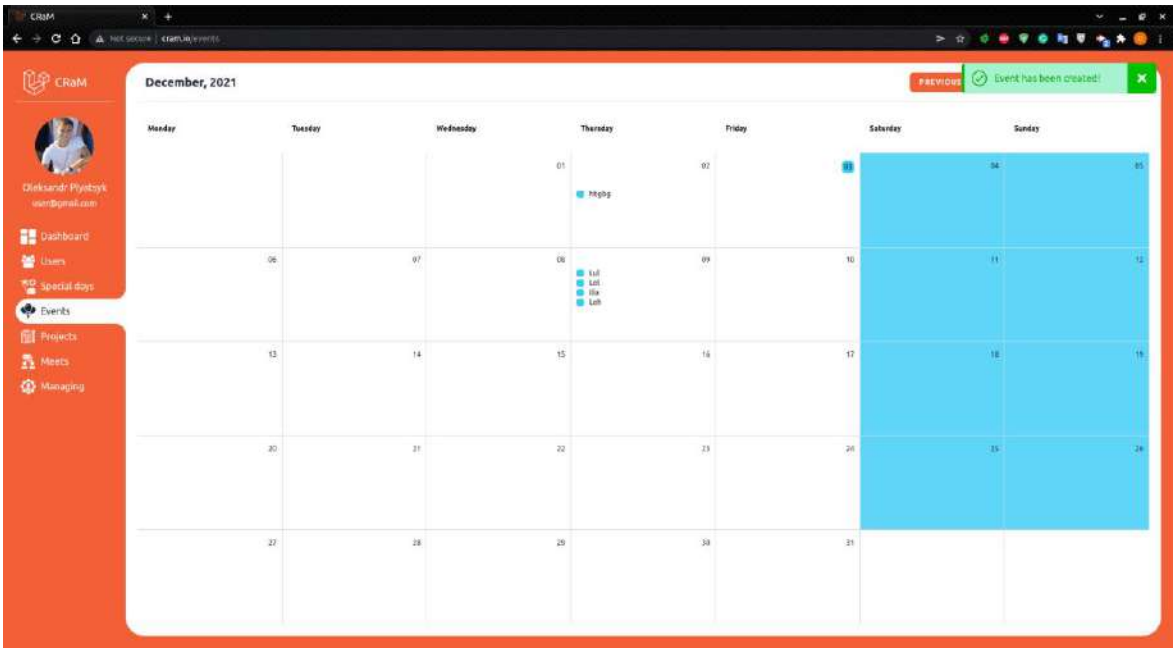


Рисунок 3.18 – Сторінка подій з повідомленням про успішне створення події на календарі

2. Оновлення сутності «Події» (рисунок 3.19):

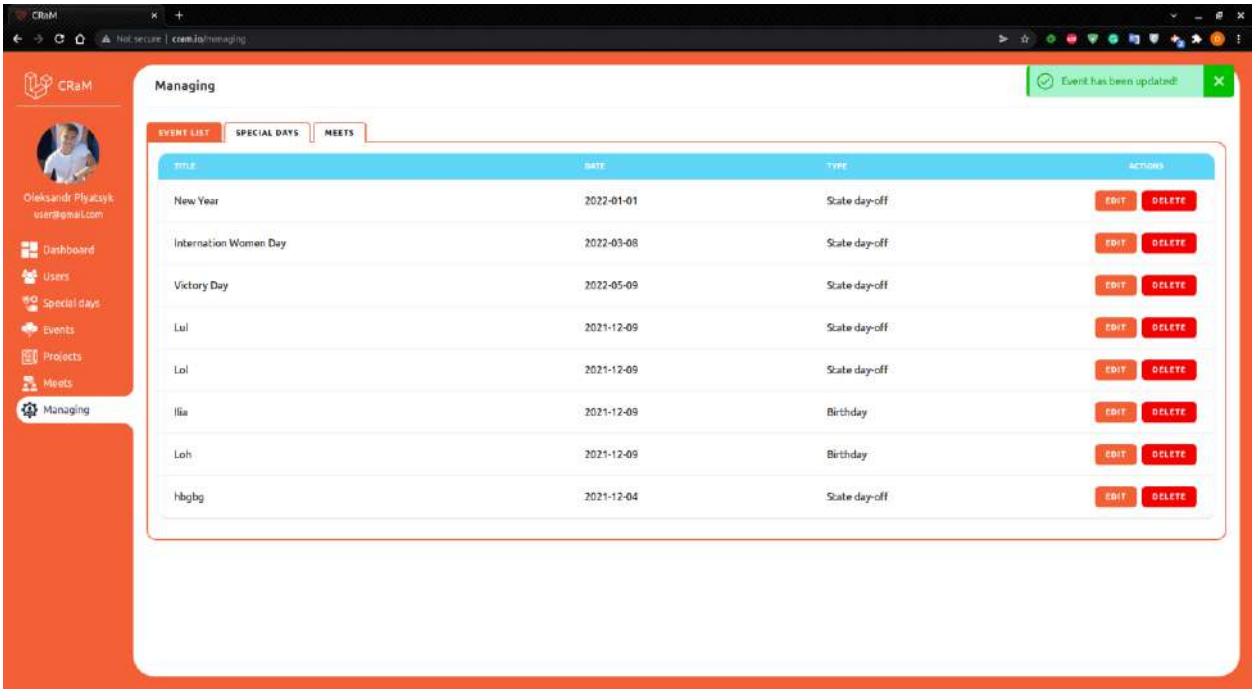


Рисунок 3.19 – Сторінка подій з повідомленням про успішне оновлення події

3. Видалення сутності «Події» (рисунок 3.20):

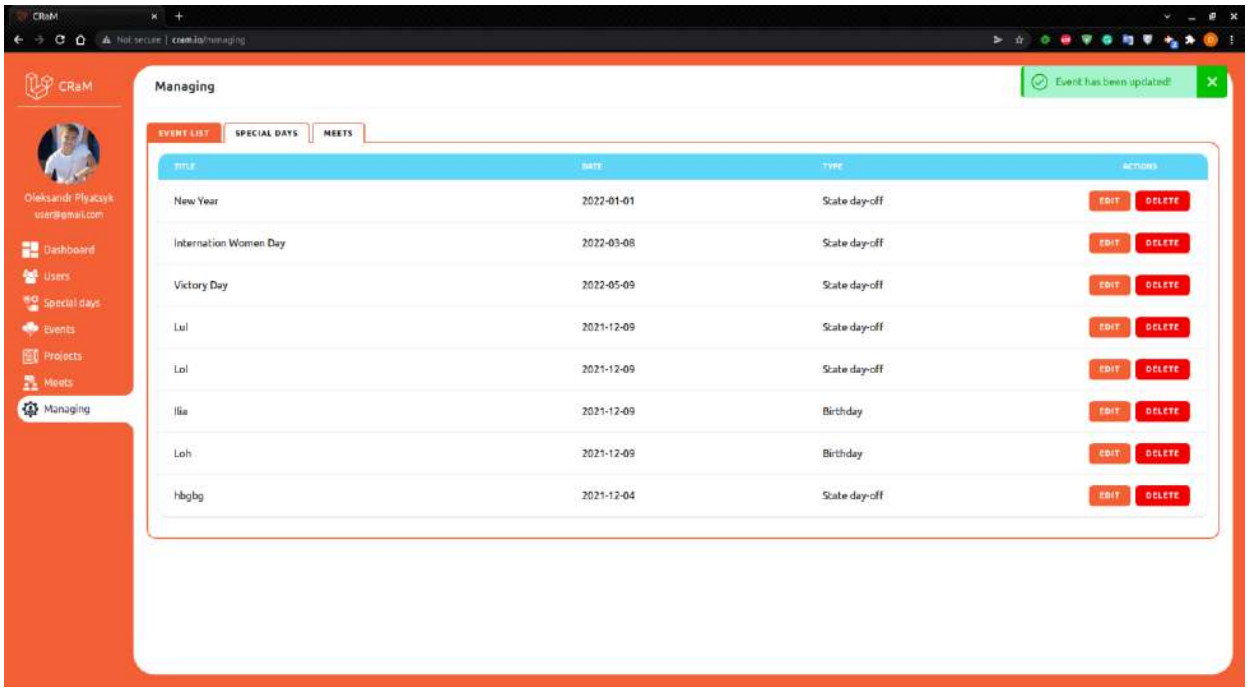


Рисунок 3.20 – Сторінка подій з повідомленням про успішне видалення події

Робота з сторінкою Projects:

1. Створення сутності «Проекти» (рисунок 3.21):

