Хмельницький національний університет
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛАФІКАЦІЙНА РОБОТА

Кравчука Ільї Віталійовича
Прізвище, ім'я, по батькові студента

на здобуття ступеня вищої освіти Бакалавра

Клієнтська частина вебплатформи з управління персоналом компанії
Назва теми

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

КПІІ.2201124.01.03.ПЗ

Виконав студент ІІІ курсу, група ІПЗс-22-1 _____ Ілья КРАВЧУК
Підпис Ім'я, ПРІЗВИЩЕ

Керівник канд. пед. наук, доцент _____ Оксана ОНИШКО
Науковий степінь, звання Підпис Ім'я, ПРІЗВИЩЕ

Нормоконтролер канд. техн. наук, доцент _____ Оксана ЯШИНА
Науковий степінь, звання Підпис Ім'я, ПРІЗВИЩЕ

До захисту допускаю:
Завідувач кафедри інженерії
програмного забезпечення _____ Леонід БЕДРАТЮК
Підпис Ім'я, ПРІЗВИЩЕ

10 червня 2025р.

Хмельницький 2025

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій

Кафедра Інженерія 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

Зав. кафедри ІПЗ

Підпис Бедратюк Л.П.
Прізвище, ініціали

2 01 2025 р.

ЗАВДАННЯ НА

КВАЛІФІКАЦІЙНУ РОБОТУ

Кравчуку Ільї Віталійовичу
Прізвище, ім'я, по батькові студента

1. Тема роботи Клієнтська частина вебплатформи з управління персоналом компанії

Керівник роботи Онишко О.Г. канд. пед. наук, доцент
Прізвище, ініціали, науковий ступінь, вчене звання

Затверджено 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) Вступ. Дослідження предметної області та постановка задачі. Проектування програмного забезпечення. Програмна реалізація та тестування застосунку. Висновки.

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень) «Діаграма способів використання», «Діаграма класів», «Діаграма компонентів», «Діаграма розгортання».

6. Консультанти розділів кваліфікаційної роботи

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» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів (розділів) кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Ознайомлення з тематикою дипломного проектування, визначення та узгодження індивідуальних тем кваліфікаційних робіт (КвР)	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 переліку джерел посилання та додатків. Оформлення пояснювальної записки КвР згідно вимог	01.05 – 25.05.2024	
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 _____ Ілья КРАВЧУК
Підпис Ім'я, ПРІЗВИЩЕ

Керівник роботи _____ Оксана ОНИШКО
Підпис Ім'я, ПРІЗВИЩЕ

АНОТАЦІЯ

Тема кваліфікаційної роботи: «Клієнтська частина вебплатформи з управління персоналом компанії».

Автор роботи: Кравчук Ілья Віталійович.

Керівник роботи: Онишко Оксана Григорівна.

Пояснювальна записка: 101 ст., 41 рис., 2 дод., 41 джерело.

Графічна частина: 17 презентаційних слайдів.

Метою кваліфікаційної роботи є розробка клієнтської частини вебзастосунку, призначеного для керівників і менеджерів компаній, які прагнуть підвищити ефективність управління персоналом. Основний акцент зроблено на створенні інтерфейсу, що забезпечує зручну взаємодію з модулями системи.

У процесі виконання роботи було здійснено ґрунтовний аналіз предметної області, що охоплює як загальні принципи роботи CRM-систем, так і особливості їх застосування в галузі IT. Визначено перелік функціональних і нефункціональних вимог, що стали основою для побудови архітектури системи та логіки взаємодії її складових.

Розробку програмного забезпечення здійснено із застосуванням сучасних підходів до побудови вебсистем. Клієнтська частина створена на основі бібліотеки React[4], що дозволяє будувати динамічні компоненти з високим рівнем відгуку. Оформлення здійснено з використанням TailwindCSS[13], що забезпечує гнучкість стилізації при збереженні єдиного візуального підходу. Збірка клієнтської частини здійснювалась з використанням Webpack[15], що оптимізує процес розробки й мінімізує час завантаження інтерфейсу.

Уся система є придатною до подальшого масштабування та вдосконалення, що дозволяє легко реалізовувати нові функції або інтегрувати зовнішні сервіси в межах наступних ітерацій розвитку програмного продукту.

10.06.2025
Дата

Підпис

ВІДОМІСТЬ ДОКУМЕНТІВ

№ рядка	Формат	Позначення документа	Найменування документа	К-сть аркушів	№ Екз.	Примітка
			Текстові документи			
1	A4	КПІІ.2201124.01.03.ПЗ	Пояснювальна записка	101		
2	A4		Завдання на Дипломний проект	1		
3	A4		Анотація	1		
			Графічні документи			
4	A4		Презентаційні слайди	18		
5	A3	КПІІ.2201124.01.03.Е8	Діаграма Використання	1		
6	A3	КПІІ.2201124.01.03.Е8	Діаграма Станів	1		
7	A3	КПІІ.2201124.01.03.Е8	Діаграма Класів	1		
8	A3	КПІІ.2201124.01.03.Е8	Діаграма Компонентів	1		
9	A3	КПІІ.2201124.01.03.Е8	Діаграма Розгортання	1		

Змін.	Аркуш	№ докум.	Підпис	Дата
Виконав		Кравчук І.В.		10.06
Керівник		Онишко О.Г.		10.06
Н. контр.		Яшина О.М.		10.06
Зав. каф.		Бедратюк Л.П.		10.06

КПІІ.2201124.01.03.ВД

Клієнтська частина вебплатформи з управління персоналом компанії

Відомість Документів

Літ	Аркуш	Аркушів
	1	1

ХНУ. ІПЗс-22-1

ЗМІСТ

ВСТУП

У сучасних умовах ведення бізнесу, особливо в галузі інформаційних технологій, компанії дедалі частіше стикаються з нагальною потребою впорядкування процесів зберігання, обробки та структуризації даних, що стосуються їхнього кадрового складу. Зі збільшенням кількості працівників у штаті компанії відбувається ускладнення механізмів внутрішнього управління, зростає ризик дублювання або втрати інформації, а сам процес взаємодії між відділами стає менш контрольованим і повільним. У багатьох випадках організації намагаються вирішити ці проблеми шляхом застосування декількох окремих онлайн-сервісів. Проте більшість таких інструментів, особливо якщо вони є безкоштовними або мають обмежений функціонал, не здатні повною мірою задовольнити комплексні вимоги компаній до управління персоналом. Саме тому виникає необхідність у впровадженні централізованих CRM-систем, призначених для повного охоплення внутрішніх процесів, що пов’язані з обліком працівників, автоматизацією рутинних задач і формалізацією внутрішніх комунікацій [2].

CRM, або Customer Relationship Management, є не лише назвою програмного забезпечення, а й більш широким поняттям, що позначає концепцію ведення бізнесу, при якій клієнт або користувач ставиться у центр усіх бізнес-процесів. У широкому сенсі, CRM об’єднує в собі стратегії, підходи, методики, інструменти та технологічні засоби, що дозволяють не просто залучати нових клієнтів, а й формувати з ними довгострокові відносини, спрямовані на підвищення лояльності та довіри. Незважаючи на те, що початково ці системи розроблялися для зовнішніх комунікацій, зокрема для покращення процесів продажів, обслуговування клієнтів і маркетингу, з часом вони знайшли своє застосування і в контексті внутрішнього управління персоналом, що стало надзвичайно актуальним у великих компаніях.

Застосування CRM як внутрішнього управлінського інструмента дає змогу формувати єдину екосистему для обміну інформацією, організації робочих процесів, а також швидкого доступу до необхідних даних усередині компанії. Це,

своєю чергою, забезпечує оперативність прийняття рішень, зменшує кількість помилок, сприяє ефективному делегуванню повноважень і підвищує загальну продуктивність колективної роботи. У межах будь-якої організації, незалежно від її масштабу чи напрямку діяльності, CRM-система може бути однаково корисною для відділів управління персоналом, маркетингу, продажів, технічної підтримки або адміністративного забезпечення.

Коли чисельність працівників перевищує позначку у двадцять осіб, виникає об'єктивна необхідність відмовитися від хаотичних підходів до управління ресурсами на користь централізованих та автоматизованих рішень. CRM дозволяє формалізувати подання заявок, планування зустрічей, організацію розкладів, відстеження результатів роботи та створення внутрішніх звітів без потреби в сторонніх сервісах. Це істотно знижує навантаження на адміністративний персонал і керівників, мінімізує ризик людського фактору та підвищує рівень прозорості усіх внутрішніх процесів.

Розроблена в рамках даної кваліфікаційної роботи CRM-система реалізована у вигляді вебзастосунку, який забезпечує швидку та зручну взаємодію між користувачами без потреби встановлення додаткового програмного забезпечення на локальні пристрої. Такий підхід дозволяє уникнути надмірного навантаження на комп’ютерну інфраструктуру користувача, спрощує доступ до системи з будь-якого сучасного браузера і знижує витрати на технічне обслуговування.

З погляду технічної реалізації, створена інформаційна система базується на сучасному, стабільному та добре зарекомендованому технологічному стеку, який відповідає актуальним вимогам до розробки корпоративного програмного забезпечення. Серверна частина програмного рішення була реалізована з використанням фреймворку Laravel[9], який підтримує модель архітектури MVC (Model-View-Controller), що дозволяє чітко розмежувати бізнес-логіку, представлення даних та обробку запитів. Такий підхід забезпечує не лише зручність подальшої підтримки і розширення коду, а й підвищує структурованість розробки в цілому.

Фронтенд або клієнтська складова системи створена із залученням React.js[4] – популярної JavaScript-бібліотеки, що дає змогу ефективно створювати складні, інтерактивні інтерфейси користувача з реактивною логікою та оновленням компонентів без перезавантаження сторінки. Реалізація інтерфейсу з дотриманням сучасних стандартів користувацького досвіду була забезпечена завдяки використанню CSS-фреймворку TailwindCSS[13], який надає можливість швидко й гнучко компонувати стилі за допомогою утилітарних класів, що особливо зручно в умовах швидких змін вимог до дизайну.

Процес безпосередньої розробки здійснювався у професійному інтегрованому середовищі розробки PhpStorm[12], яке забезпечує розширену підтримку мов програмування PHP[9] та JavaScript[4], а також інтеграцію з системами контролю версій і тестування. Для ефективного управління зовнішніми бібліотеками та залежностями використовувалися менеджери пакетів Composer (для PHP-компонентів) та NPM (для JavaScript-модулів), що значно спрощувало підтримку актуального програмного середовища.

Для ізоляції програмних компонентів і забезпечення однакових умов запуску в різних середовищах використовувалась технологія контейнеризації Docker[5], яка дозволяє створити стандартизовані контейнери з програмним кодом, залежностями та конфігураціями. Як засіб зберігання структурованих даних було обрано систему управління базами даних MySQL[11].

Збирання та оптимізація фронтенд-коду здійснювались за допомогою Webpack[15] – інструменту, який виконує роль модулятора та компілятора, що дозволяє ефективно структурувати файли, зменшувати об’єм ресурсів і покращувати швидкодію інтерфейсу під час завантаження у браузері. Використання Webpack також дозволило впровадити автоматичну збірку та перевірку коду в процесі розробки.

Створене програмне рішення є прикладом реалізації CRM-системи, що орієнтована на автоматизацію внутрішніх бізнес-процесів у межах IT-компанії. вебзастосунок сприяє цифровій трансформації управлінських підходів, у сфері діяльності організації.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Визначення предметної області, її функціональних і структурних особливостей

У сучасних реаліях розвитку бізнесу значна частина компаній, незалежно від галузі діяльності, активно впроваджує цифрові інструменти для автоматизації внутрішніх процесів. Зокрема, у сфері управління персоналом та координації командної взаємодії, керівники підприємств і організацій усе частіше стикаються з потребою у централізованому рішенні, яке дозволяє не лише фіксувати, а й систематизувати інформацію про працівників, процеси та завдання. Через надмірне навантаження і зростання кількості працівників збирати персональні дані вручну або за допомогою розрізнених сервісів стає малоефективно. Саме тому актуальним є створення CRM-системи, яка надає змогу кожному працівнику самостійно вводити, редагувати та оновлювати дані без залучення керівництва.

Предметною областю дослідження даного дипломного проекту виступає цифрове управління IT-компанією з використанням CRM-платформи, яка дозволяє організувати облік, збереження і обробку даних, а також забезпечити самостійну участь працівників у веденні внутрішньої документації. У межах розробки передбачається впровадження функціоналу, який дає змогу користувачу створювати облікові записи, оновлювати особисту інформацію, взаємодіяти з колективом, планувати зустрічі, оформлювати заявки на відпустку та ініціювати внутрішні звернення – усе це без необхідності безпосереднього втручання керівника.

Для компаній, бізнес-модель яких заснована на роботі з великою кількістю взаємодій, незалежно від того, чи це обслуговування клієнтів, чи комунікація між відділами, ключовим є контроль за кожним етапом процесу. Ведення баз даних, фіксація історії комунікацій, відстеження запитів, моніторинг задач, призначення відповідальних осіб, планування робочих зустрічей і координація дій – усе це вимагає єдиного простору для ефективного управління. Раніше для вирішення

таких задач застосовувались окремі інструменти: електронна пошта, месенджери, таблиці, календарі, таск-менеджери. Така фрагментація спричиняла труднощі з контролем і призводила до втрати частини інформації.

CRM-системи були створені з метою об'єднання цих функцій у єдиному інтерфейсі. Застосування подібної системи дозволяє працівникам позбутись потреби постійного перемикання між додатками. Вся основна робота концентрується в межах однієї інтегрованої платформи, що істотно зменшує час на виконання рутинних дій. Завдяки широким можливостям автоматизації CRM-системи здатні охопити повний цикл внутрішньої діяльності підприємства – від генерації завдань і нагадувань до відправки сповіщень, оформлення шаблонних документів, планування зустрічей і створення звітів.

Такі функції, як масове надсилання листів, формування автоматичних відповідей, налаштування робочих процесів за подіями, розподіл задач за виконавцями з можливістю пріоритезації – усе це забезпечується в рамках однієї платформи. Уніфікована логіка управління дозволяє уникнути людських помилок, знижує ризик втрати даних і дає змогу ефективно контролювати виконання поставлених завдань. Окремо варто виділити механізм моніторингу воронки продажів або прогресу завдань – у CRM це реалізовано у вигляді інтерактивної аналітики, що адаптується до потреб конкретного бізнесу.

Універсальність CRM-систем полягає в їхній здатності адаптуватися до специфіки різних підприємств, незалежно від розміру штату, структури чи галузевої приналежності. Поширена помилкова думка, ніби такі системи призначені виключно для масштабних корпорацій. Насправді ж, невеликі компанії з гнучкою структурою та розподілом функцій між співробітниками часто отримують ще більший ефект від впровадження CRM, оскільки потребують автоматизації не менше, ніж великі бізнеси.

Під час вибору CRM-рішення особливу увагу варто приділяти низці важливих аспектів, таких як загальний рівень зручності користування, гнучкість у налаштуванні функціоналу під індивідуальні потреби, а також наявність адаптованої мобільної версії, що дозволяє менеджерам і керівникам бути на

зв’язку з робочими процесами у будь-який час. Важливо також оцінити складність інтеграції: якщо система потребує тривалого навчання або втручання спеціалістів для налаштування, процес адаптації персоналу може сповільнитись. Гнучке CRM-рішення має дозволяти змінювати назви, структуру даних, правила комунікації – усе відповідно до еволюції бізнесу.

Мобільна версія є критично необхідною для спеціалістів, які працюють віддалено або виконують обов’язки за межами офісу. Без неї втрачається оперативність внесення даних, що знижує актуальність інформації в системі. Для керівників підприємств, що динамічно розвиваються, CRM є ключовим інструментом для постійного моніторингу стану справ і швидкого реагування на зміни. Варто також ураховувати складність інсталяції та освоєння системи – якщо CRM вимагає залучення сторонніх фахівців і тривалого навчання персоналу, впровадження може стати непотрібно обтяжливим процесом, що стримуватиме розвиток.