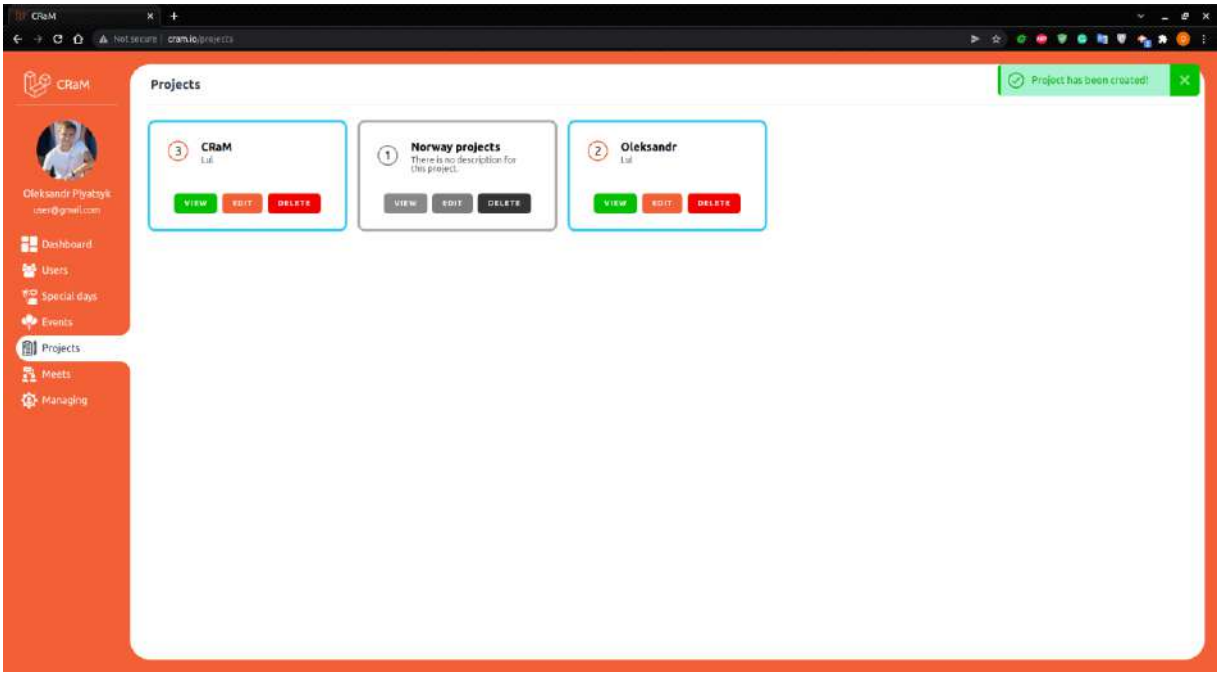


Рисунок 3.21 – Сторінка проєктів з повідомленням про успішне створення проєкту

2. Оновлення сутності «Проекти» (рисунок 3.22):

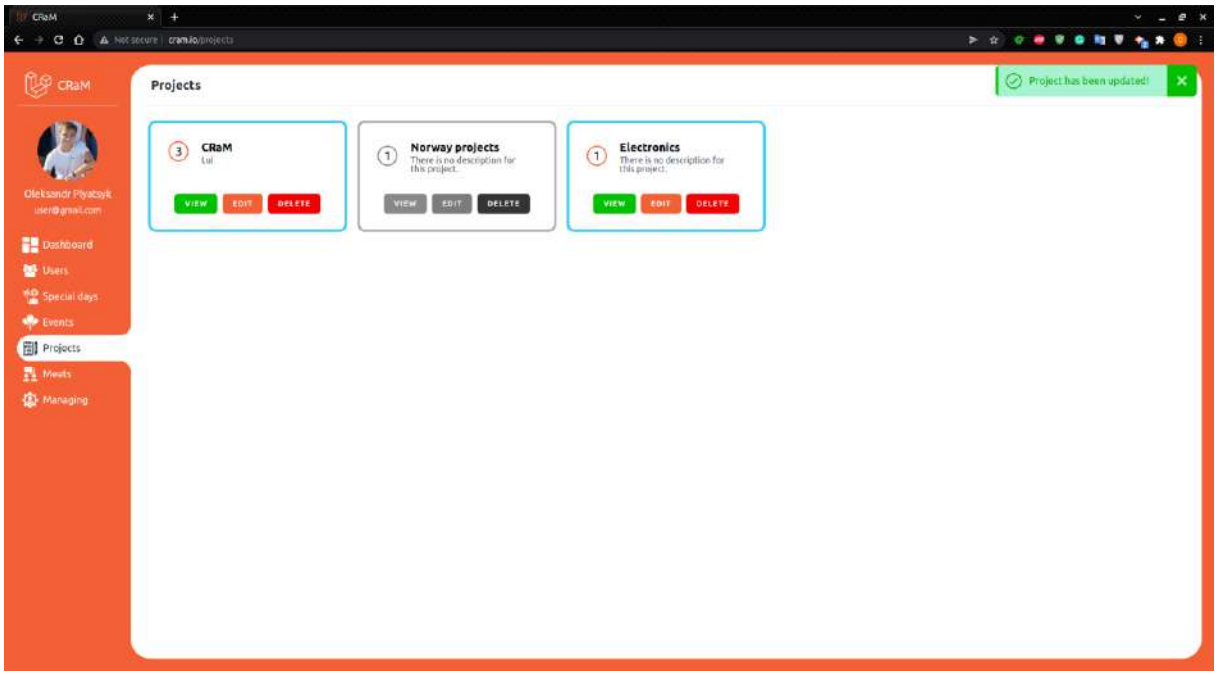


Рисунок 3.22 – Сторінка проєктів з повідомленням про успішне оновлення проєкту

3. Видалення сутності «Проекти» (рисунок 3.23):

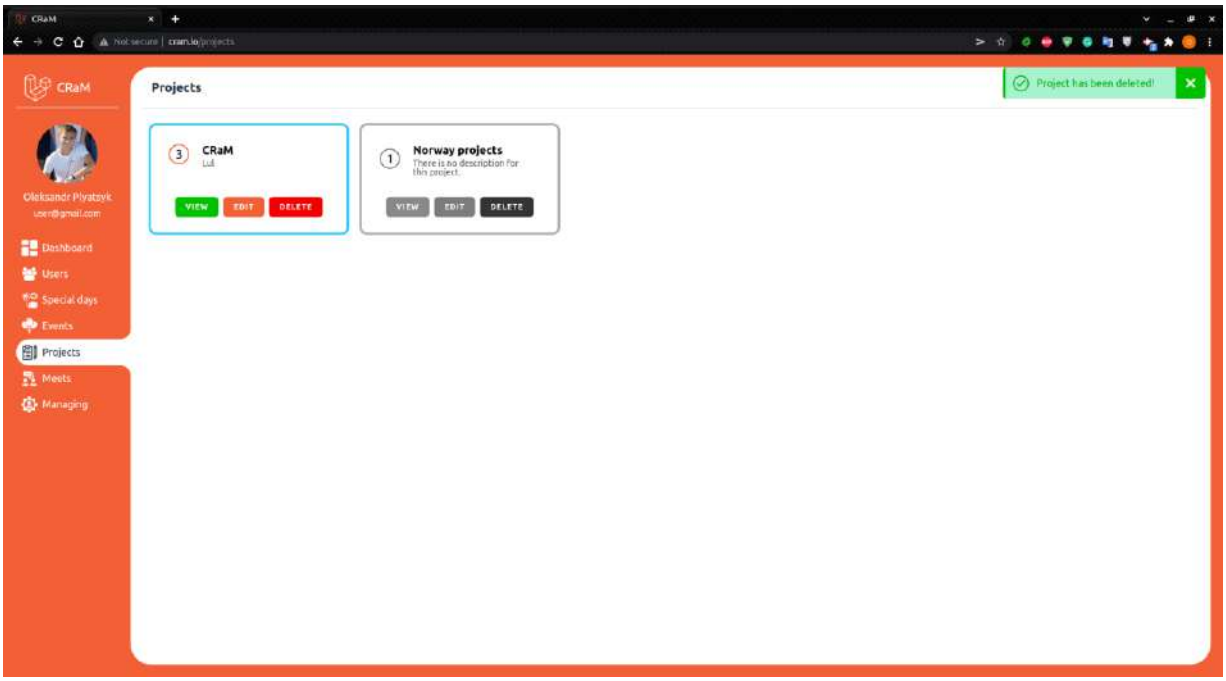


Рисунок 3.23 – Сторінка проєктів з повідомленням про успішне видалення проєкту

Робота з сторінкою Meets:

1. Створення сутності «Зустрічі» (рисунок 3.24):

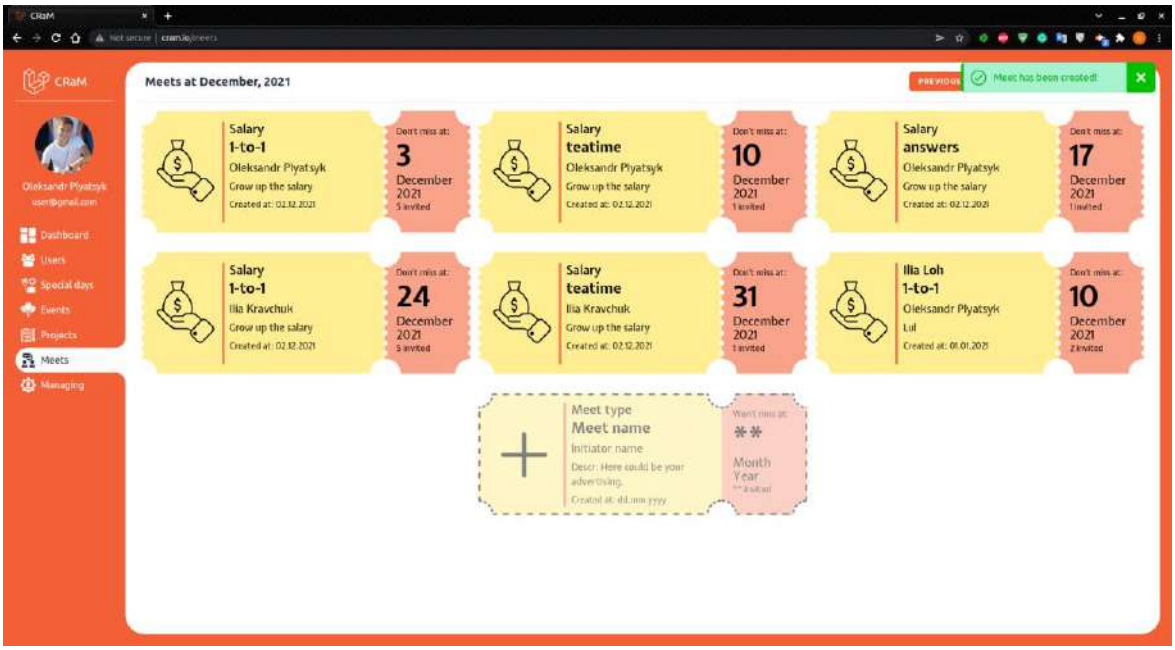


Рисунок 3.24 – Сторінка зустрічей з повідомленням про успішне створення зустрічі

2. Оновлення сутності «Зустрічі» (рисунок 3.25):

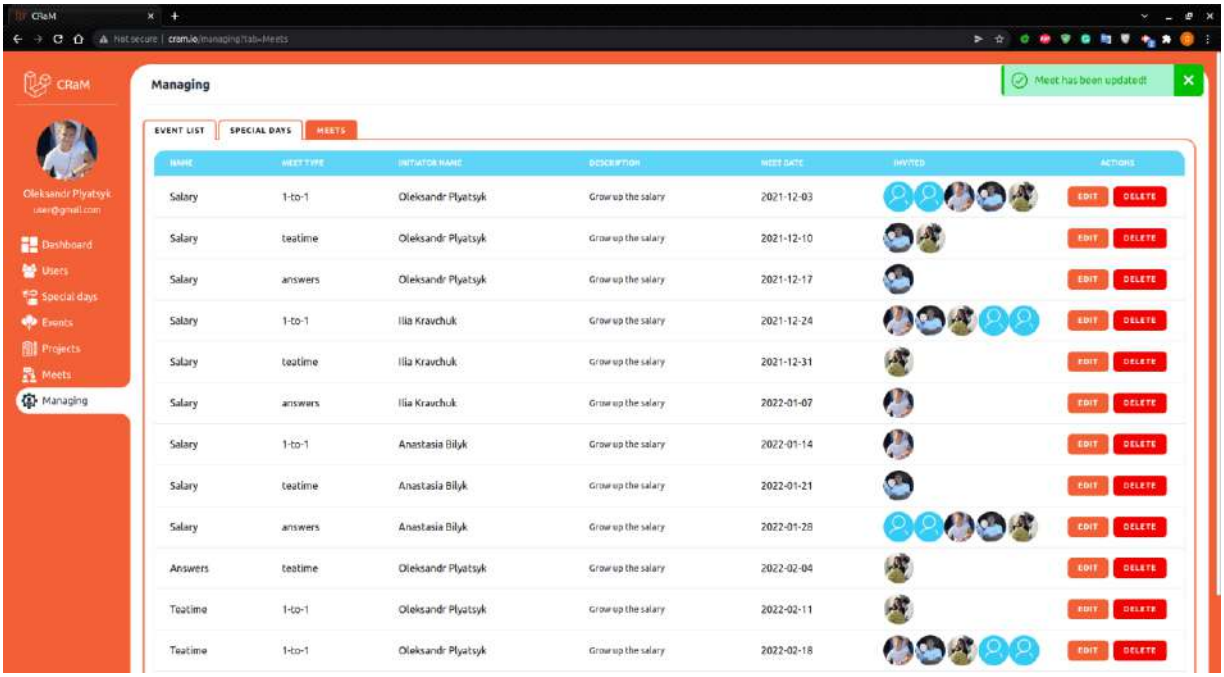


Рисунок 3.25 – Сторінка зустрічей з повідомленням про успішне оновлення зустрічі

3. Видалення сутності «Зустрічі» (рисунок 3.26):

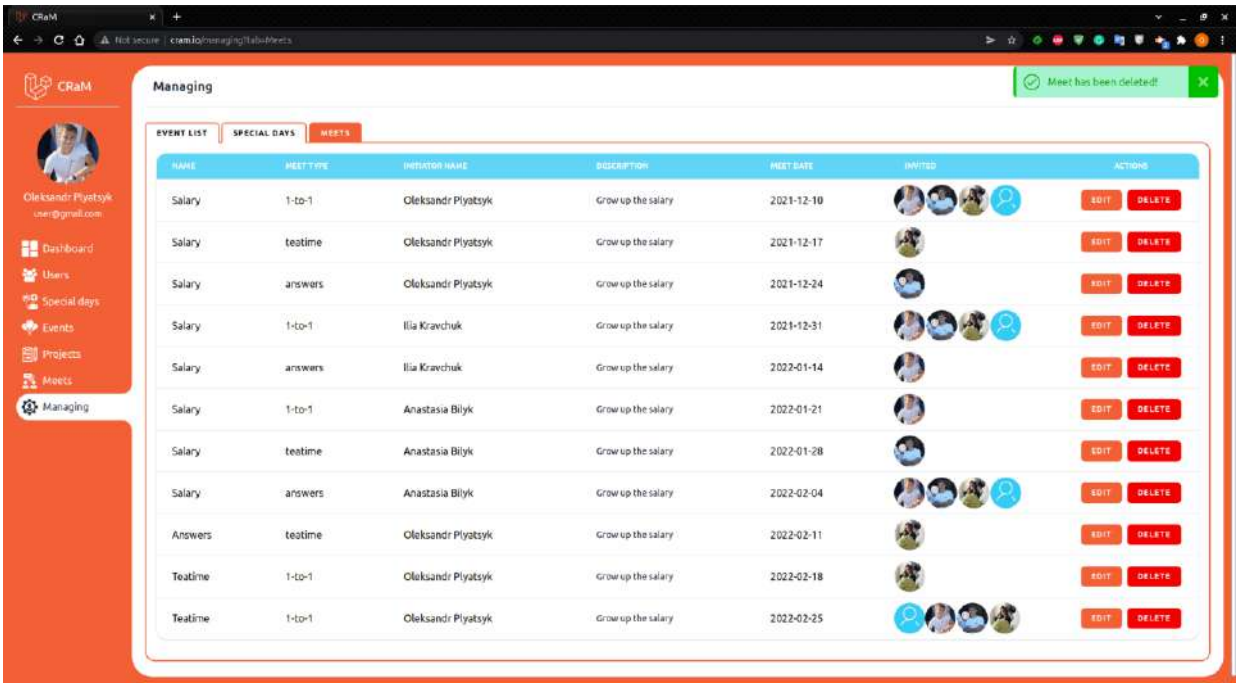


Рисунок 3.26 – Сторінка зустрічей з повідомленням про успішне видалення зустрічі

Робота з сторінкою Users:

1. Створення сутності «Користувачі» (рисунок 3.27):

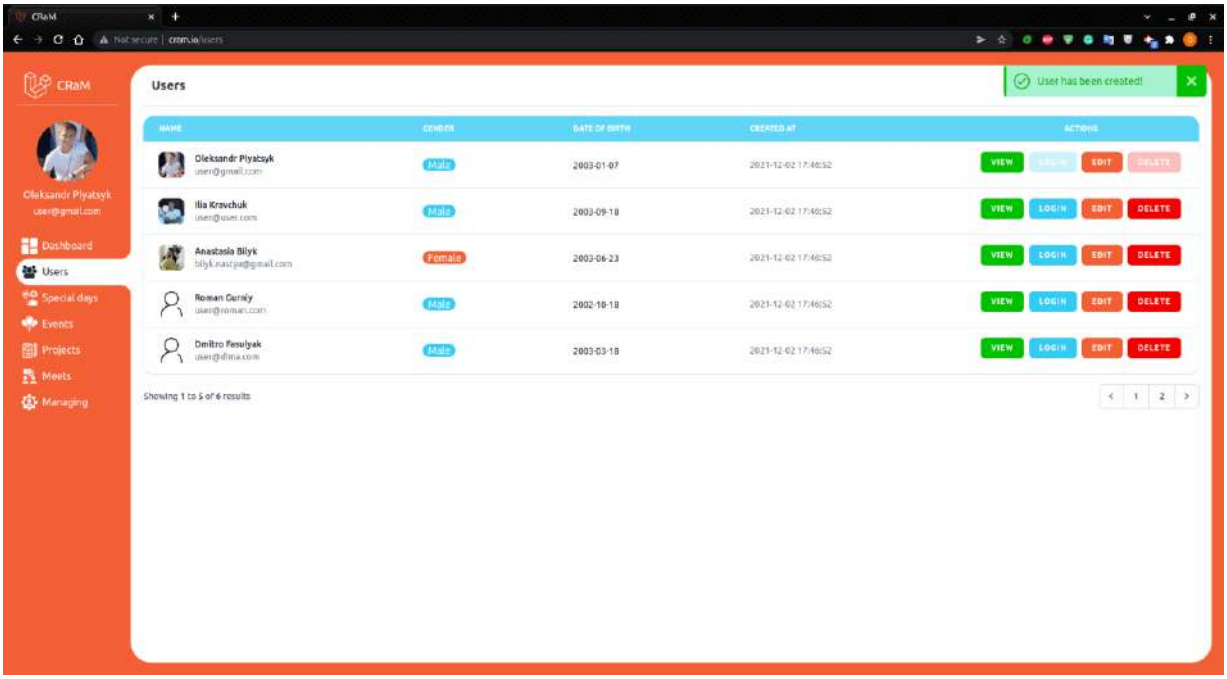


Рисунок 3.27 – Сторінка коистувачів з повідомленням про успішне створення користувача