1.2 Аналіз наявних рішень у вигляді програмно забезпечення даної предметної області

Для того щоб створена система максимально відповідала завданням, поставленим у межах даного дипломного дослідження, а також була зручною, адаптованою до реальних умов роботи IT-компанії та містила актуальні функції, доцільно провести попередній аналіз уже існуючих на ринку вебзастосунків, орієнтованих на управління завданнями, персоналом та внутрішніми комунікаціями в межах корпоративного середовища.

Одним із найбільш поширених прикладів такого програмного забезпечення є онлайн-сервіс Trello[14]. Даний застосунок призначений для організації командної роботи шляхом створення візуальних дошок із завданнями, які можуть бути згруповані за статусами, відповідальними особами або етапами виконання. Завдяки інтуїтивно зрозумілому інтерфейсу та гнучкій логіці взаємодії, система Trello активно використовується у багатьох компаніях для координації проектної діяльності. Основною концепцією цієї платформи є контроль за виконанням задач

у режимі реального часу, що значно спрощує моніторинг поточного стану справ. На рисунку 1.2.1 наведено зовнішній вигляд вебзастосунку Trello.

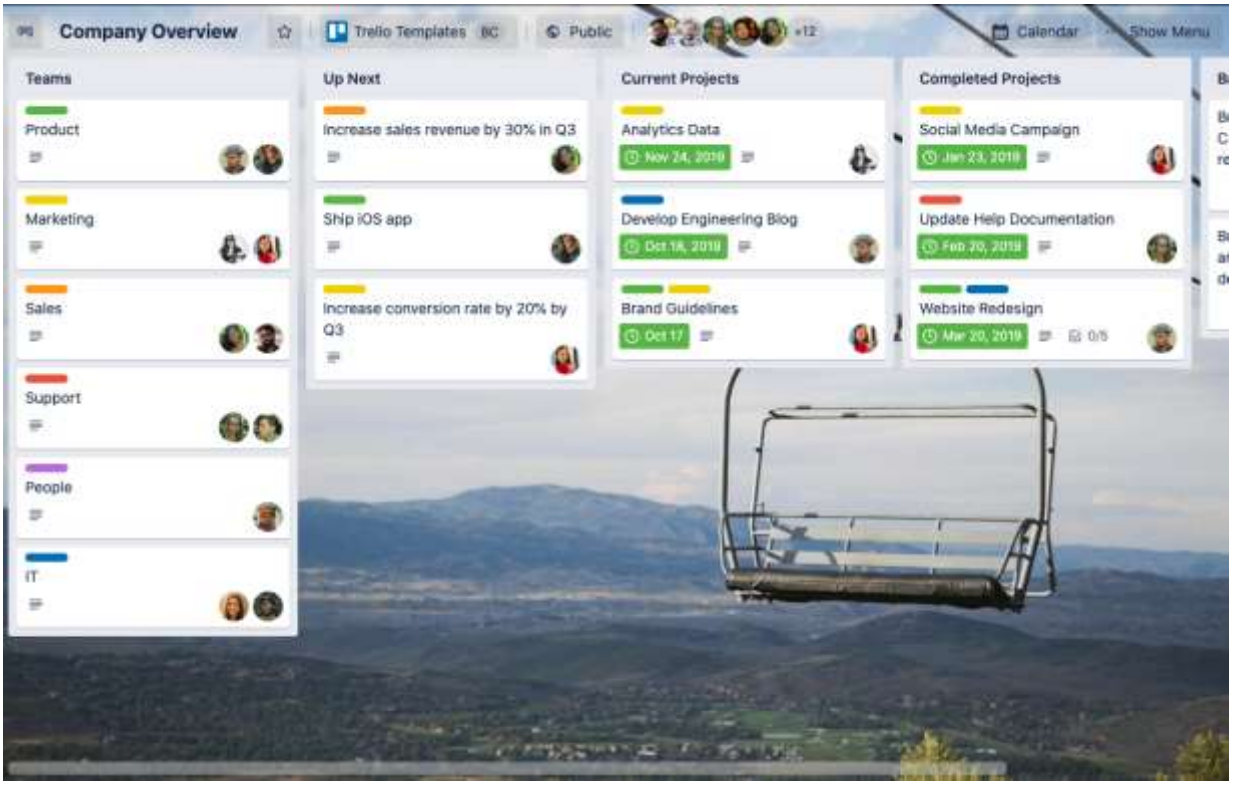


Рисунок 1.2.1 – Вигляд вебдодатку Trello

Іншим потужним інструментом є багатофункціональна CRM-платформа Bitrix24[1]. Це рішення поєднує в собі не лише класичний функціонал системи управління взаєминами з клієнтами, а й численні додаткові модулі, включаючи таск-менеджмент, документообіг, аналітику, відеозв'язок, інтернет-магазини, календарі та інші засоби внутрішньої взаємодії. Bitrix24 активно впроваджується в компаніях середнього та великого бізнесу завдяки широким можливостям масштабування. Проте, однією з основних складностей використання цієї платформи є її загальна складність та перевантаженість інтерфейсу, що може ускладнити процес адаптації, особливо для працівників, які не мають достатнього досвіду роботи з подібними цифровими інструментами. На рисунку 1.2.2 зображено приклад інтерфейсу системи Bitrix24.

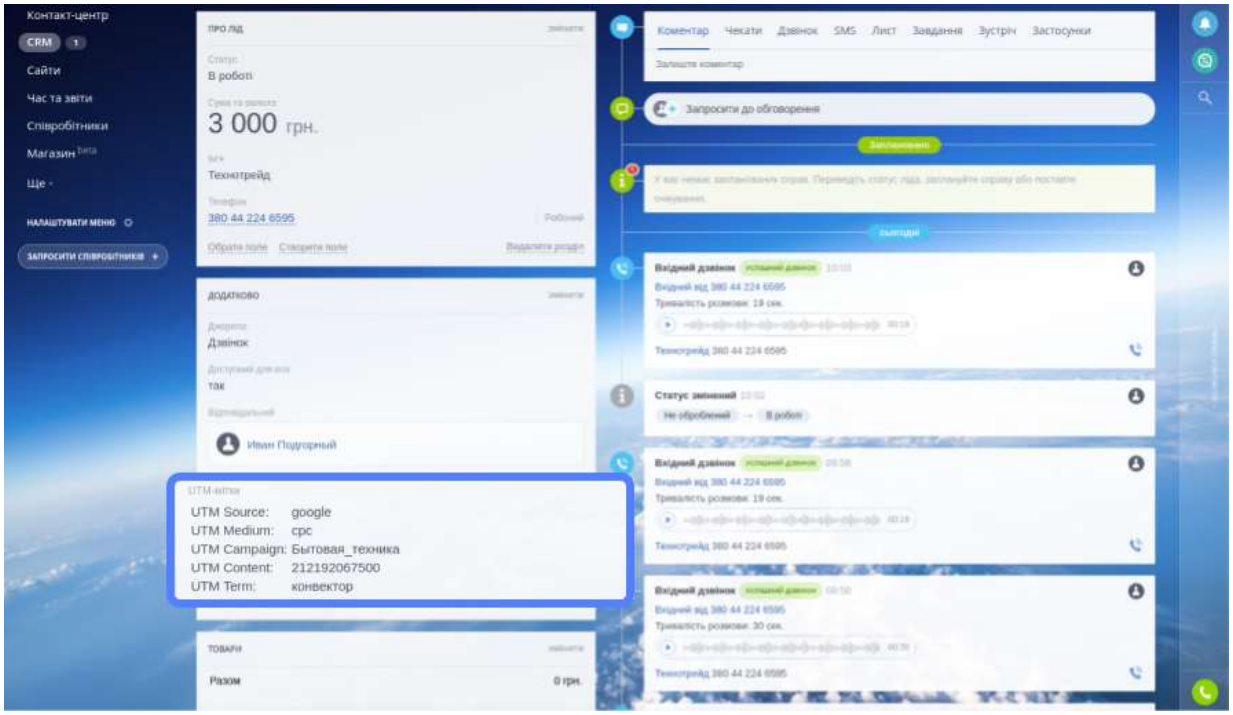


Рисунок 1.2.2 – Вигляд платформи Bitrix24

Серед інших існуючих рішень на ринку можна знайти як надто спрощені сервіси, що не забезпечують необхідного рівня функціональності, так і дороговартісні корпоративні продукти, які мають надмірний набір можливостей, не завжди затребуваних у конкретних умовах. У таких випадках важливо знайти баланс між функціональністю, зручністю використання, адаптивністю до змін і фінансовою доцільністю впровадження. Часто надлишкова функціональність стає не перевагою, а недоліком, оскільки ускладнює навчання персоналу і гальмує впровадження нової системи у поточні робочі процеси.

Беручи до уваги практичний досвід застосування подібних CRM-систем, доцільно адаптувати в межах власної розробки лише ті функції, які справді мають значення для ефективного управління персоналом у невеликій або середній IT-компанії. Так, планується реалізувати ключові компоненти, пов’язані з організацією зустрічей, створенням задач, розподілом обов’язків, поданням запитів, а також формуванням основних звітів. Інші компоненти, які є надто специфічними або мають вузьке призначення, можуть бути реалізовані пізніше – у наступних ітераціях розвитку застосунку, відповідно до реальних потреб користувачів.

Таким чином, огляд аналогів дозволяє визначити не лише сильні сторони наявних рішень, а й типові помилки, яких слід уникати при проектуванні власної системи. Це дає змогу створити практично орієнтований вебзастосунок, який поєднує у собі необхідну гнучкість, простоту використання і достатній функціонал без надмірного ускладнення інтерфейсу.

1.3 Специфікація вимог до програмного забезпечення

У межах даної кваліфікаційної роботи одним із ключових етапів стало проектування та реалізація інтерфейсної частини вебзастосунку. Саме ця частина є відповідальною за взаємодію кінцевого користувача з функціональними модулями системи, обробку введених даних, а також відображення відповідей, що надходять із серверної частини після обробки запитів. Було поставлено завдання створити зручний, логічно структурований інтерфейс, який дозволяє користувачу виконувати дії без необхідності глибоких технічних знань, із забезпеченням миттєвого зворотного зв’язку.

У рамках розробки передбачено створення повноцінного редактора, що забезпечує доступ до наступних модулів системи: управління персоналом, контроль за особливими датами (наприклад, днями народження або професійними ювілеями), інтегрований календар важливих подій, модуль управління проектами, механізми для створення та редагування нотаток, тайм-трекінг, а також система сповіщень та організації внутрішніх зустрічей. Окремо виділено сутності адміністратора та звичайного користувача з розподілом прав доступу. Завдяки цьому підходу уся внутрішня інформаційна взаємодія зосереджується в одному застосунку, без потреби звертатися до зовнішніх сервісів, що суттєво скорочує час виконання дій та знижує рівень технологічної залежності.

До основних вимог, які висуваються до розробленого вебзастосунку, належать: можливість простого розширення функціоналу без необхідності перебудови існуючої структури, гнучкість у налаштуваннях параметрів користувацької поведінки, доступність та зручність інтерфейсу, а також наявність модулів для попереднього перегляду введених даних та кінцевого результату.

Уся інформація, яка генерується або змінюється у процесі роботи, зберігається у централізованій реляційній базі даних, реалізованій на основі СУБД MySQL[11]. При побудові структури бази даних дотримано принципів нормалізації до третьої нормальної форми, що дозволяє мінімізувати дублювання даних, забезпечити логічну цілісність та спростити обслуговування.

Для реалізації серверної логіки використовуються мови програмування PHP[9] та JavaScript[4], а також мова розмітки HTML. Розробка здійснюється у середовищі PhpStorm[12], що надає розширений функціонал для роботи з PHP-кодом, вбудований редактор для HTML, CSS та JavaScript, інтегрований термінал і підтримку систем контролю версій. Застосування препроцесора Less дозволяє структурувати стилі за модульним принципом, що полегшує масштабування проекту.

Фронтенд частина реалізована на основі бібліотеки React[4], яка є ефективним інструментом для створення динамічних інтерфейсів. React дозволяє розбивати сторінку на окремі компоненти, кожен із яких представляє автономну одиницю функціоналу. Така компонентна модель дає змогу гнучко керувати побудовою інтерфейсу, легко оновлювати його частинами без перезавантаження сторінки, а також забезпечити повторне використання коду. Приклад поділу інтерфейсу на компоненти продемонстровано на рисунку 1.3.1.

Рисунок 1.3.1 – Приклад React компонент

HTML у системі використовується для структуризації контенту, тоді як CSS, включно з TailwindCSS[13], відповідає за стилізацію елементів, створення адаптивної верстки, задання кольорових схем, відступів, сіток тощо. TailwindCSS, як утилітарний фреймворк, дозволяє надавати стилі напряму через HTML, що пришвидшує процес розробки та підвищує прозорість коду.

Середовище PhpStorm[12], створене компанією JetBrains, є професійною платформою для веброзробки, яка підтримує основні операційні системи, включаючи Windows, Linux та macOS. Хоча даний продукт є платним, розробники-студенти мають можливість отримати безкоштовну академічну ліцензію, що надає доступ не лише до PhpStorm, а й до інших інструментів компанії JetBrains. Серед переваг середовища варто відзначити підтримку аналізу коду «на льоту», підказки, автоматичне виправлення помилок, а також інтегрований SQL-редактор, який дозволяє створювати, редагувати та перевіряти запити безпосередньо в IDE. Рисунок 1.3.2 демонструє робоче вікно середовища PhpStorm.

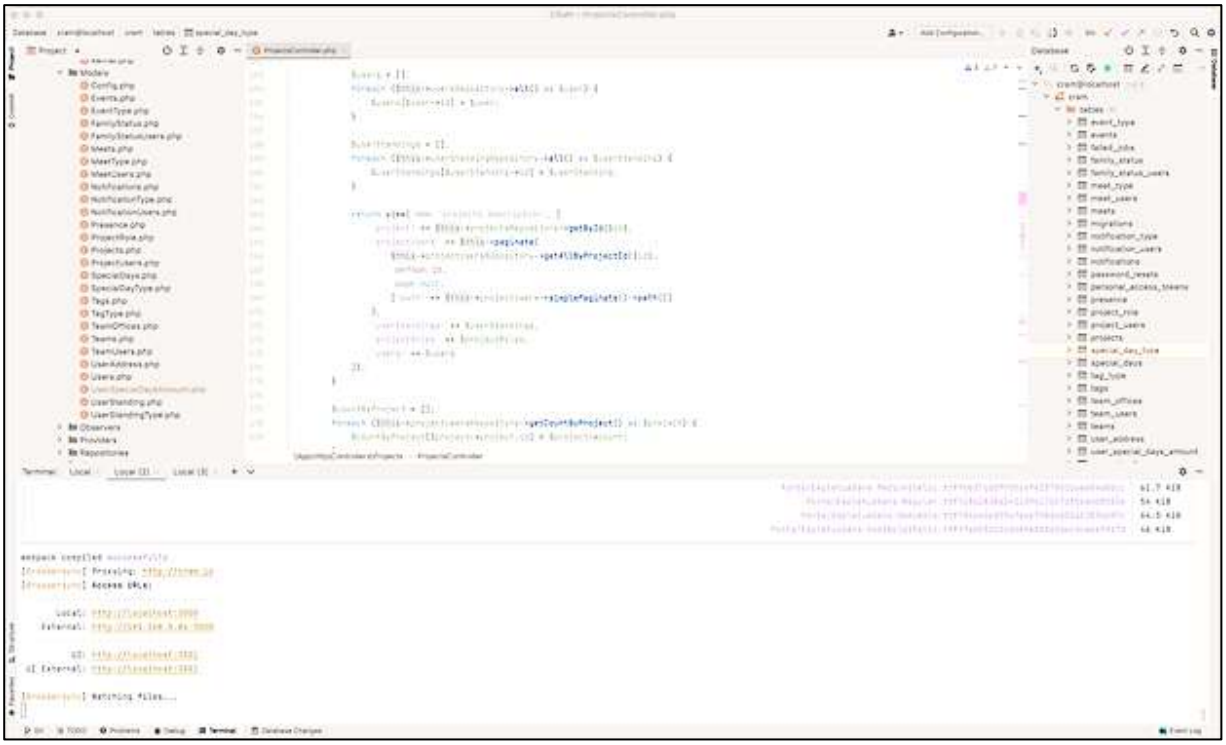


Рисунок 1.3.2 – Вигляд робочого вікна PhpStorm

Для локального розгортання системи й тестування у мережі було обрано середовище XAMPP, яке включає в себе вебсервер (наприклад, Apache або Nginx), інтерпретатор PHP, систему керування базами даних MySQL та інші компоненти. Завдяки кросплатформеній природі (що позначено в абревіатурі літерою "X") дане середовище однаково ефективно працює на Windows, macOS та Linux, а також може бути адаптоване для використання на мобільних пристроях.