2. Оновлення сутності «Користувачі» (рисунок 3.28):

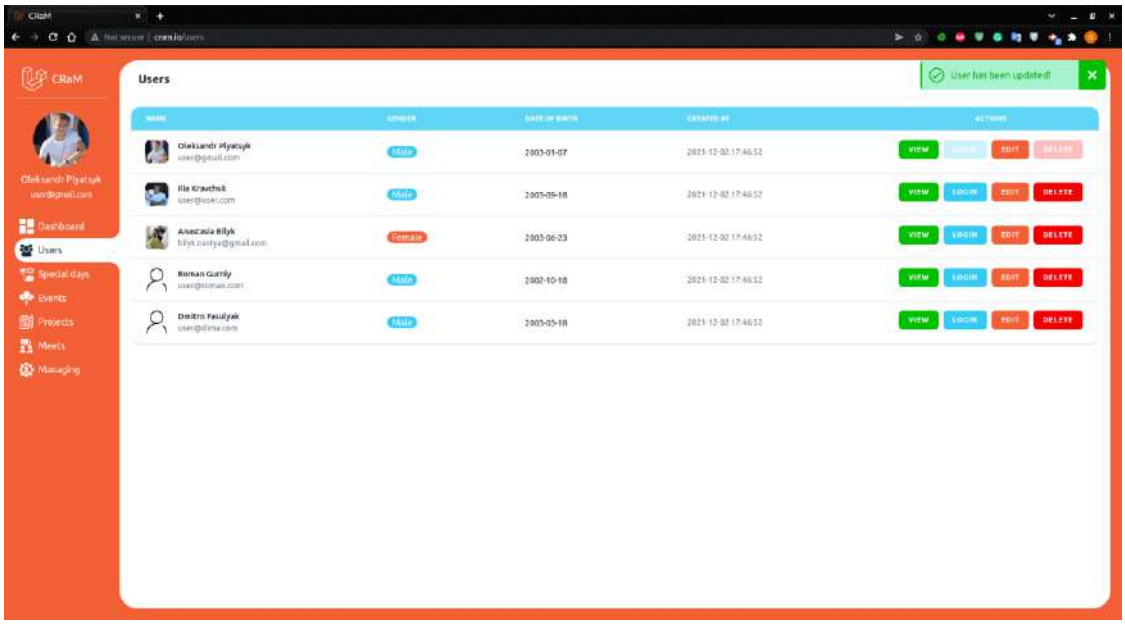


Рисунок 3.28 – Сторінка коистувачів з повідомленням про успішне оновлення користувача

3. Видалення сутності «Користувачі» (рисунок 3.29):

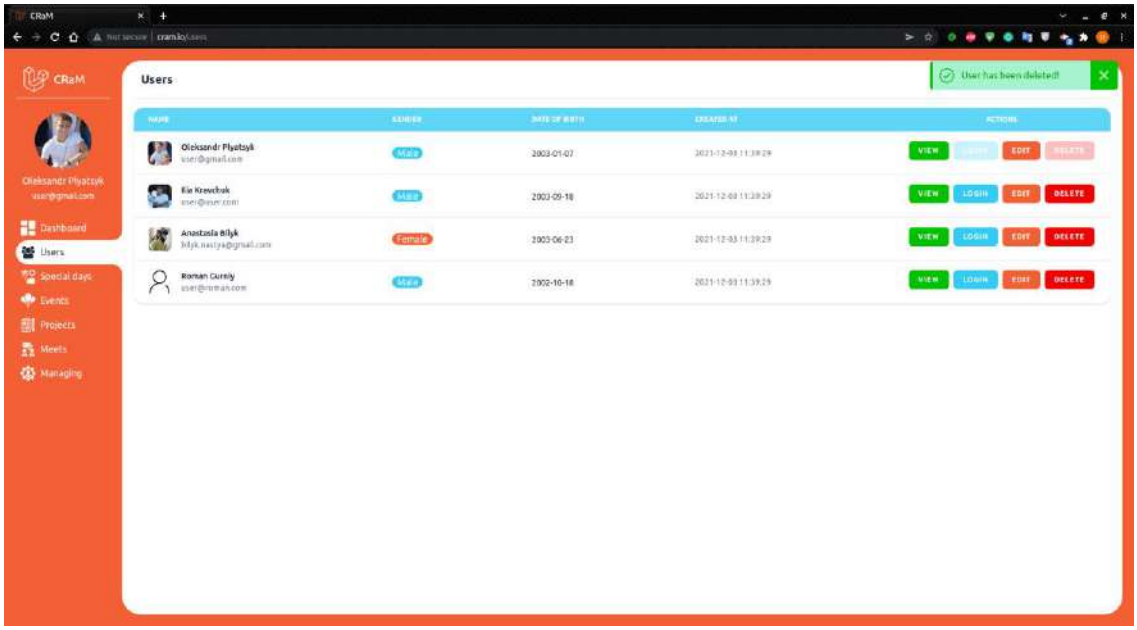


Рисунок 3.29 – Сторінка коистувачів з повідомленням про успішне видалення користувача

Як видно, усі ключові дії працюють, тому вебзастосунок готовий для використання.

3.6 Висновки до третього розділу

У процесі реалізації проєкту було обґрунтовано вибір системи керування базами даних, зокрема MySQL, яка забезпечує високу надійність, безпеку збереження даних і сумісність із PHP-фреймворком Laravel. Проведено нормалізацію таблиць відповідно до третьої нормальної форми, що дозволило уникнути дублювання інформації та підвищити ефективність роботи із даними.

Інструменти, використані в розробці, зокрема система контролю версій Git, дали змогу вести розробку поетапно з фіксацією змін і зручною організацією командної роботи. Завдяки цьому забезпечено просте відстеження історії правок, контроль за інтеграцією нового функціоналу та зменшення ризиків при внесенні змін.

Проєкт передбачає інтуїтивне використання для кінцевого користувача, однак при його підтримці та розширенні важливо розуміти базові принципи архітектури системи. Для конфігурації додатку створено окремий лаунчер, що полегшує взаємодію з інтерфейсом.

На етапі реалізації особливу увагу приділено тестуванню. Проведено як функціональні, так і нефункціональні перевірки, що дозволило оцінити роботу системи з точки зору коректності виконання бізнес-вимог, зручності, продуктивності та надійності. Аналіз вимог здійснювався шляхом формалізації сценаріїв взаємодії («вхід – результат», «подія – реакція»), що дозволило виявити потенційні проблеми ще до їх появи у фінальній версії продукту.

У результаті отримано стабільну, логічно структуровану систему, готову до подальшого впровадження, підтримки та масштабування, яка відповідає технічному завданню та сучасним стандартам розробки програмного забезпечення.

ВИСНОВКИ

Отже, у процесі розробки кваліфікаційної роботи було створено вебзастосунок під назвою «Customer Relationships and Management» (CRaM) та налаштовано спеціальний сервіс для демонстрації його функціональності. Розробка здійснювалася з використанням принципів об’єктно-орієнтованого програмування, що забезпечило модульність, масштабованість і гнучкість програмного продукту. Перед початком безпосередньої розробки було проведено глибокий аналіз предметної області, що дозволило чітко визначити вимоги до системи та основні функціональні можливості.

У якості середовища розробки було обрано IDE PhpStorm, яка забезпечує ефективну роботу з PHP-кодом, а для бекенд-частини застосовано популярний і потужний PHP-фреймворк Laravel. Збереження та обробка даних реалізовані на основі реляційної бази даних MySQL, що гарантує надійність, швидкість і масштабованість системи. У межах розробленого продукту реалізовано основний функціонал для роботи з користувацькими даними та суміжними сутностями, включаючи управління персоналом, проєктами, подіями та особливими днями.

Перший розділ роботи присвячено аналізу предметної області, де обґрунтовано актуальність створення CRM-системи, проведено огляд існуючих рішень на ринку, визначено їхні сильні та слабкі сторони. Це дозволило не лише з’ясувати поточні тренди у сфері CRM, а й врахувати їх при формуванні технічного завдання на розробку.

Другий розділ зосереджено на теоретичних засадах, пов’язаних з архітектурою програмних систем. Проведено огляд основних типів архітектур, вибрано оптимальний клієнт-серверний підхід, що забезпечує високу продуктивність і масштабованість системи. Також описано процес проєктування структури даних і визначено логіку взаємодії між різними компонентами системи, що є ключовим для підтримки цілісності і узгодженості інформації.

У третьому розділі детально описано процес розробки вебзастосунку: від вибору технологій та написання коду до особливостей інсталяції та експлуатації системи. Описаний реалізований функціонал, включаючи ключові модулі, механізми обробки даних, систему ролей і прав доступу, що забезпечують безпеку та контроль над інформацією. Було проведене тестуваня розробленої системи. Здійснено перевірку відповідності функціоналу вимогам технічного завдання, проведено функціональне і нефункціональне тестування, за результатами якого виявлено і усунуто помилки. Також було підготовлено детальну супровідну документацію, яка допомагає користувачам швидко освоїти систему та забезпечує підтримку у подальшій експлуатації.

У результаті виконаної роботи створено стабільний, зручний у використанні програмний продукт, який відповідає поставленим завданням і може бути застосований для автоматизації процесів управління персоналом у середніх і великих IT-компаніях. Система дозволяє централізовано управляти даними, планувати події, контролювати робочий час і вести комунікацію між співробітниками, що значно підвищує ефективність бізнес-процесів.

Серед недоліків варто відзначити, що на поточному етапі система працює лише в локальному середовищі, що обмежує її масштабування та доступність у розподілених мережах. Крім того, частина додаткового функціоналу, зокрема чат для внутрішнього спілкування, наразі не реалізована і потребує подальшої розробки. Ці аспекти можуть стати пріоритетними напрямами для розвитку та вдосконалення системи в майбутньому.

Загалом, розроблений вебзастосунок є міцною основою для подальшої автоматизації управління взаємовідносинами з клієнтами та персоналом. Використані технології і архітектурні рішення забезпечують достатню гнучкість, безпеку і масштабованість, що відповідає сучасним вимогам ринку. Подальші удосконалення системи можуть включати інтеграцію з мобільними додатками, впровадження механізмів машинного навчання для аналізу даних і автоматизацію бізнес-процесів.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Онлайн-сервіс обліку часу Everhour. URL: https://www.everhour.com/ (дата звернення: 10.02.2025).
2. Платформа для керування проєктами ActiveCollab. URL: https://www.activecollab.com/ (дата звернення: 08.02.2025).
3. Сервіс обліку робочого часу Clockify. URL: https://www.clockify.me/ (дата звернення: 12.02.2025).
4. Система управління базами даних MySQL. URL: https://www.mysql.com/ (дата звернення: 18.02.2025).
5. Офіційна документація мови програмування PHP. URL: https://www.php.net/ (дата звернення: 21.02.2025).
6. Фреймворк для веброзробки Laravel.. URL: https://www.laravel.com/ (дата звернення: 25.02.2025).
7. Інструмент для локального серверного середовища XAMPP. URL: https://www.apachefriends.org/index.html (дата звернення: 01.03.2025).
8. Інтегроване середовище розробки PhpStorm. URL: https://www.jetbrains.com/phpstorm/ (дата звернення: 06.03.2025).
9. Система контролю версій Git. URL: https://git-scm.com/ (дата звернення: 10.03.2025).
10. Основи патерну проектування MVC. URL: https://www.guru99.com/mvc-tutorial.html (дата звернення: 15.03.2025).
11. Керівництво з JavaScript для початківців. URL: https://uk.javascript.info/ (дата звернення: 19.03.2025).
12. Основи HTML-розмітки. URL: https://www.css.in.ua/ (дата звернення: 22.03.2025).
13. Основи CSS-стилів. URL: https://www.css.in.ua/ (дата звернення: 22.03.2025).
14. Офіційна сторінка мови програмування Perl. URL: https://www.perl.org/ (дата звернення: 28.03.2025).

15. Платформа контейнеризації Docker. URL: https://www.docker.com/ (дата звернення: 01.04.2025).

16. Операційна система Windows від Microsoft. URL: https://www.microsoft.com/uk-ua/ (дата звернення: 05.04.2025).

17. Операційна система з відкритим кодом Linux. URL: https://www.linux.org/ (дата звернення: 10.04.2025).

18. Операційна система Mac OS. URL: https://support.apple.com/macos/ (дата звернення: 15.04.2025).

19. JavaScript-бібліотека ReactJS. URL: https://reactjs.org/ (дата звернення: 22.04.2025).

20. Мобільна операційна система Android. URL: https://www.android.com/ (дата звернення: 01.05.2025).

21. Документація Laravel Sanctum. URL: https://laravel.com/docs/10.x/sanctum/ (дата звернення: 03.05.2024).

22. Офіційна документація Composer. URL: https://getcomposer.org/doc/ (дата звернення: 05.05.2025).

23. API-розробка з Postman. URL: https://learning.postman.com/ (дата звернення: 07.05.2025).

24. Безпечний код OWASP. URL: https://owasp.org/www-project-top-ten/ (дата звернення: 08.05.2025).

25. Документація REST API. URL: https://restfulapi.net/ (дата звернення: 09.05.2025).

26. Swagger (OpenAPI) Docs. URL: https://swagger.io/specification/ (дата звернення: 10.05.2025).

27. Тестування з PHPUnit. URL: https://phpunit.de/ (дата звернення: 11.05.2025).

28. Документація Laravel Testing. URL: https://laravel.com/docs/10.x/testing/ (дата звернення: 12.05.2025).

29. Документація GitHub Actions. URL: https://docs.github.com/en/actions/ (дата звернення: 13.05.2025).

30. Docker Compose Docs. URL: https://docs.docker.com/compose/ (дата звернення: 14.05.2025).

31. Документація про JWT (JSON Web Token). URL: https://jwt.io/introduction/ (дата звернення: 15.05.2025).

32. React Router Docs. URL: https://reactrouter.com/en/main/ (дата звернення: 16.05.2025).

33. Laravel Eloquent ORM. URL: https://laravel.com/docs/10.x/eloquent (дата звернення: 17.05.2025).

34. Підхід MVC на MDN. URL: https://developer.mozilla.org/en-US/docs/Glossary/MVC/ (дата звернення: 18.05.2025).

35. Архітектура мікросервісів. URL: https://microservices.io/ (дата звернення: 19.05.2025).

36. UX/UI Принципи. URL: https://www.nngroup.com/articles/definition-user-experience/ (дата звернення: 20.05.2025).

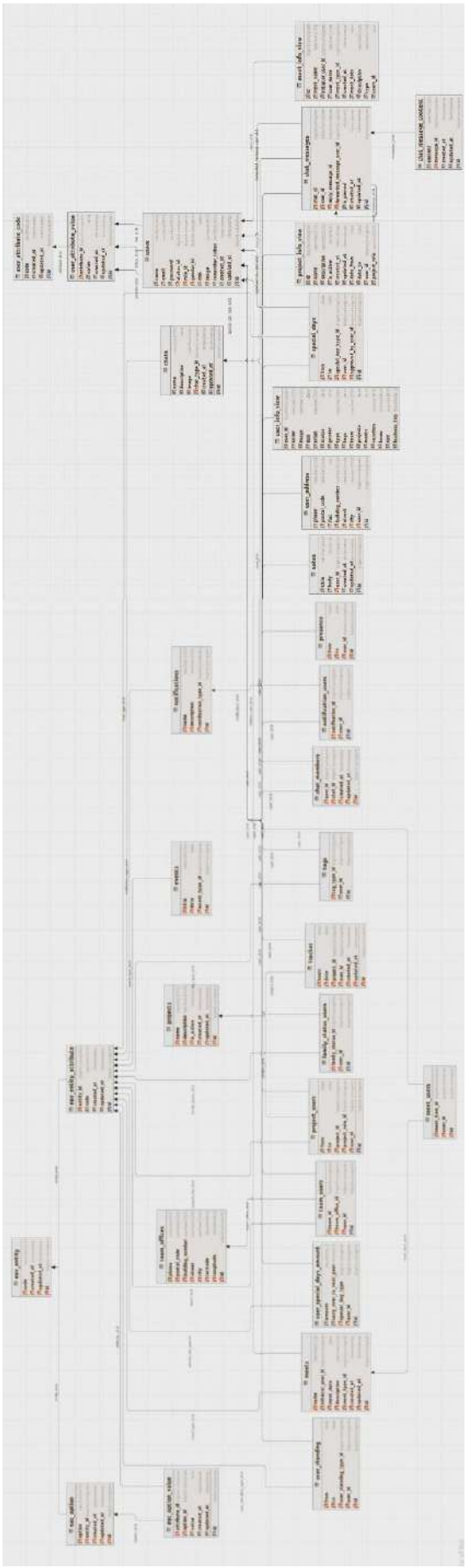
37. Agile Manifesto. URL: https://agilemanifesto.org/ (дата звернення: 21.05.2025).