Окрему роль у структурі розробки відіграє використання технології Docker[5], яка забезпечує контейнеризацію середовища. Завдяки Docker можна створити відокремлені віртуалізовані середовища (контейнери), кожне з яких виконує певну функцію – наприклад, окремий контейнер для вебсервера, інший – для бази даних. Такий підхід дозволяє уникнути конфліктів між залежностями, пришвидшує налаштування середовища та спрощує розгортання застосунку. Розробка у середовищі, ізольованому від зовнішніх впливів, дозволяє працювати автономно – без необхідності постійного доступу до Інтернету чи зовнішнього хостингу.

Таким чином, вибір технологій, інструментів та середовищ розробки здійснювався з урахуванням вимог до продуктивності, гнучкості, масштабованості та зручності подальшого обслуговування системи, що відповідає найкращим практикам сучасної веброзробки.

1.4 Висновки до першого розділу. Постановка задачі

Ефективна реалізація програмного забезпечення, зокрема у сфері створення корпоративних вебзастосунків, потребує чіткого, формалізованого підходу до постановки задачі. З цією метою доцільним є використання мови моделювання UML (Unified Modeling Language), яка надає широкий спектр інструментів для візуалізації, специфікації, конструювання та документування програмних систем. Діаграми UML можуть застосовуватися на всіх ключових етапах розробки – від початкового аналізу бізнес-вимог до етапів технічного проектування та супроводу.

Використання UML дозволяє створити уніфіковане представлення структури, поведінки та взаємозв’язків між компонентами системи. Завдяки своїй гнучкості та широким можливостям деталізації, UML-діаграми не лише сприяють полегшенню процесу розробки документації, а й слугують ефективним засобом підтримки, супроводу та розширення проекту у майбутньому. У контексті розробки системи управління персоналом на основі CRM-підходу, UML дозволяє візуалізувати сценарії взаємодії користувачів з системою, відобразити можливі стани елементів, уточнити процеси обробки даних, а також моделювати ролі користувачів і їхні права доступу.

Для досягнення поставленої мети було передбачено поділ програмного функціоналу на окремі логічні блоки, кожен з яких відповідає за реалізацію певного напряму роботи. Такий підхід дозволяє дотримуватись принципів модульності, спрощує тестування й обслуговування, а також полегшує подальше масштабування. Крім того, реалізація системи передбачає здатність до роботи з різними типами даних, що обумовлює необхідність впровадження надійних механізмів перевірки вхідних значень, а також обробки виняткових ситуацій.

З метою чіткого опису функціональних сценаріїв було сформовано діаграму використання, створену за допомогою онлайн-інструменту Draw.io[3], яка відображає типові дії користувача під час взаємодії з системою. На відповідній UML-діаграмі подано послідовність дій: користувач переходить на вебсторінку системи, де має можливість пройти процедуру авторизації, за умови, що його облікові дані вже присутні в базі. У разі втрати пароля, користувач має змогу ініціювати процедуру його скидання, після чого отримає лист із посиланням для відновлення доступу. Після успішної авторизації система надає доступ до базових функцій перегляду інформації, а по завершенні сесії користувач може здійснити вихід зі свого облікового запису. Описана послідовність дій також знаходить своє відображення у діаграмі станів, що подана у додатках до цієї роботи. Вона демонструє зміну станів системи залежно від дій користувача, що є важливим для розуміння логіки її роботи з точки зору внутрішніх переходів.

У рамках розробки було визначено дві основні сутності користувачів: звичайний користувач та адміністратор. Повноваження звичайного користувача обмежуються переглядом доступної інформації та ініціацією запитів (наприклад, на вихідний день або участь у події). Натомість адміністратор має повний доступ до всіх розділів системи, може змінювати, створювати, видаляти записи, призначати права доступу іншим користувачам, а також здійснювати контроль за загальним функціонуванням системи.

Окремо необхідно зазначити, що у рамках цього проекту не передбачається реалізація самостійної реєстрації нових користувачів. Реєстрація та створення облікових записів здійснюється виключно адміністративно. Зі сторони користувача доступними є лише перегляд існуючих даних, ініціювання дій у межах дозволеного функціоналу та зміна особистих параметрів профілю в межах встановлених прав.

2 ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Технічне завдання

Повна назва проекту: «CRM система управління IT-компанією».

Тестова назва проекту: «CRaM», що є абревіатурою від англійського словосполучення Customer Relationship and Management, яке в контексті даної системи інтерпретується як «Управління та менеджмент персоналу».

Метою розробки цього програмного продукту є створення функціональної, масштабованої та зручної у використанні інформаційної системи, що забезпечує повноцінне управління персоналом IT-компанії, моніторинг проектної діяльності, внутрішню комунікацію, а також ведення супутньої адміністративної документації у цифровому форматі. Головне функціональне призначення системи полягає у створенні єдиного середовища для оперативної взаємодії між працівниками, керівниками та адміністраторами компанії в режимі реального часу.

Програмне забезпечення повинно реалізовувати комплекс можливостей, до якого належать:

- управління персоналом з доступом до облікових записів, персональних даних, статусів, запитів;
- контроль та реєстрація особливих дат (наприклад, дні народження, професійні ювілеї, інші внутрішні події);
- візуалізація подій за допомогою календаря, доступного для перегляду всіма користувачами системи;
- реалізація функціональної системи сповіщень щодо майбутніх подій, змін статусів завдань або нових повідомлень;
- організація внутрішніх зустрічей з можливістю створення запрошень, додавання учасників та спільного перегляду подій;
- створення заміток і чат-функціонал для обміну короткими повідомленнями між користувачами в межах команди;

– система тайм-трекінгу – фіксація часу, витраченого працівниками на виконання певних завдань або робочих сесій, з метою подальшого аналізу ефективності їхньої діяльності.

Розробка даної системи зумовлена потребою у спрощенні процесу комунікації між керівництвом компанії та її працівниками, мінімізації рутинних адміністративних дій, автоматизації процесів планування й оперативного контролю. За своєю суттю, система має забезпечити швидкий доступ до запланованих подій, завдань, заявок та іншої критично важливої інформації у межах однієї платформи.

Взаємодія в системі відбуватиметься у реальному часі, що забезпечить актуальність інформації, мінімізує затримки в обміні даними та сприятиме підвищенню рівня цифрової дисципліни в команді. Важливим аспектом є акцент на інтуїтивно зрозумілий та зручний для користувача інтерфейс, який не потребує спеціальної технічної підготовки та дозволяє новим користувачам швидко адаптуватися до роботи з системою.

Простота використання, логічна структура, передбачувана поведінка елементів інтерфейсу та оптимізована швидкодія – усі ці якості повинні забезпечити високий рівень користувацького задоволення та сформувати довіру до інструменту в щоденному використанні. Крім того, своєчасні сповіщення про майбутні події мають запобігти пропуску важливих зустрічей чи термінів виконання завдань, що, у підсумку, підвищить рівень відповідальності працівників і покращить загальну ефективність управління у компанії.

2.2 Етапи проектування

Процес проектування програмного забезпечення є фундаментальним етапом у життєвому циклі будь-якої цифрової системи, оскільки саме на цій стадії формуються архітектурні рішення, визначаються технологічні підходи та закладається логіка взаємодії між усіма компонентами майбутнього застосунку. Він охоплює не лише розробку зовнішнього вигляду або алгоритмів, а й включає в себе створення високорівневих структур – таких як архітектура ПЗ, організація

логіки обміну даними, моделі поведінки користувачів та інші елементи системного рівня.

Після визначення мети та специфікації майбутнього продукту, розробник формує концептуальне бачення рішення, яке знаходить своє втілення у проектній документації, візуалізаціях, структурних діаграмах і технічному дизайні. На цьому етапі створюються моделі програмного забезпечення, що описують як логіку внутрішніх процесів, так і взаємодію користувача з інтерфейсом.

У загальному випадку типовий життєвий цикл проектування програмного забезпечення включає такі етапи:

– первинний аналіз проблемної області та збір інформації;
– формалізація функціональних і нефункціональних вимог;
– розробка архітектури та концептуальних рішень;
– безпосередня реалізація технічної частини системи;
– проведення тестування всіх ключових модулів;
– оформлення супровідної документації та розгортання продукту;
– технічне обслуговування, підтримка та оновлення ПЗ.

На початковому етапі особливу увагу було приділено дослідженню предметної області. Ретельний аналіз ринку, аналіз існуючих систем управління персоналом, оцінка поширених проблем, з якими стикаються компанії у своїй щоденній діяльності, дали змогу сформулювати чітке уявлення про функціональні потреби цільової аудиторії. Було з’ясовано, що багато організацій потребують систематизації внутрішньої взаємодії, планування подій, управління часом та персоналом, а також підвищення прозорості робочих процесів.

На наступному етапі формулювання вимог створювалася концептуальна модель майбутнього додатку, яка описує типові сценарії поведінки користувачів та очікувану функціональність. Було визначено набір ключових модулів, обрано технологічний стек для реалізації, а також проведено попередню класифікацію ролей користувачів, що взаємодіятимуть із системою.

На етапі конструювання проекту були визначені способи реалізації поставлених вимог. Саме на цьому етапі розпочалося застосування мови

моделювання UML, яка дозволяє візуалізувати структуру та динаміку поведінки системи. Діаграми моделювалися за допомогою популярного вебінструмента Draw.io[3], який підтримує створення діаграм усіх основних типів і є кросплатформеним.

Діаграма станів, що представлена в додатку, описує всі можливі стани об'єктів системи та переходи між ними під час життєвого циклу. Така діаграма дозволяє моделювати динамічну поведінку користувача або компонентів системи в залежності від зовнішніх та внутрішніх подій.

Діаграма варіантів використання (use case diagram) демонструє основні взаємодії кінцевого користувача з системою: автентифікація, перегляд сторінок користувачів, подій, проектів, особливих днів і зустрічей. Усі ці сценарії моделюють ключові дії, які має реалізувати система вже на початковому етапі впровадження.

Діаграма компонентів дає уявлення про структуру програмного забезпечення на високому рівні абстракції, показуючи, з яких частин складається система та як вони взаємодіють між собою. У проекті визначено такі основні компоненти:

- система керування базами даних MySQL[11] – відповідає за створення, зберігання, зв’язки та цілісність даних у реляційній структурі;
- моделі – описують структуру таблиць бази даних, визначають поля, типи даних, а також методи доступу й конфігурації взаємодії з ними;
- контролери – реалізують обробку вхідних HTTP-запитів, керують логікою застосунку та повертають відповідні відповіді або у вигляді представлень, або у форматі даних;
- шаблони – це динамічні HTML-документи з елементами PHP-коду, що відповідають за відображення інформації користувачам;
- інтерфейсні вебсторінки – надають користувачу змогу взаємодіяти з функціоналом системи через браузер;
- маршрутизатор (роутер) – керує переходами між сторінками, обробляє маршрутизацію URL-адрес та зберігає поточний стан взаємодії.

У межах проекту активно застосовуються принципи об’єктно-орієнтованого програмування. Ключову роль у моделюванні відіграють класи, які формують основу логіки системи. Було реалізовано кілька типів зв’язків між класами: агрегація – коли один клас містить у собі об’єкти інших, але не має повного контролю над їхнім життєвим циклом; композиція – тісніший зв’язок, де об’єкти не можуть існувати поза класом-контейнером, що гарантує структурну цілісність і сприяє інкапсуляції.

Також побудовано діаграму розгортання, яка ілюструє фізичну структуру системи: сервери, вузли, бази даних, а також взаємозв’язки між ними. Ця діаграма дозволяє зрозуміти, як елементи системи будуть розподілені по інфраструктурі після впровадження, які мережеві зв’язки потрібні для їхньої взаємодії та як забезпечується передача даних між фізичними або віртуальними пристроями.

У результаті проектування сформовано чітке уявлення про структуру, поведінку та архітектурну логіку програмного забезпечення. Передбачається, що реалізація відбуватиметься із застосуванням мови програмування PHP на базі фреймворку Laravel[9], із використанням HTML, CSS, JavaScript і середовища PhpStorm[12]. Такий вибір зумовлений гнучкістю, масштабованістю, наявністю численних інструментів для розробки й тестування, а також зручністю інтеграції з сучасними технологіями клієнтського інтерфейсу.

2.3 Проектування інтерфейсу програмного забезпечення

Проектування інтерфейсу є ключовим етапом у процесі розробки сучасного програмного забезпечення. Саме на цьому етапі визначається зовнішній вигляд, структура елементів взаємодії з користувачем, механізми передачі даних, логіка представлення інформації, а також архітектурні рішення, що забезпечують ефективну роботу всіх функціональних складових системи.

Розроблюваний програмний продукт реалізується у вигляді вебзастосунку, що функціонує через браузерний інтерфейс і не потребує встановлення на локальний пристрій. Інтерфейс реалізовано з використанням мов PHP, JavaScript, HTML та CSS, на основі фреймворку Laravel[9]. вебсторінки системи

забезпечують користувачам доступ до основного функціоналу, включаючи керування персоналом, перегляд подій, створення нотаток та комунікацію з іншими користувачами.

Процес створення нового проекту починається в інтегрованому середовищі розробки PhpStorm[12], де ініціалізується Laravel-додаток, у межах якого реалізується архітектурний патерн MVC (Model–View–Controller).