38. Scrum Guide 2020. URL: https://scrumguides.org/ (дата звернення: 22.05.2025).

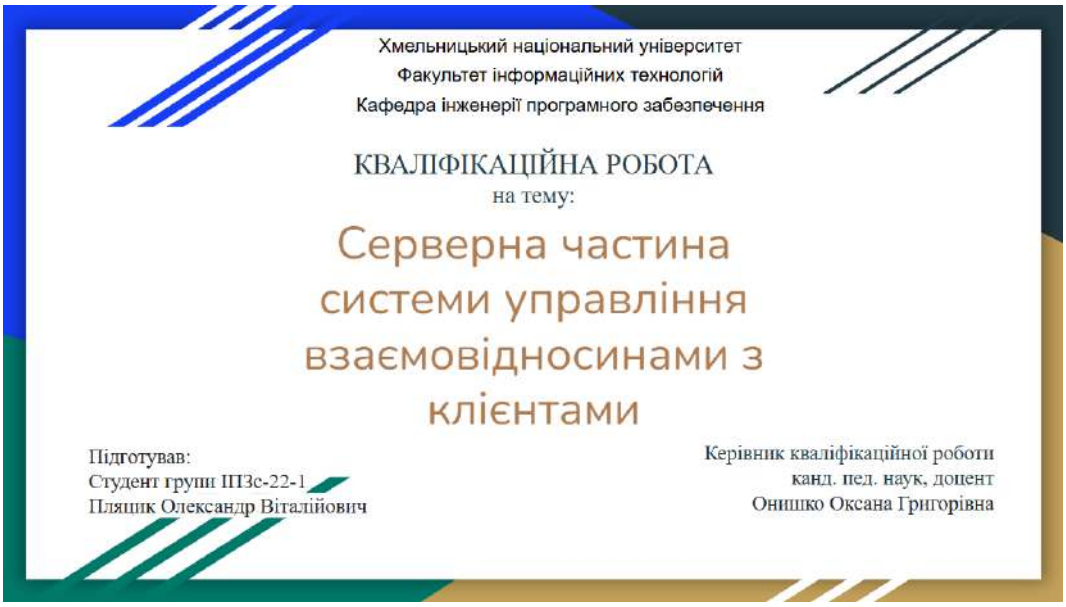
39. Візуалізація даних з Chart.js. URL: https://www.chartjs.org/docs/latest/ (дата звернення: 23.05.2025).

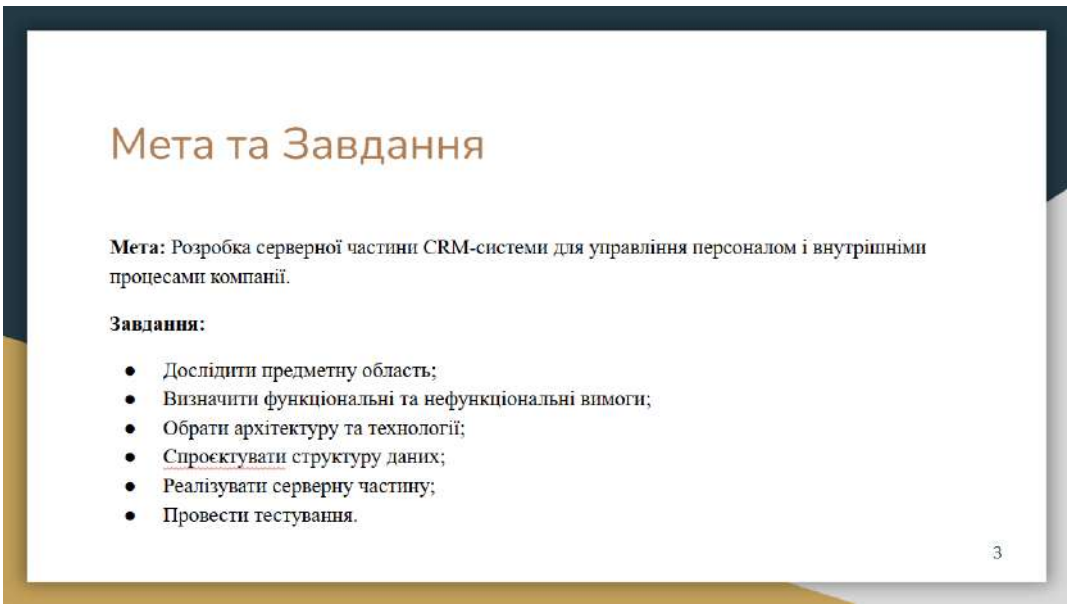
40. Стандарти API Microsoft. URL: https://learn.microsoft.com/en-us/azure/architecture/best-practices/api-design/ (дата звернення: 24.05.2025).

ДОДАТОК А – Інфологічна схема



ДОДАТОК Б – Презентаційні матеріали





Змістовий аналіз предметної області, її структурних та функціональних особливостей

CRM (Customer Relationship Management) — це інструмент для автоматизації процесів комунікації між компанією та клієнтами. Такі системи дають змогу вести базу контактів, формувати воронки продажів, генерувати звіти, автоматизувати роботу відділу продажу, маркетингу та підтримки.

Особливість предметної області: CRM повинна мати чітку структуру даних, підтримувати зв'язки між сутностями, забезпечувати зручну навігацію та гнучке розширення функціоналу.

4

Аналіз наявного програмно-технічного забезпечення

Назва	Основні функції	Переваги	Недоліки
Everhour	Відстеження часу, бюджети, завдання, інтеграції з хмарними сервісами	Хмарне рішення, простота у використанні, зручний облік часу	Орієнтованість на задачі, а не на клієнтів
ActiveCollab	Керування проєктами, планування, взаємодія в команді	Централізована робота над проєктами, календар, аналітика	Висока вартість, складність для малих команд
Clockify	Облік робочого часу, створення завдань, таймер, звіти	Безкоштовний тариф, багатофункціональний інтерфейс	Вузька спрямованість лише на трекінг часу

5

Визначення функціональних та нефункціональних вимог до ПЗ

Функціональні вимоги:

- Управління користувачами
- Додавання подій і зустрічей
- Керування проєктами
- Система особливих днів

Нефункціональні вимоги:

- Безпека
- Масштабованість
- Продуктивність
- Дружній інтерфейс
- Розширюваність

6

Вибір типу архітектури та шаблонів проєктування

Було обрано **клієнт-серверну архітектуру**.

Переваги:

- Централізоване управління даними
- Покращена безпека
- Простота масштабування
- Зменшення навантаження на клієнтську частину

Клієнт надсилає запит → сервер обробляє запит → повертає лише результат.

7

Опис декомпозиції, залежностей, інтерфейсів

Проєкт реалізовано за шаблоном MVC:

- **Model** – робота з базою даних
- **View** – представлення (фронтенд)
- **Controller** – логіка обробки запитів

Сутності:

- Користувачі
- Події
- Проєкти
- Особливі дні
- Зустрічі

8

Проєктування модулів і даних

База даних: MySQL

Моделі: users, projects, events, special_days, meets

Нормалізація: до 3-ї нормальної форми. Виділення окремих таблиць для типів, ролей, зв'язків між сутностями.

Інструменти: PhpStorm, міграції Laravel, графічне моделювання

9

Аналіз та вибір технологій

Серверна частина: Laravel (PHP)

База даних: MySQL

Інші інструменти:

- PhpStorm (IDE)
- Docker (контейнеризація)
- Git (контроль версій)
- MVC (архітектурний шаблон)

10

Реалізація модулів і база даних

Реалізовані модулі:

- Аутентифікація та авторизація
- Система ролей
- Робота з подіями, проєктами, зустрічами
- Створення та погодження особливих днів

База даних:

- Реалізована структура бази даних
- Реалізовані базові запити

11

Вимоги до технічного та програмного забезпечення

Сервер:

- Apache/Nginx
- PHP 8+
- MySQL 8+

Інструменти:

- XAMPP (локальна розробка)
- PhpStorm
- Docker (для деплоїменту)

ОС: Windows / Linux / macOS

12

Тестування ПЗ

- Функціональне та нефункціональне тестування
- Перевірка ролей, відповідності вимогам
- Мануальне тестування

Результати: система стабільно працює та відповідає заявленим вимогам.

13

Висновки

Завдання	Виконання
Аналіз предметної області	✅ виконано
Вибір архітектури	✅ клієнт-серверна, обґрунтована
Вибір технологій	✅ Laravel + MySQL
Реалізація	✅ виконано, додано базову функціональність
Тестування	✅ проведено, система працює стабільно

Коментар: Всі поставлені завдання реалізовано повністю. Система готова до подальшого впровадження та масштабування.

14

Дякую за увагу!