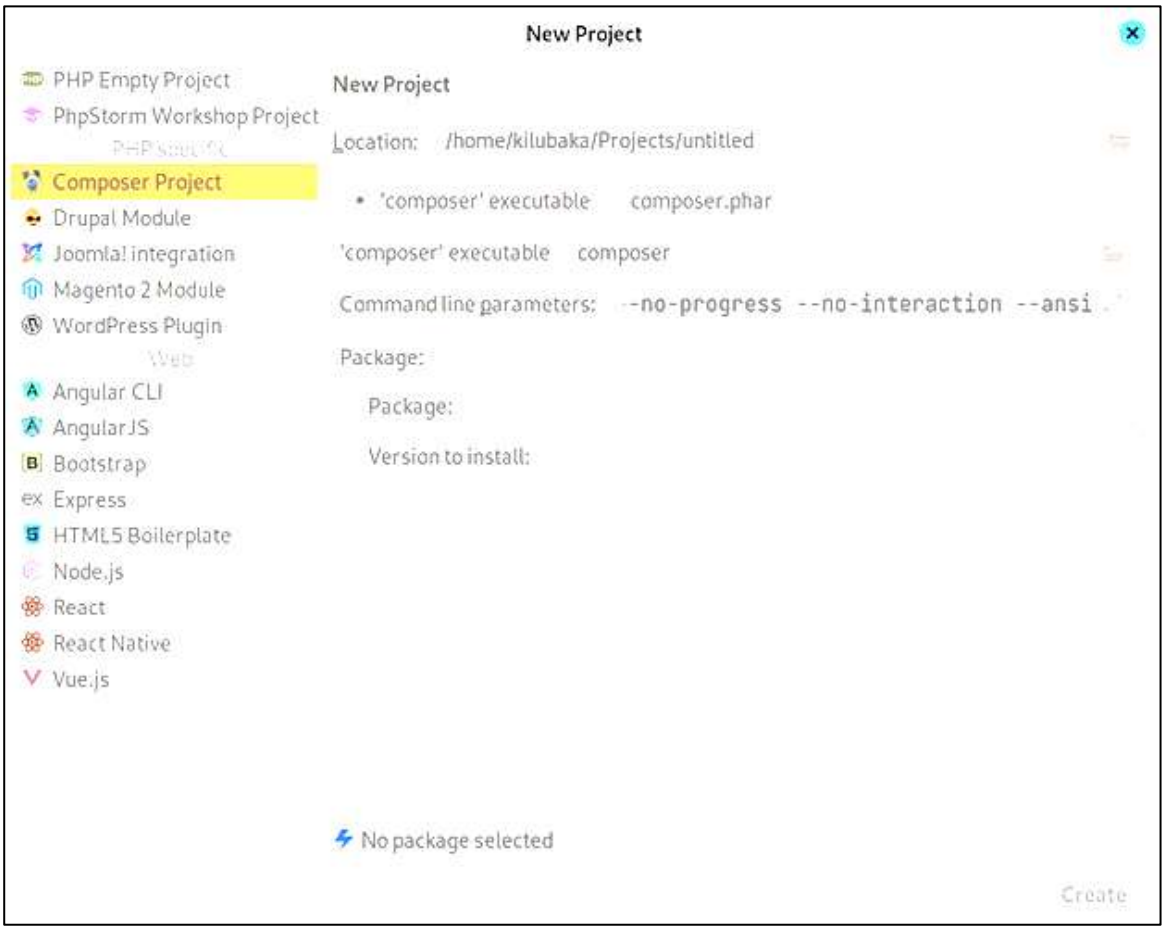


Рисунок 2.3.1 – Вигляд панелі для створення проекту

Застосування шаблону MVC дає змогу логічно розділити відповідальність між компонентами: модель відповідає за доступ до даних, вигляд – за відображення інформації, а контролер – за обробку запитів і координацію дій між компонентами.

Переваги використання MVC полягають у модульності, зменшенні складності коду, можливості повторного використання компонентів, а також підвищенні гнучкості та зручності супроводу системи.

У структурі шаблону:

– модель (Model) є центральним елементом, який керує даними, бізнес-логікою, валідацією та збереженням;
– вигляд (View) відповідає за відображення інформації, включаючи графіки, таблиці, текстові елементи тощо;
– контролер (Controller) перехоплює запити від користувача, формує необхідні дії та координує передачу даних між моделлю та виглядом.

Переглянути детальніше схему роботи патерну MVC можна на рисунку 2.3.2.

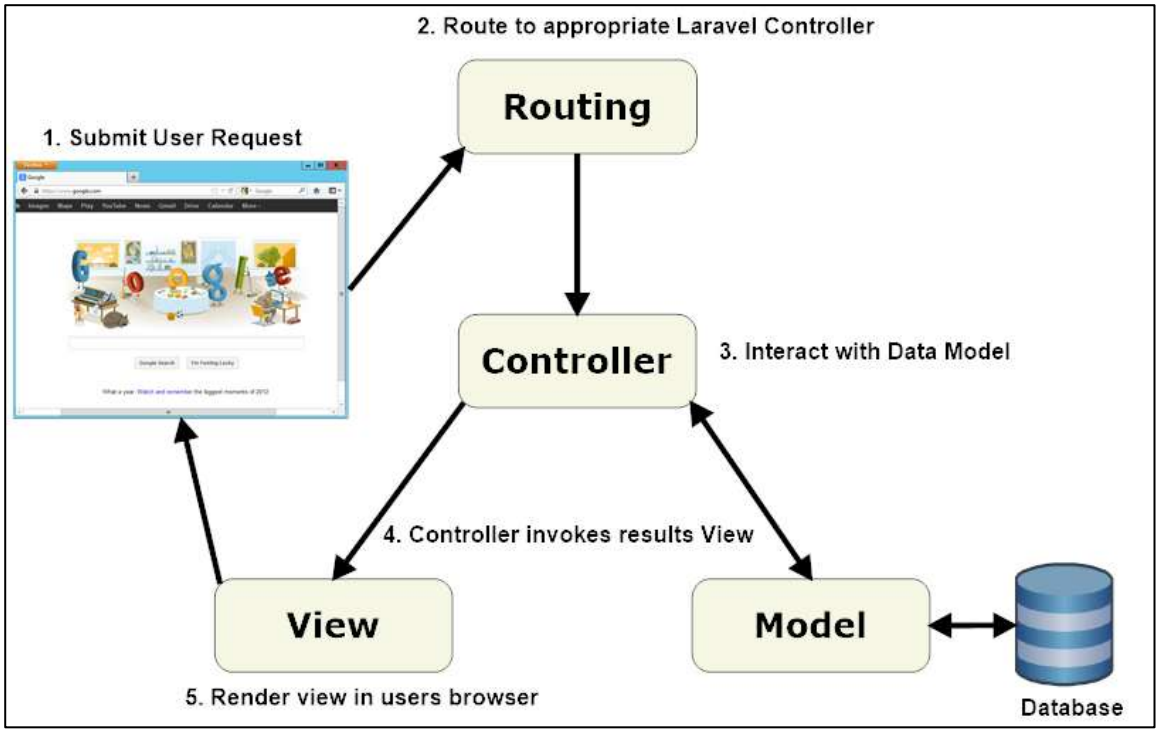


Рисунок 2.3.2 – Вигляд релізації патерну MVC

Для забезпечення продуктивності та зменшення навантаження на сервер критичною є коректна ідентифікація й типізація даних. Дані мають відповідати очікуваним форматам і типам.

Основним сховищем даних у системі є таблиці бази даних MySQL[11]. Крім того, застосовуються локальні змінні, масиви та словники, які використовуються для тимчасового зберігання і обробки структурованої інформації у форматі ключ–значення. Роботу з базою даних, міграціями, сідерами, моделями, репозиторіями та обсерверами виконував співавтор проекту – Пляцик Олександр.

Взаємодія між клієнтом та сервером реалізована через API-інтерфейси, зокрема RESTful API, що базується на архітектурі клієнт–сервер. API визначає правила для комунікації між застосунками, описує структуру запитів і відповідей, а також формат даних, зазвичай JSON. Основні складові REST API включають:

– endpoint – адреса виклику функціоналу;
– HTTP-методи – GET, POST, PUT, DELETE;
– заголовки запиту – мета-дані, включно з токенами безпеки;
– тіло запиту – передані дані або параметри.

Важливим аспектом у побудові будь-якого сучасного вебдодатку є організація взаємодії між клієнтською та серверною частинами. У даному проекті для цього використовується REST API – один із найпоширеніших стилів архітектури прикладного програмного інтерфейсу, призначений для побудови розподілених систем, у яких передача даних відбувається через протокол HTTP у стандартизованому форматі (найчастіше – JSON).

REST (Representational State Transfer) передбачає реалізацію низки принципів, що забезпечують масштабованість, простоту та ефективність у побудові серверно-клієнтських комунікацій. Основними з цих принципів є:

– відсутність стану (statelessness): кожен запит до сервера повинен містити всю необхідну інформацію для його обробки. Сервер не зберігає контексту або стану попередньої взаємодії з клієнтом;
– єдиний інтерфейс: взаємодія з ресурсами здійснюється уніфіковано – через HTTP-методи (GET, POST, PUT, DELETE) та URL-адреси;
– можливість кешування: відповіді сервера можуть бути помічені як кешовані, що дозволяє зменшити кількість запитів;
– клієнт-серверна архітектура: чітке розділення між інтерфейсом користувача (frontend) і логікою обробки даних (backend);
– шарова система: дані можуть проходити через послідовність проміжних шарів, наприклад, через проксі або кеш;
– можливість коду на вимогу (опціонально): клієнт може отримувати виконуваний код для розширення функціоналу (наприклад, JavaScript).

REST API реалізується за допомогою чітко структурованих HTTP-запитів, кожен з яких складається з чотирьох обов’язкових компонентів. Першим елементом є endpoint – це конкретна URL-адреса, на яку клієнт надсилає запит до сервера для доступу до певного ресурсу. Другим компонентом є HTTP-метод, який визначає дію, що має бути виконана: отримання даних (GET), створення нового ресурсу (POST), оновлення існуючого (PUT) або його видалення (DELETE). Третім елементом виступають HTTP-заголовки, які містять службову інформацію про запит, зокрема токени автентифікації, формат даних або мову відповіді. Завершує структуру запиту тіло повідомлення (body), у якому передаються самі дані, найчастіше у форматі JSON, необхідні для виконання дії.

Загальна модель REST API у розроблюваній системі включає три основні логічні елементи. Перший з них – клієнт (client), який представлений у вигляді фронтенд-додатку і є безпосереднім інтерфейсом взаємодії користувача із системою. Другий – сервер (server), що обробляє вхідні запити, виконує логіку бізнес-процесів, взаємодіє з базою даних та формує відповіді. Третім елементом є ресурс (resource) – це об'єкт або сутність, до якої надсилається запит, наприклад, список користувачів, календар подій або перелік проектів.У фреймворку Laravel реалізація REST API здійснюється за допомогою вбудованого механізму маршрутизації (routing), який забезпечує відповідність між HTTP-запитом та дією в контролері. Laravel дозволяє:

- визначати маршрути з підтримкою параметрів;
- групувати маршрути за спільною логікою або префіксами;
- обробляти домени та піддомени;
- застосовувати проміжне програмне забезпечення (middleware) для фільтрації запитів;
- працювати з методами GET, POST, PUT, DELETE для реалізації RESTful API.

На рисунку 2.3.3 подано узагальнену схему реалізації REST API та взаємодії між клієнтом, сервером і базою даних у контексті даного проекту.

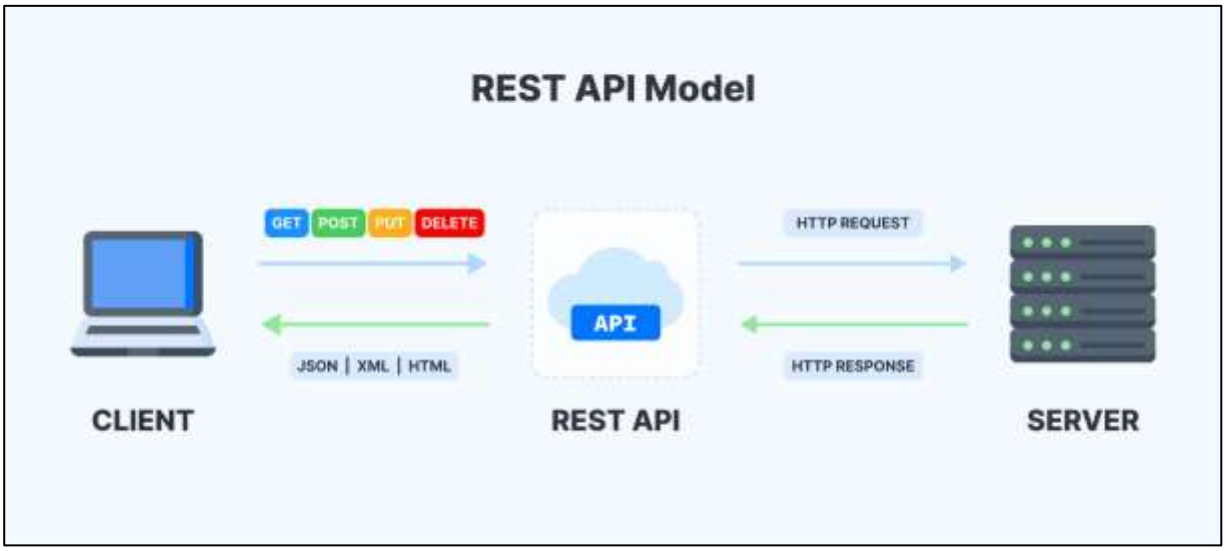


Рисунок 2.3.3 – Вигляд реалізації REST API

Головними об’єктами користувацького інтерфейсу у межах реалізованого програмного забезпечення є робоча область, яка складається з кількох ключових візуальних та функціональних елементів. Центральне місце в інтерфейсі займає головне вікно, в якому відображається зміст поточної дії або активного розділу. Ліворуч або зверху (залежно від розмітки) розміщується навігаційна панель керування, яка забезпечує швидкий доступ до основних модулів системи: персоналу, подій, проектів, зустрічей тощо.

Організація інтерфейсу передбачає використання вкладок, що дозволяє користувачеві швидко перемикатися між розділами без потреби перезавантаження сторінки. Це значно покращує взаємодію та пришвидшує виконання рутинних дій.

Для редагування або введення даних застосовуються стандартні HTML-елементи форм:

– текстові поля (input type="text") – для введення імен, назв, електронної пошти та інших коротких текстових значень;
– випадаючі списки (select, option) – для вибору зі списку доступних значень (наприклад, ролей або відділів);
– багаторядкові текстові площі (textarea) – для введення довших описів, нотаток або повідомлень.

Таке поєднання елементів забезпечує інтуїтивно зрозумілу логіку роботи з інтерфейсом, що особливо важливо для нових користувачів та осіб без технічного досвіду.

Загальне візуальне представлення базової структури інтерфейсу наведено нижче, рисунок 2.3.4.

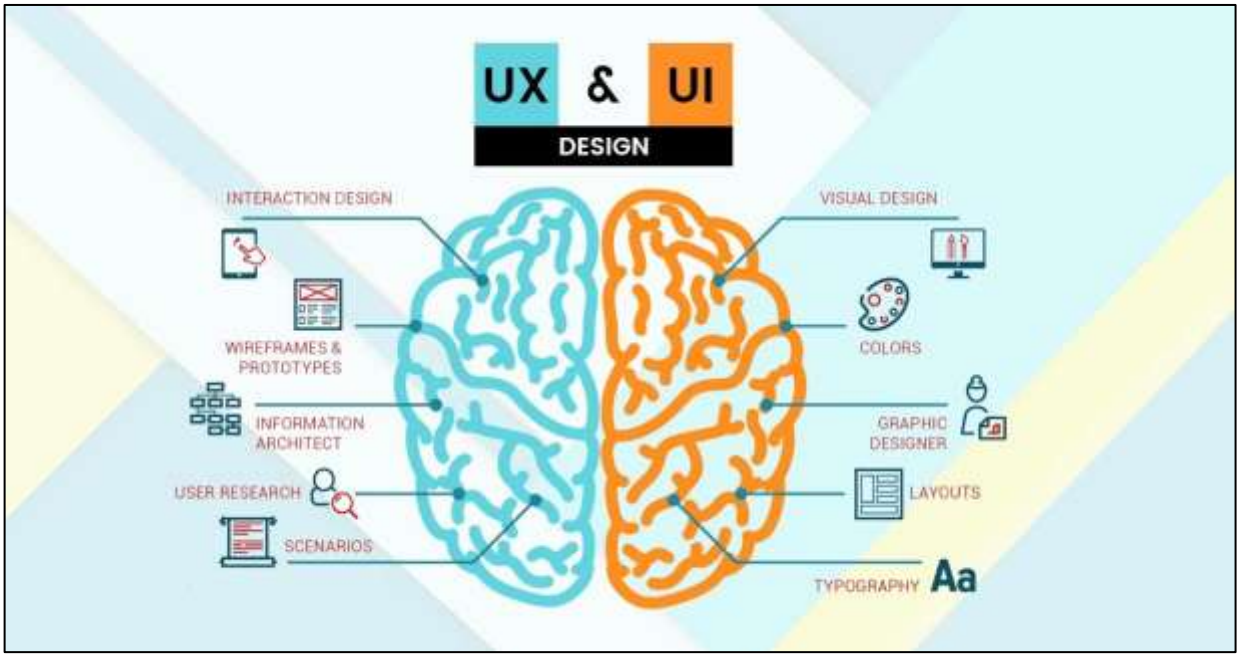


Рисунок 2.3.4 – Базові концепти дизайну

Інтерфейс вебдодатку, розробленого в межах даного проекту, реалізується у вигляді окремих вебсторінок, які об'єднують у собі компоненти користувацького інтерфейсу, стилі, динамічну поведінку та візуальну логіку взаємодії. Передача даних до сторінки виконується за посередництва контролера, що викликається через попередньо визначений маршрут (route). Контролер ініціює відповідну бізнес-логіку, отримує або обробляє дані з моделі, після чого передає їх у відповідне представлення або React-компоненту для подальшої візуалізації.