ДОДАТОК В – Код проєкту

```
<?php
namespace App\Models;

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

/**
 * @property integer $eventId
 * @property string $eventName
 * @property $eventDate
 * @property integer $eventCategoryId
 */
class CalendarEntry extends Model
{
    use HasFactory;

    const EVENT_ID = 'id';
    const EVENT_NAME = 'title';
    const EVENT_DATE = 'date';
    const EVENT_CATEGORY_ID = 'event_type_id';

    public $timestamps = false;

    protected $fillable = [
        self::EVENT_NAME,
        self::EVENT_DATE,
        self::EVENT_CATEGORY_ID
    ];

    protected $table = 'events';

    public function __get($attribute)
    {
        return $this->getAttribute($attribute);
    }

    public function __set($attribute, $value)
    {
        $this->setAttribute($attribute, $value);
    }
}
```

```
<?php

namespace App\Http\Controllers\Auth;

use App\Http\Controllers\Controller;
use Exception;
use Illuminate\Http\RedirectResponse;
use Illuminate\Http\Request;
use Illuminate\Http\Response;
use Illuminate\Support\Facades\Password;
use Illuminate\View\View;

class ResetPasswordLinkController extends Controller
{
    public function showForm(): View
    {
        return view('auth.forgot-password');
    }

    public function sendLink(Request $request)
    {
        try {
            $request->validate(['email' => ['required', 'email']]);

            $result = Password::sendResetLink($request->only('email'));

            return $result == Password::RESET_LINK_SENT
                ? back()->with('status', __($result))
                : back()->withErrors(['email' => __($result)]);
        } catch (Exception $e) {
            return back()->withErrors(['email' => __('Unable to send reset link. Please try later.')]);
        }
    }
}
```

```
<?php declare(strict_types=1);

namespace Modules\Auth\Http\Controllers;

use App\Http\Controllers\Controller;
use Illuminate\Http\JsonResponse;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\Validator;
use Illuminate\Validation\ValidationException;

class UserSessionController extends Controller
{
    public function __construct()
    {
        $this->middleware('auth:api', ['except' => ['login']]);
    }

    public function login(Request $request): JsonResponse
    {
        $credentials = $request->only(['email', 'password']);

        $validator = Validator::make($credentials, [
            'email' => ['required', 'email'],
            'password' => ['required', 'string', 'min:8'],
        ]);

        if ($validator->fails()) {
            return response()->json($validator->errors(), 422);
        }

        if (!$token = Auth::attempt($validator->validated())) {
            return response()->json([
                'message' => [
                    'status' => 'error',
                    'text' => 'Invalid credentials.'
                ]
            ], 401);
        }

        return response()->json([
            'auth' => [
                'token' => $token,
                'type' => 'Bearer',
                'ttl' => Auth::factory()->getTTL(),
            ],
            'user' => Auth::user(),
            'message' => [
                'status' => 'success',
                'text' => 'Authenticated successfully.'
            ],
        ], 202);
    }
```

```
    public function logout(): JsonResponse
    {
        Auth::logout();

        return response()->json([
            'message' => [
                'status' => 'success',
                'text' => 'Session ended.'
            ]
        ]);
    }

    public function profile(): JsonResponse
    {
        return response()->json(['user' => Auth::user()]);
    }

    public function refreshToken(): JsonResponse
    {
        return response()->json([
            'auth' => [
                'token' => Auth::refresh(),
                'type' => 'Bearer',
                'ttl' => Auth::factory()->getTTL(),
            ],
            'user' => Auth::user(),
            'message' => [
                'status' => 'success',
                'text' => 'Token refreshed.'
            ],
        ]);
    }
}
```

```
<?php declare(strict_types=1);

namespace Modules\SpecialDay\Http\Controllers;

use Carbon\Carbon;
use Exception;
use Illuminate\Http\JsonResponse;
use Illuminate\Http\Request;
use Illuminate\Routing\Controller;
use Modules\SpecialDay\Entities\SpecialDays;
use
Modules\SpecialDay\Repositories\Interfaces\SpecialDaysR
epositoryInterface;
use Modules\User\Entities\Users;
use
Modules\User\Repositories\Interfaces\UserSpecialDaysAm
ountRepositoryInterface;
use
Modules\User\Repositories\Interfaces\UsersRepositoryInter
face;
use Psr\Log\LoggerInterface;

class HolidayController extends Controller
{
    public function __construct(
        private Carbon $clock,
        private UsersRepositoryInterface $userRepo,
        private SpecialDaysRepositoryInterface
$holidayRepo,
        private UserSpecialDaysAmountRepositoryInterface
$holidayAmountRepo,
        private LoggerInterface $log
    ) {}

    public function index(): void
    {
            $current = $this->clock->now();
        $this->indexMonthYear($current->month, $current-
>year);
    }

    public function indexMonthYear($month, $year):
JsonResponse
    {
        $month = $this->limitValue((int)$month, 1, 12);
        $year = $this->limitValue((int)$year, 1970, 2100);
        $targetDate = $this->clock::create($year, $month);

        $userHolidays = $this->holidayRepo-
>getAllByUser($year, $month);
        $users = $this->userRepo->all();

        for ($d = 1; $d <= $targetDate->daysInMonth; $d++)
{
            $headers[] = $this->clock::parse("$year-$month-
$d");
        }

        foreach ($users as $person) {
            $holiday = $userHolidays[$person-
>__get(Users::ID)][0] ?? null;

            for ($d = 1; $d <= $targetDate->daysInMonth;
$d++) {
                $dateCheck = $this->clock::parse("$year-
$month-$d");

                if ($holiday && $dateCheck-
>isBetween($holiday->__get(SpecialDays::FROM),
$holiday->__get(SpecialDays::TO))) {
                    $type = $holiday-
>__get(SpecialDays::SPECIAL_DAY_TYPE_ID);
                    $approved = $holiday-
>__get(SpecialDays::APPROVED_BY_USER_ID);

                    if ($approved) {
                        $approved = $this->userRepo-
>getById($approved);
                    }

                    $days[$person->__get(Users::ID)][] = [
                        'id' => $holiday->__get(SpecialDays::ID),
                        'date' => $dateCheck->format('Y-m-d'),
                        'isEmpty' => false,
                        'from' => $holiday-
>__get(SpecialDays::FROM),
                        'to' => $holiday->__get(SpecialDays::TO),
                        'specialDayTypeId' => $type,
                        'approvedByUser' => $approved,
                        'isBetweenMonths' =>
                            $dateCheck->copy()->addMonth()-
>startOfMonth() ==
                            $this->clock::parse($holiday-
>__get(SpecialDays::TO))->startOfMonth() ||
                            $holiday['isBetweenMonths']
                    ];
                } else {
                    $days[$person->__get(Users::ID)][] = [
                        'date' => $dateCheck->format('Y-m-d'),
                        'isEmpty' => true
                    ];
                }

                if ($holiday && $dateCheck->format('Y-m-d')
== $holiday->__get(SpecialDays::TO)) {
                    array_shift($userHolidays[$person-
>__get(Users::ID)]);
```

```
                $holiday = $userHolidays[$person-
>__get(Users::ID)][0] ?? null;
            }
        }
    }

    return response()->json([
        'specialDays' => $days ?? [],
        'headerDays' => $headers ?? [],
    ])->setStatusCode(200);
}

public function store(Request $req): JsonResponse
{
    try {
        if (!$this->holidayRepo->validateSpecialDays(
            $req->input('user_id'),
            $req->input('date_from'),
            $req->input('date_to')
        )) {
            if ($this->holidayAmountRepo-
>countDays($req)) {
                $this->holidayRepo->create([
                    'user_id' => $req->input('user_id'),
                    'from' => $req->input('date_from'),
                    'to' => $req->input('date_to'),
                    'special_day_type_id' => $req-
>input('special_day_type_id'),
                    'approved_by_user_id' => $req-
>input('approved_by_user_id') ?? null
                ]);
            } else {
                return response()->json([
                    'message' => [
                        'status' => 'error',
                        'text' => 'Requested days amount is not
accessible!'
                    ]
                ])->setStatusCode(400);
            }

            return response()->json([
                'message' => [
                    'status' => 'success',
                    'text' => 'Special day successfully created!'
                ]
            ])->setStatusCode(201);
        }
    } catch (Exception $ex) {
        $this->log->error($ex->getMessage());
        return response()->json([
            'message' => [
                'status' => 'error',
                'text' => 'Something went wrong!'
            ]
        ])->setStatusCode(500);
    }

    return response()->json([
        'message' => [
            'status' => 'error',
            'text' => 'Special day create failed!'
        ]
    ])->setStatusCode(400);
}

public function create(): JsonResponse
{
    return response()->json([
        'message' => [
            'status' => 'info',
            'text' => 'Useless route'
        ]
    ])->setStatusCode(418);
}

public function show(): JsonResponse
{
    return $this->create();
}

public function update(Request $req, $id = null):
JsonResponse
{
    try {
        if ($this->holidayRepo->validateSpecialDays(
            $req->input('user_id'),
            $req->input('date_from'),
            $req->input('date_to')
        )) {
            return response()->json([
                'message' => [
                    'status' => 'warning',
                    'text' => 'Special day update failed!'
                ]
            ])->setStatusCode(202);
        }

        $updates = [];
        foreach (['date_from' => 'from', 'date_to' => 'to',
'special_day_type_id' => 'special_day_type_id', 'user_id' =>
'user_id', 'approved_by_user_id' => 'approved_by_user_id']
as $input => $field) {
            if ($val = $req->input($input)) {
                $updates[$field] = $val;
            }
        }

        if ($updates) {
            $record = $this->holidayRepo->getById($id);
            $counted = $this->holidayAmountRepo-
>countDays($req, $record, false);
            $amountRow = $this->holidayAmountRepo-
>where('user_id', $req->input('user_id'))
                ->where('special_day_type', $req-
>input('special_day_type_id'))
                ->get()->first();

            if ($counted['status']) {
                $record->update($updates);
                $this->holidayAmountRepo-
>revertDays($record)
                    ->updateById($amountRow->__get('id'),
['amount' => $counted['availableDays'] -
$counted['daysCount']]);

                return response()->json([
                    'message' => [
                        'status' => 'success',
                        'text' => 'Special day successfully
updated!'
                    ]
                ])->setStatusCode(202);
            }
            return response()->json([
                'message' => [
                    'status' => 'error',
                    'text' => 'Requested days amount is not
accessible!'
                ]
            ])->setStatusCode(400);
        }
```

```
            return response()->json([
                'message' => [
                    'status' => 'info',
                    'text' => 'Special day not changed!'
                ]
            ])->setStatusCode(204);
        } catch (Exception $ex) {
            $this->log->error($ex->getMessage());
            return response()->json([
                'message' => [
                    'status' => 'error',
                    'text' => 'Special day update failed!'
                ]
            ])->setStatusCode(304);
        }
    }

    public function destroy($id = null): JsonResponse
    {
        try {
            $special = $this->holidayRepo->getById($id);
            $this->holidayAmountRepo->revertDays($special);
            $this->holidayRepo->deleteById($id);

            return response()->json([
                'message' => [
                    'status' => 'success',
                    'text' => 'Special day successfully deleted!'
                ]
            ])->setStatusCode(200);
        } catch (Exception $e) {
            $this->log->error($e->getMessage());
            return response()->json([
                'message' => [
                    'status' => 'error',
                    'text' => 'Special day delete failed!'
                ]
            ])->setStatusCode(400);
        }
    }

    public function edit(): JsonResponse
    {
        return $this->create();
    }

    private function limitValue(int $val, int $min, int $max):
int
    {
        return $val < $min ? $min : min($val, $max);
    }
}
```