Приклад реалізації такого компонента наведено на рисунку нижче – це один із ключових блоків інтерфейсу, який відповідає за навігацію.

```jsx
import React from 'react'
import { useSelector } from 'react-redux'
import { Link } from 'react-router-dom'

import Svg from '../Svg'
import Image from '../User/Image'

import { routes } from '../../utils/routes'
import NavigationLink from './NavigationLink'

const Navigation = () => {
    const user = useSelector( selector: state => state.user.data)
    const navigationLinks = [...]

    return (
        <nav className="bg-primary w-[200px] text-white">
            <div className="logo h-16 border-b border-gray-300 px-2 mr-5">
                <Link to={routes.dashboard} className="flex flex-row">
                    <div className="block h-16 w-auto fill-current text-white py-2">
                        <Svg name="cram" className="p-1 bg-white rounded-lg"/>
                    </div>
                    <h1 className="text-xl leading-[4rem] px-2">
                        {'CRaM'}
                    </h1>
                </Link>
            </div>

            <Link to={routes.user} className="flex flex-col items-center mt-6 text-white mr-5">
                <Image className="object-cover w-24 h-24 mx-2 rounded-full" borderClassName="rounded-full" path={user.image}/>
                <h4 className="mx-2 mt-2">{user.name}</h4>
                <p className="mx-2 mt-1 text-sm">{user.email}</p>
            </Link>

            <div className="nav-menu py-6 mb-6 mr-5">
                {navigationLinks.map(( link : {…} , ind : number ) =>
                    <NavigationLink {...link} key={ind}/>
                )}
            </div>
        </nav>
    )
}

export default Navigation
```

Рисунок 2.3.5 – Приклад Navigation.jsx компоненти

Компоненти в React є основними будівельними блоками інтерфейсу. Їх можна використовувати багаторазово, вбудовувати один в одного, динамічно оновлювати вміст та передавати необхідні параметри через властивості (props). Також у компоненти можуть бути інтегровані анімації, спливаючі вікна, асинхронні запити до API або обробка дій користувача в режимі реального часу.

JavaScript-файл, який містить головну логіку застосунку, підключається централізовано один раз у базовому шаблоні, що унеможливлює повторне завантаження скриптів та знижує навантаження на систему. Аналогічним чином організується підключення стилів: скомпільований CSS-файл, створений на

основі TailwindCSS[13] і .less-препроцесора, додається один раз для всього проекту, що дозволяє уникнути дублювання коду стилізації.

У якості шаблонного рушія у Laravel використовується Blade[9], який дозволяє створювати сторінки з динамічним вмістом. Blade відрізняється простотою у використанні та повною сумісністю з чистим PHP-кодом. Файли шаблонів мають розширення .blade.php та зберігаються у директорії resources/views. Вони можуть викликатися як з маршрутів, так і з контролерів через функцію view().

Згідно з офіційною документацією Laravel[10], для передачі змінних до шаблону використовується другий аргумент функції view(), де у вигляді асоціативного масиву ["key" => "value"] передаються потрібні дані.

Для підтримки чистоти архітектури проекту рекомендовано створювати основний шаблон layout, від якого будуть наслідуватися всі інші сторінки. Це дозволить уникнути дублювання елементів, таких як заголовки, футери, навігація, а також забезпечить централізоване керування структурою сторінок.

Перевикористання компонент, у тому числі в Blade-шаблонах або через React, значно оптимізує кодову базу системи, спрощує підтримку й полегшує подальше розширення функціональності, що є критично важливим у масштабованих вебдодатках.

2.4 Висновки до другого розділу. Опис вхідної та вихідної інформації для користувачів

У межах створеної програмної системи для внутрішнього управління персоналом ключовим джерелом вхідної інформації виступають дані, які користувач самостійно вносить у систему за допомогою елементів інтерфейсу вебзастосунку. Первинним і основоположним об’єктом, до якого безпосередньо або опосередковано прив’язується інша інформація, є користувач – індивідуальна облікова одиниця, що уособлює працівника компанії. Уся архітектура логіки взаємозв’язків побудована навколо цієї сутності, оскільки саме з нею пов’язуються всі інші компоненти: події, робочі проекти, особливі дні, запити, а

також зустрічі, які формують основу щоденного функціонування організації. Користувач є відправною точкою для побудови інформаційної моделі, у межах якої реалізується збереження, обробка, відображення та адміністрування даних, пов’язаних із конкретною особою у структурі компанії.

Користувач є базовою одиницею системи, що містить персональну інформацію, яка включає прізвище, ім’я, по батькові, контактні дані, вік, сімейний стан, кваліфікацію, спеціалізацію, додаткові навички, а також роль у системі (звичайний працівник, адміністратор або супер-адміністратор). Всі інші дані в системі закріплюються за відповідним профілем користувача.

Іншою важливою сутністю є події. Це інформаційні елементи, що створюються адміністраторами і відображають важливі дати – державні свята, корпоративи, дні народження співробітників, ювілеї та інші події, які мають бути відображені у вигляді інтерактивного календаря. У кожен день може бути прив’язано кілька подій або жодної.

Проекти є окремим типом сутностей, що відображають поточну діяльність організації. Кожен проект може містити інформацію про назву, опис, статус, а також пов’язану команду учасників. Користувачі можуть бути закріплені за різними проектами з конкретними ролями – аналітик, розробник, тестувальник тощо. У майбутньому планується реалізація функції групових чатів у межах проектів.

Особливі дні – це нестандартні робочі стани працівника. До таких належать відпустка, лікарняний, робота з дому (ремоут), відрядження або неоплачувана відпустка. Користувач має змогу самостійно створювати запити на використання цих днів. Кількість доступних днів кожного типу можна буде конфігурувати адміністратором. Запити надсилаються відповідальній особі та підлягають погодженню.

Сутність зустрічей є елементом планування, що дозволяє створювати події типу обговорення, інтерв’ю, загальні збори чи консультації. Зустрічі можуть бути особистими (один-на-один) або груповими (для відділу чи всієї компанії).

Адміністратор формує запрошення, встановлює дату та час події, після чого інформація доступна користувачам у відповідному розділі.

Таким чином, основними вхідними даними є:

– особисті дані користувача, включно з його профілем, рольовою моделлю та контактною інформацією;
– параметри проектів, до яких користувачі залучені;
– конфігурації подій, створених адміністраторами;
– запити на особливі дні, ініційовані працівниками;
– записи про заплановані зустрічі, які адміністратор додає у систему.

Уся ця інформація є динамічною, може змінюватися, видалятися або доповнюватися в межах функціоналу, доступного через вебінтерфейс системи. Дані зберігаються у базі даних та передаються до клієнтської частини у вигляді відповідей API, що, у свою чергу, відображаються на сторінці за допомогою компонент інтерфейсу.

Результуюча або, інакше кажучи, вихідна інформація – це сукупність даних, яку користувач отримує у візуалізованому вигляді під час взаємодії з програмною системою. Вона формується як прямий наслідок обробки введених запитів, дій користувача та відповідних змін, що зберігаються у структурованих таблицях бази даних. У межах реалізованого програмного забезпечення така інформація репрезентується у вигляді цілісного інформаційного середовища, яке інтерактивно відображається на сторінках інтерфейсу та забезпечує прозору взаємодію між користувачем і функціональними модулями системи.

Наприклад, у розділі, присвяченому користувачам, відображаються детальні профілі всіх працівників компанії з вказівкою їхніх поточних проектів, залученості до запланованих зустрічей, а також з доступом до основної службової інформації. Переглянувши вкладку особливих днів, користувач має змогу ознайомитися з поданими запитами на тимчасову відсутність – зокрема, йдеться про вихідні дні, лікарняні або роботу в дистанційному режимі. У секції подій представлений календар з відміченими корпоративними заходами та державними святами, які адміністратор фіксує централізовано.

При переході до розділу проектів користувач отримує доступ до переліку активних та завершених ініціатив, із можливістю перегляду кількості учасників, розподілу ролей та ступеня залученості співробітників. У модулі зустрічей відображається розклад запланованих подій з розбивкою по місяцях, що дозволяє ефективно управляти часом та ресурсами.

Таким чином, звичайний користувач отримує персоналізований функціонал, обмежений межами власних прав доступу. Він може переглядати інформацію, подавати запити, брати участь у процесах, передбачених для його ролі. Адміністратор системи, у свою чергу, володіє повним набором прав на керування всіма сутностями, що існують у системі: йому доступне створення, редагування, підтвердження, відхилення або остаточне видалення будь-якого інформаційного об’єкта. Така модель дозволяє гнучко керувати обсягом прав залежно від рівня відповідальності конкретного користувача.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ЗАСТОСУНКУ

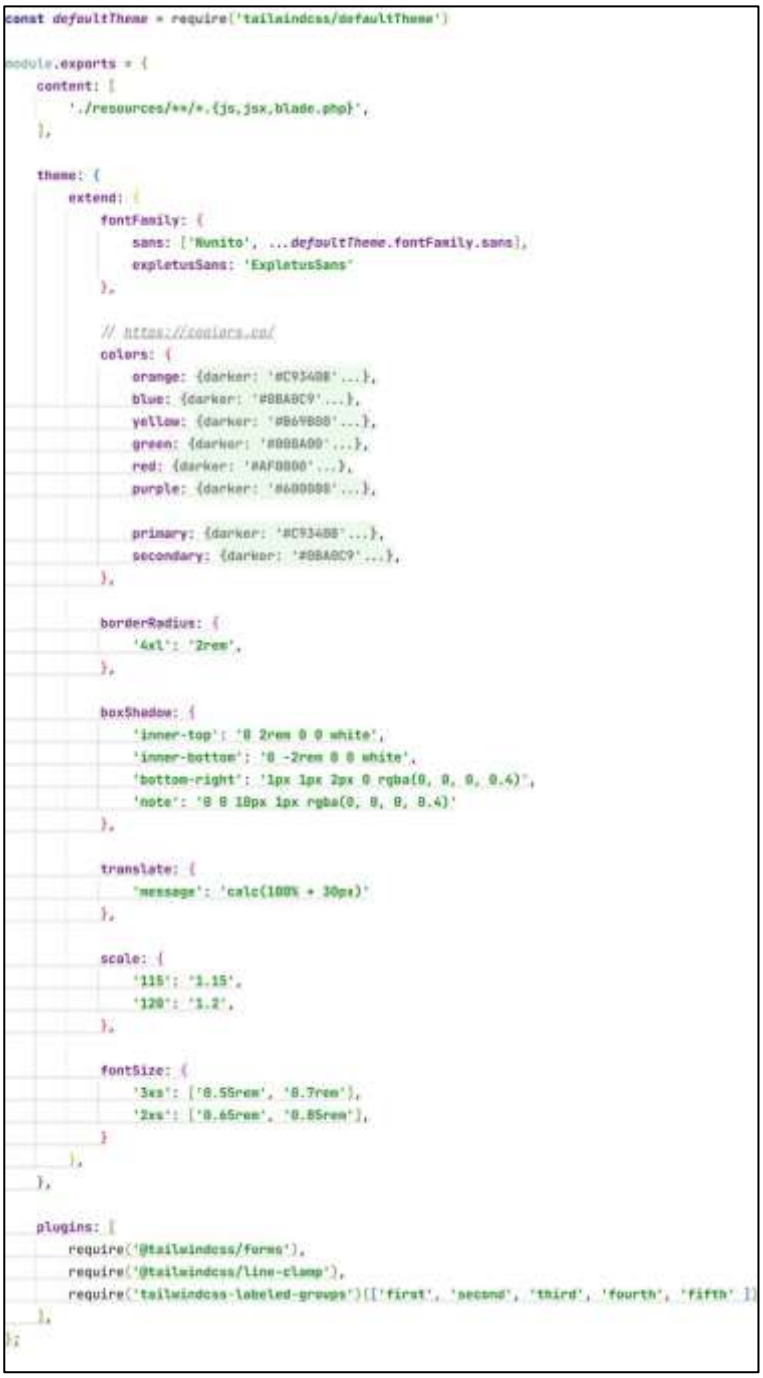
3.1 Розробка логіки проекту та кодування

Розробка програмного забезпечення передбачає не лише написання коду, але й попереднє налаштування інструментів, середовища та конфігураційних файлів, необхідних для забезпечення коректної роботи фронтенд- та бекенд-частин системи. У цьому контексті особлива увага приділяється підготовці та налаштуванню Webpack – збирача модулів, який відповідає за обробку ресурсів інтерфейсу.

```
const mix = require('laravel-mix')
const tailwindcss = require('tailwindcss')
const cssImport = require('postcss-import')
const autoprefixer = require('autoprefixer')

const ESLintPlugin = require('eslint-webpack-plugin')

mix.js( src: 'resources/index.js', output: 'public/js/')
   .react()
   .copy('resources/js/particles-config.json', 'public/js/')

   .less( src: 'resources/css/index.less', output: 'public/css')

   .options( options: {
       postCss: [
           tailwindcss( configOrPath: 'tailwind.config.js'),
           cssImport(),
           autoprefixer(),
       ],
       plugins: [
           new ESLintPlugin( options: { eslintPath: './.eslintrc.js' })
       ],
       module: {
           loaders: [{
               test: /\.json$/,
               loader: 'json-loader'
           }]
       }
   })
   .browserSync( config: {
       browser: ['google chrome'],
       proxy: 'https://cram.io',
       reloadDelay: 1000
   })
```

Рисунок 3.1.1 – Файл конфігуратора Webpack

На початковому етапі розробки було налаштовано Webpack-конфігурацію для автоматичної компіляції файлів стилів формату .less у формат .css, а також генерації HTML-розмітки з Blade-шаблонів (.blade.php). Крім того, було реалізовано механізм «гарячого» оновлення сторінки (hot reloading), який дозволяє динамічно перезавантажувати інтерфейс при збереженні змін, що значно пришвидшує процес розробки.



```
const defaultTheme = require('tailwindcss/defaultTheme')

module.exports = {
    content: [
        './resources/**/*.{js,jsx,blade.php}',
    ],

    theme: {
        extend: {
            fontFamily: {
                sans: ['Nunito', ...defaultTheme.fontFamily.sans],
                expletusSans: 'ExpletusSans'
            },

            // https://coolors.co/
            colors: {
                orange: {darker: '#C93408'...},
                blue: {darker: '#0BA0C9'...},
                yellow: {darker: '#B69B00'...},
                green: {darker: '#008A00'...},
                red: {darker: '#AF0000'...},
                purple: {darker: '#600088'...},

                primary: {darker: '#C93408'...},
                secondary: {darker: '#0BA0C9'...},
            },

            borderRadius: {
                '4xl': '2rem',
            },

            boxShadow: {
                'inner-top': '0 2rem 0 0 white',
                'inner-bottom': '0 -2rem 0 0 white',
                'bottom-right': '1px 1px 2px 0 rgba(0, 0, 0, 0.4)',
                'note': '0 0 10px 1px rgba(0, 0, 0, 0.4)'
            },

            translate: {
                'message': 'calc(100% + 30px)'
            },

            scale: {
                '115': '1.15',
                '120': '1.2',
            },

            fontSize: {
                '3xs': ['0.55rem', '0.7rem'],
                '2xs': ['0.65rem', '0.85rem'],
            }
        },
    },

    plugins: [
        require('@tailwindcss/forms'),
        require('@tailwindcss/line-clamp'),
        require('tailwindcss-labeled-groups')(['first', 'second', 'third', 'fourth', 'fifth' ]),
    ],
};
```