СУПРОВІДНІ ДОКУМЕНТИ

Завідувачу кафедри інженерії програмного
забезпечення проф. Леоніду БЕДРАТЮКУ
здобувача вищої освіти
Пляцика Олександра Віталійовича
факультет ІТ, ІІІ курс, група ІПЗс-22-1

ЗАЯВА

З правилами чинного Положення про систему забезпечення академічної доброчесності в Хмельницькому національному університеті, згідно з яким виявлення академічного плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту і застосування заходів дисциплінарної та академічної відповідальності, ознайомлений. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на наявність академічного плагіату оповіщений та надаю свою згоду на обробку й збереження університетом моєї роботи в інституційному репозитарії Хмельницького національного університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-обчислювального комплексу StrikePlagiarism та/або програмно-технічного засобу AntiPlagiarism і використання роботи для виявлення академічного плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення текстових збігів у роботах.

Робота надається для перевірки в електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

01.06.2025
дата

підпис

Anti-Plagiarism (UA) v-15.281 Educational

The maximum coincidence with one document: 3.0%

Dictionaries check: en_US, ru_RU, uk_UA. Errors in the document: 14%

ID: 245736 Title: БКР_[illegible] Added to a DB: 2025-06-05 Authors: [illegible] Heads: [illegible] Consultants: Opponents:	Document		Sum coincidence in the DB	
	Symbols	Lexemes	Symbols	Lexemes
	48234	1046	3578 (5%)	58 (5%)

Plagiarism sources

ID	Description	Plagiarism presence in the document	
		Symbols	Lexemes

Протокол аналізу звіту подібності науковим керівником

Заявляю, що я ознайомився (-лась) з Повним звітом подібності, який був згенерований Системою виявлення і запобігання плагіату щодо роботи:

Автор: Пляцик О

Співавтор:

Назва: БКР_Серверна частина системи управління взаємовідносинами з клієнтами

Науковий керівник:

Підрозділ: Кафедра інженерії програмного забезпечення

Коефіцієнт подібності 1:0.3%

Коефіцієнт подібності 2:0%

Мікропробіли: 0

Заміна букв: 651

Інтервали: 0

Білі знаки: 0

Дата створення звіту: 2025-06-05 22:54:44.0

Після аналізу Звіту подібності констатую наступне:

☑ **Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається.**

☐ **Запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Таким чином робота повертається на доопрацювання.**

☐ **Виявлено запозичення і плагіат або навмисні текстові спотворення (маніпуляції), як передбачувані спроби укриття плагіату, які роблять роботу невідповідною вимогам законодавства (Ст. 32. ЗУ Про вищу освіту, пункт 3.1, Ст. 42. ЗУ Про освіту) та вимог НАЗЯВО (Критерій 5), а також кодексу етики і процедурам. Таким чином робота не приймається.**

Обгрунтування:

Дата 10.06.2025 експерт

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ
освітнього ступеня «Бакалавр»

Дипломник Пляцик Олександр Віталійович

Тема Серверна частина системи управління взаємовідносинами з клієнтами

Спеціальність 121 – Інженерія програмного забезпечення

Обсяг кваліфікаційної роботи:

Кількість листів креслень 1; кількість сторінок записки 76

1.Короткий зміст пояснювальної записки та прийнятих рішень У кваліфікаційній роботі проведено аналіз предметної області CRM-систем, досліджено актуальні вимоги до програмного забезпечення такого типу. Визначено функціональні та нефункціональні вимоги до серверної частини системи управління взаємовідносинами з клієнтами. Було проаналізовано існуючі рішення на ринку, виявлено їхні переваги та недоліки. На основі цього обґрунтовано необхідність розробки власного рішення. Вибрано сучасні інструменти та архітектуру для реалізації серверної частини, виконано програмну реалізацію модуля. Розроблену систему протестовано, результати тестування підтвердили працездатність та відповідність функціональним вимогам.

2. Висновок про відповідність роботи поставленому завданню Кваліфікаційна робота виконана відповідно до поставленого завдання, охоплює всі етапи розробки серверної частини CRM-системи та відповідає технічному завданню.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки та передових методів роботи У вступі обґрунтовано актуальність теми, сформульовано мету, задачі та об'єкт дослідження. У першому розділі здійснено огляд предметної області, проведено аналіз існуючих CRM-рішень та сформовано вимоги до розробки. У другому розділі досліджено сучасні архітектурні підходи, обґрунтовано вибір клієнт-серверної архітектури з REST API та базою даних. У третьому розділі реалізовано серверну частину: описано структуру коду, реалізовано основні API-ендпоінти, налаштовано взаємодію з базою даних, проведено модульне тестування та оцінено результат. Робота базується на сучасних технологіях, таких як Laravel, MySQL тощо.

4. Позитивні сторони роботи Тематика є актуальною, адже CRM-системи залишаються важливою складовою автоматизації бізнес-процесів. У роботі реалізовано сучасні технічні рішення, дотримано принципів безпеки, масштабованості та структурованості коду. Документація до API є зрозумілою та відповідає сучасним стандартам розробки.

5. Негативні сторони роботи Робота охоплює лише серверну частину без інтеграції з клієнтським інтерфейсом, що ускладнює повну демонстрацію функціоналу. Також було б доцільно реалізувати логування дій користувачів та більше автоматизованих тестів.

6. Оцінка графічного оформлення та пояснювальної записки Графічний матеріал представлений у вигляді діаграм архітектури, моделей бази даних та схем взаємодії компонентів. Пояснювальна записка оформлена відповідно до діючих стандартів, має логічну структуру та достатню глибину розкриття теми.

7. Відгук про кваліфікаційну роботу в цілому Кваліфікаційна робота виконана на достатньо високому рівні. Вона структурована, змістовна, відображає глибоке розуміння предметної області. Технічна реалізація відповідає сучасним вимогам до розробки серверного програмного забезпечення.

8. Інші зауваження Було б доцільно додати деяку інформацію про роботу інтеграції із фронтендом або мобільним клієнтом для повноцінної демонстрації можливостей системи.

9. Оцінка кваліфікаційної роботи Кваліфікаційна робота виконана у повному обсязі, відповідає поставленій задачі та заслуговує на оцінку «добре».

РЕЦЕНЗЕНТ (ПІБ, посада, місце роботи) Юрій Клюєв, к.т.н. зав. кафедри кібербезпеки ХНУ

"9" 06 2025 р. _____
(підпис)

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ
КАФЕДРИ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатами звіту/звітів перевірки роботи, продукованими програмно-технічним засобом (ами), на наявність текстових збігів.
Назва кваліфікаційної роботи: «Серверна частина системи управління взаємовідносинами з клієнтами»
Автор: Пляцик Олександр Віталійович
Освітня програма: Освітньо-професійна програма «Інженерія програмного забезпечення»
Спеціальність: 121 – Інженерія програмного забезпечення
Науковий керівник: Онишко Оксана Григорівна, кандидат педагогічних наук, доцент
Після аналізу звіту/звітів зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається до захисту.	**відповідає**
2	Виявлені запозичення не є академічним плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота приймається до захисту, але має бути відкоригована.	
3	Виявлені запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Виявлені запозичення частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнуті. Робота може бути допущена до захисту після того, як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття текстових запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
5	Інше:	

Підтвердження:
Запозичення, виявлені у роботі, є законними і не є плагіатом, оскільки:
1) у тексті кваліфікаційної роботи системою виявлення і запобігання плагіату Anti-Plagiarism виявлено схожість з деякими документами у частині загальновживаних обов'язкових словосполучень, що зустрічаються у типовій технічній документації, а також у структурі змісту, назвах розділів/підрозділів, що є характерним для робіт даного напряму. Дані свідчать про використання поширених фраз, технічної термінології та елементів, які не становлять предмету авторського права;
2) запозичення, виявлені у тексті роботи, є фрагментарними.
Максимальний обсяг запозичень, визначений системою Anti-Plagiarism, становить 3.0% з одного джерела. Загальна сумарна подібність у базі даних складає 5% за символами та 5% за лексемами. Крім того, за результатами додаткового аналізу коефіцієнт подібності 1 становить 0.3%, коефіцієнт подібності 2 - 0%. Не виявлено мікропробілів, зайвих білих знаків або маніпуляцій з інтервалами, а зафіксована заміна символів (651 випадок) не впливає на змістовну унікальність тексту.

Дата 10.06.25

Завідувач кафедри _____ Леонід БЕДРАТЮК

Гарант освітньої програми _____ Леонід БЕДРАТЮК

Керівник кваліфікаційної роботи _____ Оксана ОНИШКО