Рисунок 3.1.2 – Файл конфігуратора TailwindCSS

Наступним кроком стало конфігурування TailwindCSS – CSS-фреймворку, який генерує стилі безпосередньо на основі класів, використаних у JSX-компонентах. У файлі конфігурації Tailwind було визначено шляхи до файлів, що підлягають відстеженню, а також розширено базову палітру кольорів за рахунок кастомізованих відтінків. Це забезпечило можливість швидкого візуального налаштування інтерфейсу згідно з вимогами дизайну.

```
/**
 * Auth types
 * @type {string}
 */
export const USER_GUEST = 'USER_GUEST'
export const USER_LOGGED = 'USER_LOGGED'

export const USER_TYPES = {
    USER_GUEST,
    USER_LOGGED,
}

/**
 * User roles
 * @type {string}
 */
export const ROLE_USER = 'ROLE_USER'
export const ROLE_ADMIN = 'ROLE_ADMIN'
export const ROLE_SUPER = 'ROLE_SUPER'

export const USER_ROLES = {
    ROLE_USER,
    ROLE_ADMIN,
    ROLE_SUPER
}

/**
 * window.localStorage && window.sessionStorage && window.cookies
 *
 * @type {string}
 */
export const LOCAL_STORAGE_USER_TOKEN = 'LOCAL_STORAGE_USER_TOKEN'
export const SESSION_STORAGE_USER_TOKEN = 'SESSION_STORAGE_USER_TOKEN'
export const COOKIE_USER_TOKEN = 'COOKIE_USER_TOKEN'

/**
 * Color picker palette
 */
export const COLORS = ['#fff', '#f28b82', '#fbbc04', '#fff475', '#ccff90', '#a7ffeb', '#aecbfa', '#d7aefb']
```

Рисунок 3.1.3 – Файл констант

Окрім базової кольорової гами, у конфігурацію були додані додаткові параметри стилів, зокрема індивідуальні значення радіуса межі (border-radius), масштабування (scale), розміру шрифту, а також визначено фірмовий шрифт ExpletusSans. Було підключено додаткові плагіни TailwindCSS, що розширюють функціональність фреймворку та дозволяють реалізовувати специфічні ефекти або адаптивну поведінку окремих елементів.

Для централізації управління логікою станів застосунку було створено окремий файл constants.js, у якому оголошуються глобальні константи. Серед них – прапорець авторизованого стану користувача, поточна роль (наприклад, admin або user), ключі збереження даних у локальному сховищі браузера, сховищі сесій і cookie. Тут же визначено набір стандартних кольорів, що використовуються у візуальному оформленні інтерфейсу. Таке структурування спрощує подальшу підтримку системи та забезпечує її масштабованість.

Звісно ж, потрібно налаштувати пакетний менеджер NPM. Дуже багато розробників в процесі створення програм часто створюють так звані ‘велосипеди’ – шматки коду або функціоналу який повторюється і який не доцільно реалізовувати кожного разу. Саме для таких випадків використовуюють пакетні менеджери, для економії часу розробки і покращення кінцевої якості продукту.

Окремим важливим етапом підготовки середовища розробки є налаштування пакетного менеджера NPM (Node Package Manager), який виконує роль інструмента для керування залежностями JavaScript-проекту. Його використання дозволяє уникати дублювання коду, автоматизувати процеси встановлення бібліотек та інструментів, а також значно зменшити час на реалізацію рутинного або повторюваного функціоналу.

У процесі розробки програмного забезпечення досить часто виникає спокуса реалізувати власноруч ті чи інші елементи логіки, які вже багаторазово створювались іншими розробниками – так звані "велосипеди". Йдеться про повторне написання типових функцій, компонентів або модулів, які мають вже готові, перевірені та протестовані альтернативи у вигляді пакетів з відкритим

вихідним кодом. Саме в таких ситуаціях доцільно використовувати NPM як стандартний інструмент для встановлення й оновлення сторонніх бібліотек.

```
{
    "name": "cram",
    "version": "1.0.0",
    "private": true,
    "license": "MIT",
    "scripts": {
        "dev": "npm run development",
        "development": "mix",
        "hot": "mix watch --hot",
        "prod": "npm run production",
        "production": "mix --production",
        "watch": "mix watch",
        "watch-poll": "mix watch -- --watch-options-poll=1000"
    },
    "eslintConfig": {
        "extends": [
            "react-app",
            "react-app/jest"
        ]
    },
    "dependencies": {
        "@reduxjs/toolkit": "^1.8.1",
        "axios": "^0.21",
        "html-react-parser": "^1.4.12",
        "i18next": "^21.6.16",
        "masonry-layout": "^4.2.2",
        "parse5": "^7.0.0",
        "particles.js": "^2.0.0",
        "react-i18next": "^11.16.7",
        "react-toastify": "^9.0.3",
        "redux-saga": "^1.1.3"
    },
    "devDependencies": {"@babel/core": "^7.17.10"...},
    "authors": [
        {
            "name": "Oleksandr Pliatsyk",
            "email": "plsaniok55@gmail.com"
        },
        {
            "name": "Ilia Kravchuk",
            "email": "kilubaka@gmail.com"
        }
    ]
}
```

Рисунок 3.1.4 – Файл для NPM - package.json

Завдяки NPM у проекті було інтегровано низку критично важливих залежностей, зокрема React, Axios, TailwindCSS, Webpack, ESLint, Redux Toolkit, Moment.js та інші бібліотеки, які дозволяють реалізувати інтерфейс, обробку запитів, керування станом програми, валідацію форм та взаємодію з датами.

Крім того, NPM надає змогу створити власні скрипти для автоматизації завдань, зокрема запуску компіляції, тестування, збірки або запуску локального сервера. Всі ці команди централізовано зберігаються у файлі package.json, що є ядром конфігурації NPM-проекту. Його структура дозволяє підтримувати узгодженість версій залежностей та спрощує перенесення або спільну розробку програмного продукту в команді.

Як наслідок було використано низку ключових програмних пакетів, що забезпечують реалізацію клієнтської частини, маршрутизацію, форматування, обробку запитів та стилізацію інтерфейсу. Базовим елементом фронтенду виступає бібліотека React.js, яка є основою для побудови всіх інтерфейсних компонентів додатку. Для керування глобальним станом програми та зберігання даних між різними частинами інтерфейсу було впроваджено Redux – ефективне сховище станів, що дозволяє централізовано управляти даними.

Комунікація між клієнтською частиною та сервером реалізована за допомогою бібліотеки Axios, яка забезпечує зручну роботу з HTTP-запитами різних типів, зокрема GET, POST та DELETE. Для забезпечення єдиного стилю коду та автоматичної перевірки на синтаксичну і стилістичну відповідність використовується ESLint – інструмент аналізу та форматування JavaScript-коду. Для компіляції стилів, написаних мовою Less, у формат CSS використано однойменний пакет less, який дозволяє застосовувати змінні, вкладеність та інші елементи метамови при створенні стилів.

У роботі з масивами, об’єктами та іншими структурами даних активно застосовується бібліотека Lodash, яка надає зручні утиліти для маніпуляцій з даними. Обробка дат і часових форматів здійснюється за допомогою бібліотеки Moment, яка дозволяє ефективно працювати з календарними значеннями, форматувати їх та обчислювати різниці між подіями. Щоб забезпечити

кросбраузерну підтримку стилів, у проекті використовується PostCSS – інструмент для автоматичної генерації закінчень властивостей стилів для різних браузерів.

Маршрутизація між сторінками реалізована за допомогою React Router – спеціалізованого набору компонентів, який дозволяє налаштовувати навігацію в односторінковому застосунку (SPA). Для стилізації всіх інтерфейсних елементів застосовується фреймворк TailwindCSS, який дозволяє будувати дизайн без написання кастомного CSS завдяки великій кількості готових утилітарних класів. Усі компоненти проекту, залежності та збірка ресурсів об'єднуються за допомогою Webpack – сучасного пакетного компоновщика, який автоматизує трансформацію файлів, їх оптимізацію та вивід у фінальний вигляд.

Таким чином, перелік використаних пакетів охоплює усі ключові потреби сучасного вебзастосунку, забезпечуючи зручність розробки, читабельність коду, адаптивність інтерфейсу та стабільну інтеграцію з серверною частиною.

Наступним етапом у реалізації логіки застосунку стало налаштування системи маршрутизації. У межах Laravel маршрутизація (routing) виконує функцію посередника між URL-запитом і відповідним методом у контролері, який обробляє цей запит. У зв’язку з наявністю значної кількості сутностей у системі, було прийнято рішення уникати централізації маршрутів в одному файлі, натомість винести маршрути для кожного окремого модуля в самостійні конфігураційні файли. Це забезпечує структурованість, зручність в обслуговуванні та кращу масштабованість системи.

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

use Illuminate\Support\Facades\Route;
use Modules\User\Http\Controllers\UserController;

Route::resource('user', UserController::class);
```

Рисунок 3.1.5 – Створення Routes для сутності User

На рисунку 3.5 продемонстровано приклад створення маршруту для сутності користувача (User). У рамках цієї реалізації використовується директива Route::resource(), яка дозволяє автоматично створити набір RESTful-маршрутів для стандартних CRUD-операцій (створення, читання, оновлення, видалення). В результаті однієї інструкції генерується повний набір маршрутів для взаємодії з відповідним контролером. Список усіх сформованих шляхів можна переглянути, виконавши php artisan route:list, приклад виводу наведено на рисунку 3.1.6.

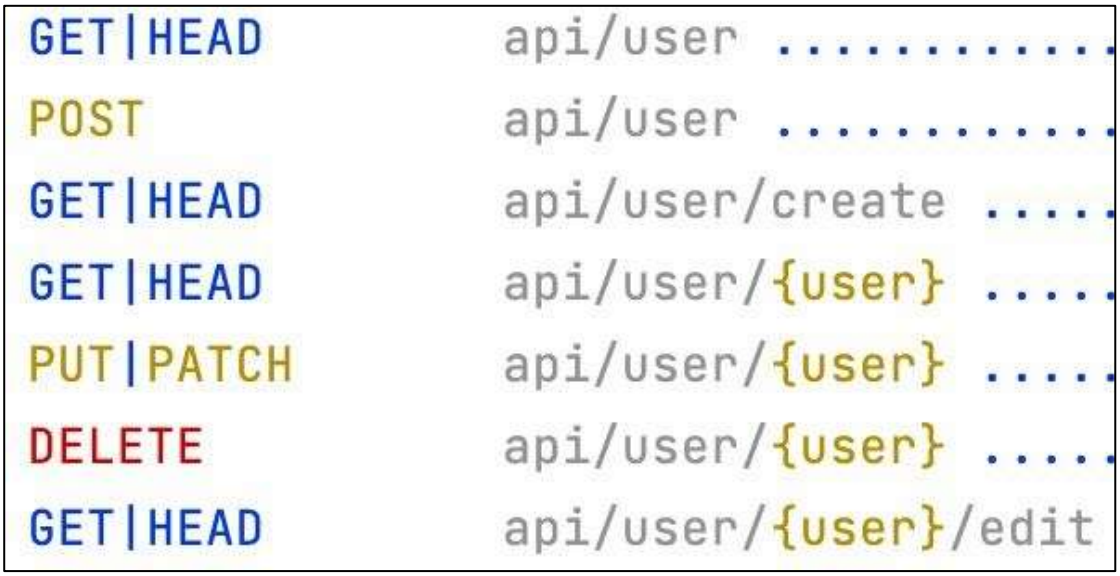


Рисунок 3.1.6 – Routes для сутності User

```php
<?php

use App\Http\Controllers\SpecialDays\SpecialDaysController;
use Illuminate\Support\Facades\Route;

Route::post('/special-days/{id}/edit', [SpecialDaysController::class, 'update'])
    ->middleware('admin')
    ->name('special-days.edit');

Route::get('/special-days/{id}/delete', [SpecialDaysController::class, 'delete'])
    ->middleware('admin')
    ->name('special-days.delete');

Route::post('/special-days/create', [SpecialDaysController::class, 'store'])
    ->middleware('auth')
    ->name('special-days.create');

Route::get('/special-days/{month?}/{year?}', [SpecialDaysController::class, 'index'])
    ->middleware('auth')
    ->name('special-days.index');
```

Рисунок 3.1.7 – Файл маршрутизатора special-days.php

Крім базової конфігурації, маршрутизатор дозволяє працювати з параметрами – як обов'язковими, так і опціональними. Для кожного маршруту задається контролер, відповідальний за обробку запиту, а також метод, що реалізує необхідну логіку – повернення представлення, оновлення даних у базі або зміна вигляду інтерфейсу. На рисунку 3.1.7 зображено приклад маршрутизатора для сутності особливих днів (special-days.php), який демонструє гнучкість у роботі з параметризованими маршрутами.

```
export const routes = {
    root: '/',
    dashboard: '/dashboard',

    logIn: '/login',
    logOut: '/logout',

    user: '/user',
    userById: '/user/:id',
    userCreate: '/user/create',
    userEdit: '/user/:id/edit',
    userList: '/user/list',

    specialDay: '/special-day',
    specialDayMonthYear: '/special-day/date/:month/:year',
    specialDayById: '/special-day/:id',
    specialDayCreate: '/special-day/create',
    specialDayEdit: '/special-day/:id/edit',
    specialDayList: '/special-day/list',

    event: '/event',
    eventMonthYear: '/event/date/:month/:year',
    eventById: '/event/:id',
    eventCreate: '/event/create',
    eventEdit: '/event/:id/edit',
    eventList: '/event/list',
```

Рисунок 3.1.8.а – Клієнтські роути

```
    project: '/project',
    projectById: '/project/:id',
    projectCreate: '/project/create',
    projectEdit: '/project/:id/edit',
    projectList: '/project/list',
    projectsByUserId: '/project/user/:id',

    meet: '/meet',
    meetMonthYear: '/meet/date/:month/:year',
    meetById: '/meet/:id',
    meetCreate: '/meet/create',
    meetEdit: '/meet/:id/edit',
    meetList: '/meet/list',
    meetsByUserId: '/meet/user/:id',

    note: '/note',
    noteById: '/note/:id',
    notesByUserId: '/note/user/:id',
    notesAllDimensions: '/note/all/dimensions',

    // EAVs from eav_entity table
    eavSpecialDay: '/eav/special-day',
    eavMeet: '/eav/meet',
    eavEvent: '/eav/event',
    eavNotification: '/eav/notification',
    eavTeam: '/eav/team',
    eavUserStanding: '/eav/user-standing',
    eavFamilyStatus: '/eav/family-status',
    eavProject: '/eav/project',
    eavTag: '/eav/tag',

    notFound: '*'
}
```

Рисунок 3.1.8.б – Продовження зображення клієнтські роути

З метою забезпечення зручності клієнтської розробки усі запити до API було винесено в окремий файл, у якому централізовано зберігаються адреси всіх серверних маршрутів. Це дозволяє уникнути жорсткого кодування (hardcoding) у компонентах інтерфейсу та полегшує модифікацію структури запитів у майбутньому. Приклад такої реалізації наведено на рисунках 3.1.8 та 3.1.9.

```
import {
    USER_GUEST,
    USER_LOGGED
} from './constants'
import { routes } from './routes'

import { default as User } from '../pages/User'
import { default as UserRegister } from '../pages/User/Register'
import { default as UserEdit } from '../pages/User/Edit'
import { default as UserList } from '../pages/User/List'

export const pagesRoutes = [
    {
        name: 'User',
        path: routes.user,
        type: [USER_LOGGED],
        PageComponent: User,
    },
    {
        name: 'UserById',
        path: routes.userById,
        type: [USER_LOGGED],
        PageComponent: User,
    },
    {
        name: 'UserCreate',
        path: routes.userCreate,
        type: [USER_LOGGED],
        PageComponent: UserRegister,
    },
    {
        name: 'UserEdit',
        path: routes.userEdit,
        type: [USER_LOGGED],
        PageComponent: UserEdit,
    },
    {
        name: 'UserList',
        path: routes.userList,
        type: [USER_LOGGED],
        PageComponent: UserList,
    },
```

Рисунок 3.1.9 – pagesRoutes.js

Після завершення налаштування серверної частини маршрутизації наступним кроком стало створення централізованого файлу сторінкової маршрутизації на фронтенді. У цьому файлі були оголошені всі доступні сторінки, включаючи сторінку авторизації, інтерфейси для всіх сутностей, а також спеціальні сторінки на випадок помилок, зокрема 404. Такий підхід дозволяє ефективно керувати навігацією у межах SPA-архітектури, не створюючи зайвих залежностей.

Отже, налаштована маршрутизація забезпечує надійний зв’язок між користувачем, інтерфейсом і логікою обробки даних, а також дозволяє масштабувати застосунок без необхідності значної модифікації існуючої структури.

Після формування всієї системи маршрутів можна переходити безпосередньо до створення інтерфейсних шаблонів сторінок та їх функціональних елементів. Візуальна частина клієнтської логіки реалізується у вигляді компонентів, написаних на JavaScript із використанням синтаксису JSX. Всі файли клієнтської частини розміщено в окремій директорії resources, яка є основною точкою розміщення JSX-шаблонів у проекті, побудованому на базі React.js.

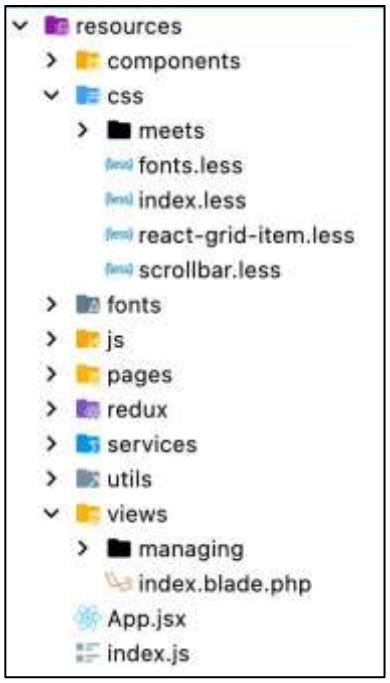


Рисунок 3.1.10 – Тека resources

У межах цієї директорії слід виокремити дві важливі підструктури: папки components та components/Layouts. Папка components містить універсальні, дрібномасштабні елементи інтерфейсу, які можуть бути використані багаторазово у різних шаблонах. Це, зокрема, кнопки, поля вводу, модальні вікна, елементи списків тощо. Використання таких компонент дозволяє уникнути повторення коду (принцип DRY – Don’t Repeat Yourself), підвищити узгодженість дизайну та спростити подальшу підтримку коду.

Папка components/Layouts, на відміну від попередньої, містить базові каркаси сторінок, які виконують роль шаблонів вищого рівня. Вони містять основні секції інтерфейсу: шапку сайту (header), бічну навігацію (sidebar), контентну частину (main content) та нижній колонтитул (footer). На рисунках 3.1.11 та 3.1.12 представлено приклади таких макетів, які наслідуються іншими сторінками при формуванні остаточної структури інтерфейсу.

```jsx
import React from 'react'
import PropTypes from 'prop-types'

import Particles from '../Particles'
import Svg from '../Svg'

const Guest = ({ children }) => {
    return (
        <main className="bg-primary font-mono">
            <Particles/>
            <div className="font-sans text-gray-900 antialiased relative">
                <div className="min-h-screen flex flex-col sm:justify-center items-center pt-6 sm:pt-0">
                    <Svg name="cram" className="w-48 h-32 px-2 cursor-pointer bg-white rounded-lg"/>
                    <div className="w-full sm:max-w-md mt-6 px-6 py-4 bg-white shadow-md overflow-hidden sm:rounded-lg">
                        {children}
                    </div>
                </div>
            </div>
        </main>
    )
}

Guest.propTypes = {
    children: PropTypes.oneOfType( types: [
        PropTypes.arrayOf(PropTypes.node),
        PropTypes.node,
    ]).isRequired,
}

export default Guest
```

Рисунок 3.1.11 – Layout файл Guest.jsx

```jsx
import ...

const App = ({ children }) => {
    return (
        <main className="bg-primary flex p-5 pr-3 min-h-screen font-mono overflow-x-hidden overflow-y-scroll">
            <Navigation/>

            <div className="bg-white w-full rounded-4xl px-5 flex flex-col">
                {children}
            </div>
        </main>
    )
}

App.propTypes = {
    children: PropTypes.oneOfType( types: [
        PropTypes.arrayOf(PropTypes.node),
        PropTypes.node,
    ])
}

export default App
```

Рисунок 3.1.12 – Layout файл App.jsx

У кожному окремому JSX-файлі шаблону дотримано стандартної структури, яка забезпечує порядок і передбачуваність під час розробки. На початку файлу здійснюється підключення необхідних бібліотек, зовнішніх модулів або локальних компонентів, що використовуватимуться у межах поточного шаблону. Далі оголошується функція (або функціональний компонент), що містить розмітку сторінки, оформлену з використанням класових стилів TailwindCSS – утилітарного фреймворку для швидкої адаптивної стилізації.

Наступним етапом є конфігурування параметрів компонента через об’єкт propTypes, який визначає типи, обов’язковість і структуру властивостей, що передаються у компонент під час його використання. Завершується файл інструкцією export default, яка надає змогу імпортувати компонент в інші частини програми для повторного використання.

Такий підхід до побудови інтерфейсу дозволяє організувати ефективну компонентну структуру, що легко масштабується, повторно використовується та відповідає принципам сучасної frontend-архітектури.

Точкою входу до клієнтської логіки є базовий файл index.blade.php, що розташовується у директорії представлень (resources/views). Цей файл слугує

головним контейнером для підключення React-застосунку до DOM-дерева сторінки. На рисунку 3.1.13 наведено вигляд цього файлу.

```
<!DOCTYPE html>
<html lang="{{ str_replace('_', '-', app()->getLocale()) }}">
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <meta name="csrf-token" content="{{ csrf_token() }}">
    <title>{{ config('app.name', 'Laravel') }}</title>
</head>
<body class="font-sans">
    <noscript>{{ __('You need to enable JavaScript to run this app.') }}</noscript>
    <div id="app"></div>
    <script src="{{ mix('/js/index.js') }}"></script>
</body>
</html>
```

Рисунок 3.1.13 – Головний .blade файл

Саме тут відбувається ініціалізація бібліотеки React шляхом підключення основного JavaScript-файлу index.js, який, у свою чергу, виконує роль координатора, що зв'язує браузерне середовище з логікою застосунку. Структуру цього файлу зображено на рисунку 3.1.14. index.js імпортує кореневий компонент, ініціалізує контексти та передає управління далі у застосунок.

```
import React from 'react'
import ReactDOM from 'react-dom/client'

import App from './App'

const root = ReactDOM.createRoot(document.getElementById( elementId: 'app'))

root.render(
    <App/>
)
```

Рисунок 3.1.14 – Головний index.js файл

Наступним рівнем завантаження є компонент App.js, який слугує центральним хабом для підключення ключових підсистем клієнтської частини. Тут реалізовано конфігурацію маршрутизації з використанням бібліотеки react-

router, налаштовано підтримку роботи з cookie, а також підключено систему сповіщень. У цьому ж файлі підключаються стилі, згенеровані за допомогою Webpack, а також користувацькі шрифти. Компонент App.js відіграє роль логічного ядра застосунку, з якого формується загальна структура інтерфейсу.

```
import React from 'react'
import { Provider } from 'react-redux'
import { CookiesProvider } from 'react-cookie'
import { applyMiddleware, compose, createStore } from 'redux'
import createSagaMiddleware from 'redux-saga'

import rootReducers from './redux/reducers'

import Router from './components/Router'

import { ToastContainer } from 'react-toastify'
import 'react-toastify/dist/ReactToastify.css'
import '../public/css/index.css'
import './utils/i18n'


import rootSaga from './redux/sagas/rootSaga'

const sagaMiddleware = createSagaMiddleware()

const composeEnhancers = window.__REDUX_DEVTOOLS_EXTENSION_COMPOSE__ || compose
const store = createStore(rootReducers,
    composeEnhancers(applyMiddleware(sagaMiddleware))
)
sagaMiddleware.run(rootSaga)

const App = () => {
    return (
        <CookiesProvider>
            <Provider store={store}>
                <Router />
                <ToastContainer />
            </Provider>
        </CookiesProvider>
    )
}

export default App
```

Рисунок 3.1.15 – Компонента App.js

Щодо стилізації, використовується файл app.less, який виступає основним джерелом стилів для застосунку. TailwindCSS дозволяє значно зменшити обсяг коду завдяки своїй утилітарній природі: стилі формуються безпосередньо у JSX-розмітці, тож файл app.less переважно містить лише імпорт шрифтів, базові кольори або додаткові налаштування, які не можна задати в inline-режимі. Це сприяє мінімізації розміру фінального CSS-файлу та покращенню продуктивності сторінок.

```
@tailwind base;
@tailwind components;
@tailwind utilities;


@import "./meets/ticket.less";
@import "./fonts.less";
@import "./scrollbar.less";

@import "../../node_modules/react-grid-layout/css/styles.css";
@import "../../node_modules/react-resizable/css/styles.css";
@import "./react-grid-item.less";
```

Рисунок 3.1.16 – Головний файл стилів app.less

Окрему роль у клієнтській архітектурі відіграє файл api.js, який реалізує обгортку для бібліотеки Axios. Він спрощує взаємодію з API-сервером, дозволяючи централізовано обробляти всі вихідні запити. У випадку, коли користувач авторизований, до кожного запиту автоматично додається відповідний токен. Крім того, файл реалізує механізм обробки відповідей, включаючи виведення сповіщень у інтерфейсі. Такий підхід забезпечує єдину точку контролю за мережею, дозволяє обробляти помилки уніфіковано та спрощує масштабування клієнтського коду.

```
import axios from 'axios'
import { toast } from 'react-toastify'

import { COOKIE_USER_TOKEN } from '../utils/constants'

let cookies = {}
document.cookie.split( separator: '; ')
        .map(elem => {
            const [key, value] = elem.split( separator: '=')
            cookies[key] = value
        })

const api = axios.create({
    baseURL: '/api/'
})

const messageHandler = (message) => {
    if (message) {
        toast[message.status](message.text)
    }
}

api.interceptors.request.use( onFulfilled: config => {
    config.headers['Authorization'] = `Bearer ${cookies[COOKIE_USER_TOKEN]}` ?? ''
    return config
}, onRejected: error => {
    console.error('api.interceptors.request:', error)
    return Promise.reject(error)
})

api.interceptors.response.use( onFulfilled: response => {
    const data = response?.data
    messageHandler(data?.message)
    return data
}, onRejected: error => {
    if (error.response.status == 401) {
        document.cookie = `${COOKIE_USER_TOKEN}=; expires=Thu, 01 Jan 1970 00:00:00 GMT;`
    } else {
        console.error('api.interceptors.response:', error)
    }
    messageHandler(error.response.data?.message ?? {status: 'error', text: error.response.statusText})
    return Promise.reject(error)
})

export default api
```

Рисунок 3.1.17 – Axios-обгортка api.js

Уся решта функціональних компонентів та сторінок розміщена у відповідних теках, логічно розбитих за принципом сутностей – аналогічно до модульної структури бекенд-частини. Кожна сутність має свій набір сторінок (наприклад, create, edit, index) та допоміжних компонентів, які відповідають за відображення, взаємодію, валідацію даних і бізнес-логіку. Така структуризація забезпечує високу модульність, логічну цілісність і спрощує командну роботу над проектом.

3.2 Порядок тестування розробки

Тестування програмного забезпечення є важливим етапом життєвого циклу розробки, що належить до процесів контролю якості. Воно охоплює низку діяльностей, спрямованих на забезпечення відповідності програмного продукту встановленим вимогам. Цей процес включає планування тестових робіт, розробку сценаріїв тестування, безпосереднє виконання тестів та аналіз результатів їх проходження. Тестування дозволяє здійснити верифікацію та валідацію функціональних і нефункціональних властивостей програмного забезпечення з метою виявлення розбіжностей між очікуваною і фактичною поведінкою системи.

Одиницею виміру у процесі тестування є тестовий випадок – це формалізована конструкція, що містить конкретні вхідні дані, передумови, очікувану поведінку системи та критерії оцінки результатів. Кожен тестовий випадок призначений для перевірки одного або кількох аспектів функціональності програмного продукту та є найменшою самостійною одиницею перевірки.

Залежно від глибини охоплення та об'єкта тестування, розрізняють декілька рівнів тестування. Першим є компонентне, або модульне тестування, що передбачає ізольовану перевірку окремих функціональних одиниць – методів, класів або компонентів. Таке тестування здебільшого проводиться в середовищі розробки за допомогою спеціалізованих фреймворків для модульного тестування. Зазвичай дефекти, виявлені на цьому рівні, усуваються оперативно. Один із сучасних ефективних підходів до модульного тестування – це принцип «розробки через тестування» (test-driven development), за якого тести створюються ще до написання відповідного функціонального коду. В процесі реалізації розробник поступово додає частини функціональності, перевіряючи їх проходження вже заздалегідь підготовленими тестами, що дозволяє мінімізувати помилки на ранніх етапах.

Наступним рівнем є інтеграційне тестування, що спрямоване на перевірку взаємодії між окремими компонентами системи. Його метою є виявлення помилок у точках інтеграції та забезпечення стабільності обміну даними між модулями, а

також з операційною системою, зовнішнім обладнанням або сторонніми службами. Інтеграційне тестування може здійснюватися за кількома підходами. Підхід «знизу вгору» передбачає початкове тестування найнижчих рівнів системи з поступовим підключенням вищих рівнів. Такий підхід особливо ефективний у разі, коли більшість нижчого рівня вже реалізовано. Альтернативний підхід «зверху вниз» базується на тестуванні загальної логіки високорівневих компонентів із використанням симуляцій (заглушок) замість ще не реалізованих підлеглих модулів. У міру готовності ці модулі замінюються справжніми компонентами. Третій варіант – підхід «великий вибух», за якого майже всі модулі одразу інтегруються в одне ціле для спільного тестування. Хоча такий підхід дозволяє скоротити час на підготовку, він створює значні ризики у разі неякісної підготовки тестових сценаріїв.

Заключним етапом є системне та приймальне тестування. Системне тестування перевіряє роботу всієї системи в цілому, тоді як приймальне тестування проводиться на підставі сценаріїв, сформованих відповідно до вимог замовника. Цей етап проводиться після завершення основного циклу розробки і передує переданню системи до впровадження або продуктивного середовища. Рішення про початок приймального тестування приймається у разі досягнення системою заданого рівня функціональної та якісної готовності. Замовник має бути ознайомлений із планом приймальних робіт, де вказано перелік тестових сценаріїв, відповідальних осіб, дати проведення перевірки та критерії прийняття рішення щодо затвердження або повернення програмного продукту на доопрацювання.

Фаза приймального тестування триває до моменту ухвалення остаточного рішення: чи передається система в експлуатацію, чи потребує доопрацювання з боку команди розробників.

Таким чином, усі рівні тестування – від модульного до приймального – формують послідовну систему перевірок, яка забезпечує високу якість програмного забезпечення та знижує ризики функціональних збоїв після впровадження.

3.3 Тестування готової вебплатформи

Фінальним етапом циклу розробки програмного забезпечення є повномасштабне тестування готової системи, що має на меті оцінити її стабільність, відповідність вимогам і функціональність у реальних умовах експлуатації. Проведення тестування є критично важливим для виявлення помилок, які могли залишитися поза увагою під час розробки, а також для перевірки, наскільки платформа відповідає технічному завданню, функціональним специфікаціям та очікуванням кінцевих користувачів.

Процес тестування охоплює як окремі модулі, так і систему в цілому. На початкових етапах здійснюється перевірка ізольованих частин – компонентів, функцій, контролерів, після чого поступово переходять до інтеграційного тестування взаємодії між модулями, і, зрештою, – до комплексної перевірки функціонування платформи як єдиного цілого. Завершальною фазою виступає тестування загальної поведінки системи у взаємодії з обчислювальним середовищем, що включає операційну систему, мережеву інфраструктуру, браузери та інші зовнішні сервіси.

Основна мета цього етапу полягає не лише у виявленні технічних недоліків, а й у перевірці доцільності, зручності та ефективності роботи системи в контексті бізнес-процесів. Оскільки кількість можливих варіантів введення, сценаріїв використання та комбінацій даних потенційно нескінченна, розробляється обґрунтована стратегія тестування. Вона передбачає реалізацію вибіркових тестових сценаріїв, які забезпечують максимальне покриття функціональності у межах наявних ресурсів та обмежень часу.

Критеріями оцінки при тестуванні можуть виступати: відповідність системи початковим вимогам, правильність обробки всіх типів вхідних даних, здатність виконувати визначені функції у заданий проміжок часу, зручність у користуванні, а також сумісність з різними операційними системами та версіями браузерів.

У межах тестування вебплатформи було застосовано два основних підходи – модульне та функціональне тестування. Модульне тестування реалізується на

етапі програмування для верифікації окремих частин коду. Воно дає змогу перевірити правильність логіки, охопити всі критичні маршрути виконання, протестувати граничні значення та перевірити внутрішню структуру даних. Такий підхід дозволяє оперативно усувати помилки ще до інтеграції модуля в загальну систему.

Функціональне тестування, у свою чергу, зосереджується на перевірці відповідності поведінки системи її функціональним вимогам. У фокусі знаходиться взаємодія інтерфейсу з користувачем, правильність обробки поданих запитів, видача очікуваних результатів, забезпечення цілісності та збереження даних. Особливістю цього типу тестування є те, що воно здійснюється за принципом так званого «чорного ящика» – без заглиблення у внутрішню логіку реалізації, виключно на основі вхідних та вихідних даних. Такий підхід дозволяє максимально наблизити умови перевірки до реального використання системи кінцевим користувачем.

Крім основного функціоналу, в процесі тестування оцінюються також продуктивність, надійність, адаптивність інтерфейсу до різних типів пристроїв, зокрема мобільних, а також стабільність роботи в умовах перевантаження. За результатами тестування формується звіт із виявленими помилками, пропозиціями щодо поліпшення та рекомендаціями для подальшої оптимізації.

Отже, всебічне тестування платформи на завершальному етапі забезпечує впевненість у її технічній стабільності, функціональній відповідності та готовності до впровадження в умовах реального використання.

Для підтвердження працездатності основних функцій системи було проведено демонстраційне функціональне тестування, результати якого зафіксовано на відповідних ілюстраціях. Метою даного етапу є візуалізація ключових сценаріїв взаємодії користувача з системою, а також перевірка коректності обробки стандартних і граничних ситуацій.

На рисунках 3.3.1 – 3.3.5 зображено можливі сценарії функціонального тестування вебдодатку.

Рисунок 3.3.1 – Початкова сторінка

На рисунку 3.3.1 представлено початкову сторінку вебдодатку, яка з’являється при переході користувача на головний домен системи. У межах цього сценарію було протестовано процедуру авторизації – один із базових механізмів, що забезпечує доступ до персоналізованого інтерфейсу. У разі правильного введення облікових даних система успішно ідентифікує користувача та перенаправляє його до основного робочого середовища. Одночасно з цим відображається динамічне повідомлення про успішний вхід, що підтверджує виконання відповідного функціонального блоку.

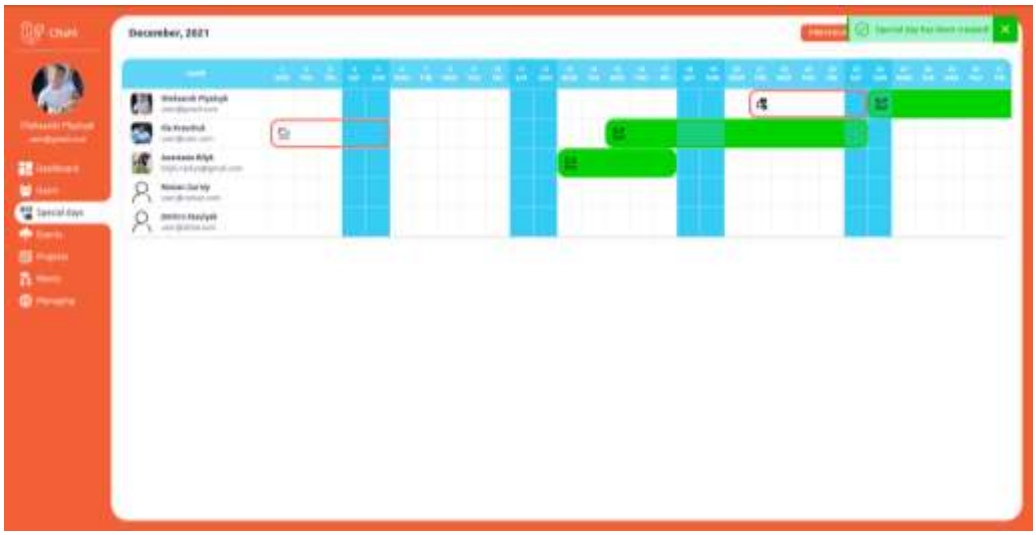


Рисунок 3.3.2 – Сторінка особливих днів

На наступних етапах було перевірено роботу модуля «особливих днів». Цей функціонал надає користувачу можливість створювати запити на відпустку, лікарняний або інші виняткові дати. На рисунку 3.3.2 демонструється сценарій успішного створення відрізку часу, відміченого як особливий день. Система реагує на запит динамічним сповіщенням, що підтверджує позитивний результат операції.

Однак не всі сценарії можуть завершуватися успішно. На рисунку 3.3.3 зафіксовано ситуацію, коли користувач намагається створити особливий день на ділянці календаря, що вже зайнята іншим активним записом. У цьому випадку система блокує дію, відображаючи попередження про неможливість збереження нової заявки. Така поведінка свідчить про наявність валідації вхідних даних на стороні сервера або клієнта.

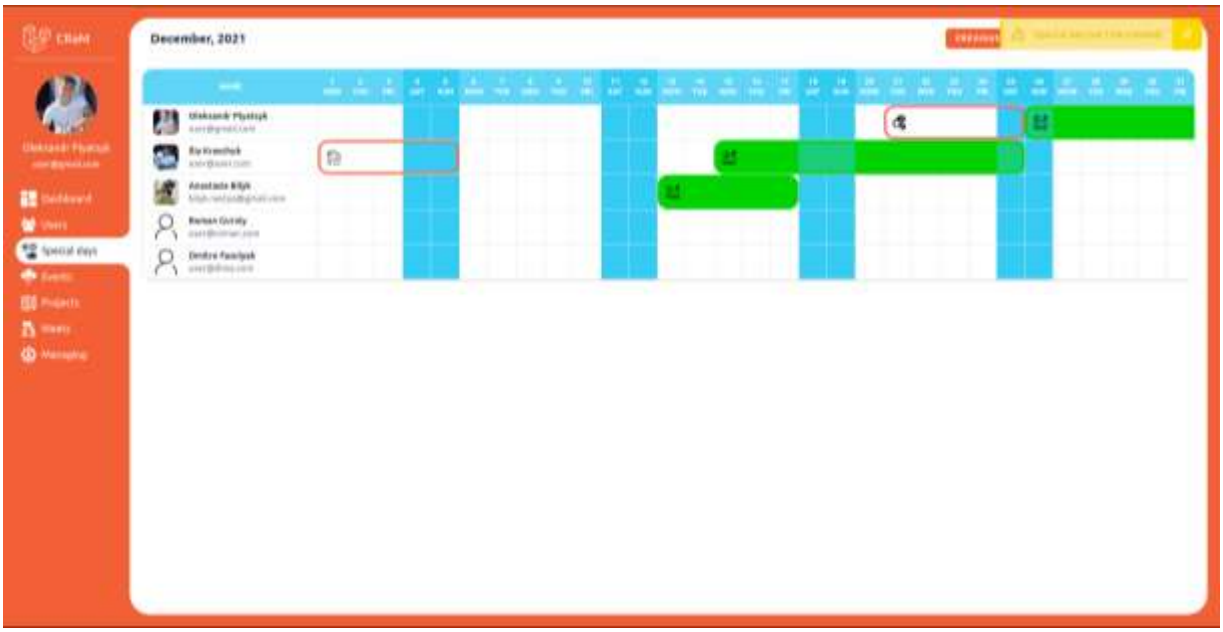


Рисунок 3.3.3 – Сторінка особливих днів

На подальших етапах було протестовано механізми редагування та видалення особливих днів. Рисунок 3.3.4 ілюструє сценарій керування цими сутностями. У випадку успішного видалення відповідного запису система повідомляє користувача про виконання дії, підтверджуючи, що відповідний обробник маршруту працює коректно, а зміни успішно синхронізуються з базою даних.

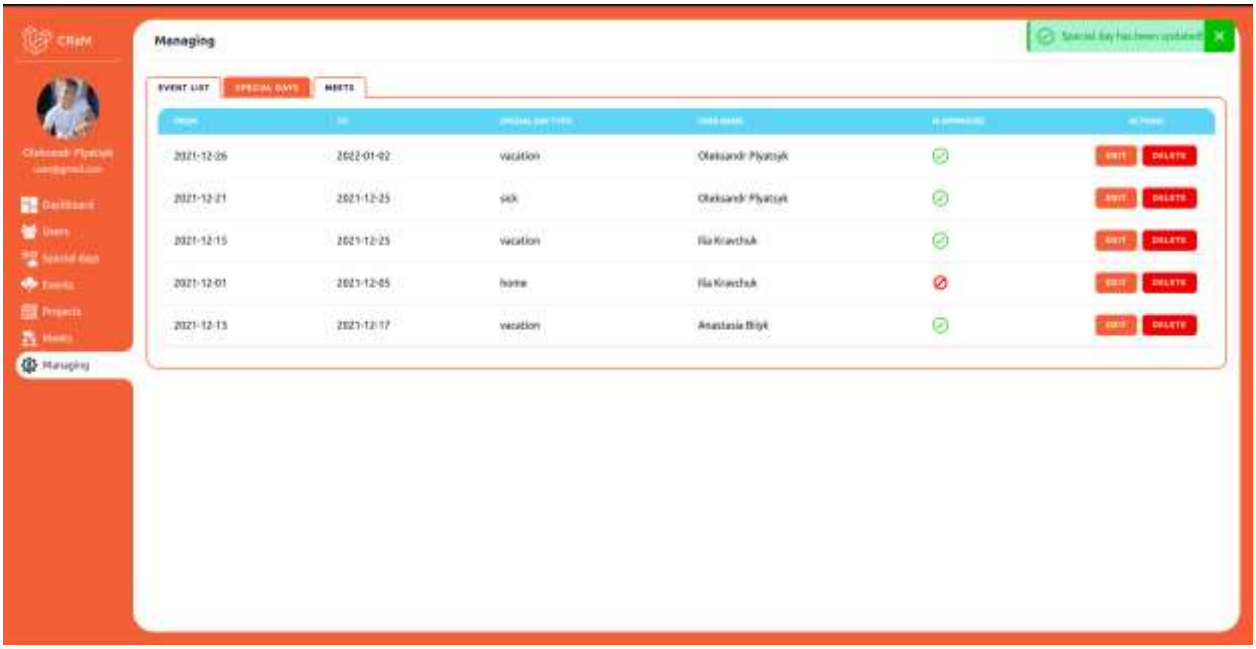


Рисунок 3.3.4 – Сторінка особливих днів

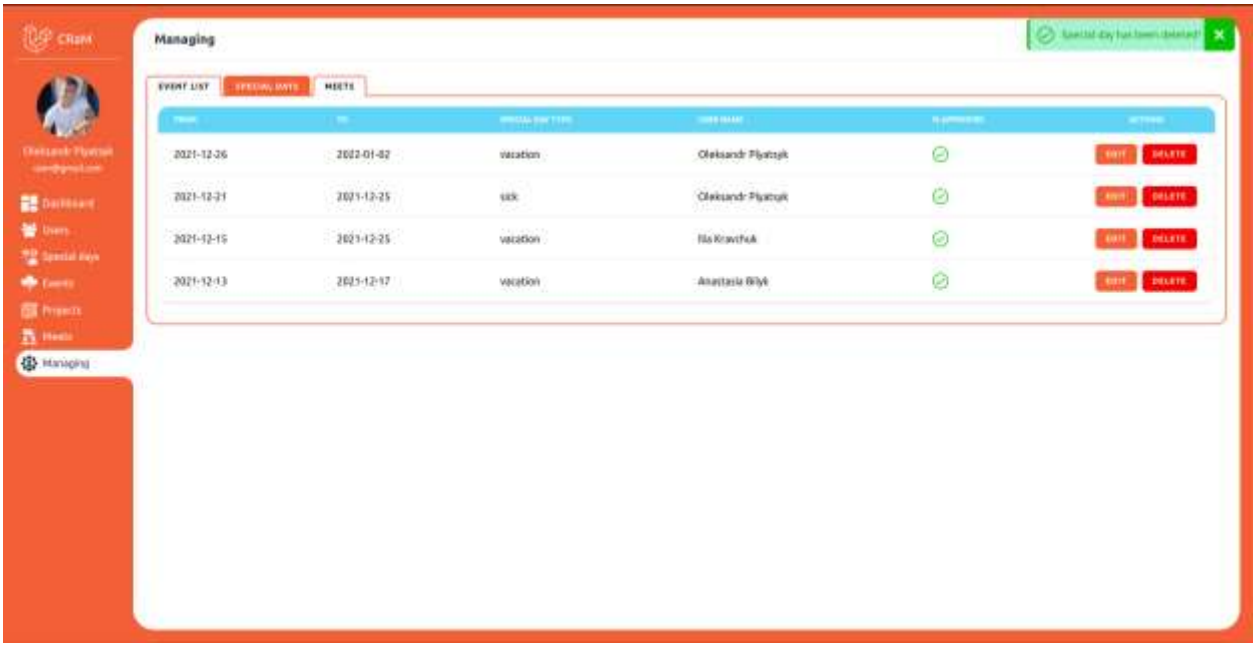


Рисунок 3.3.5 – Сторінка особливих днів

Завершальним етапом функціонального тестування стала перевірка операції видалення відрізку часу у модулі керування особливими днями. Як видно з рисунка 3.3.5, після здійснення відповідної дії користувач отримує динамічне сповіщення, яке підтверджує успішне виконання запиту. Це є свідченням того, що механізм обробки запитів на видалення функціонує коректно і забезпечує відповідну зворотну реакцію інтерфейсу.

На основі проведених перевірок можна зробити висновок, що всі ключові функції вебплатформи працюють відповідно до технічного завдання. Верифіковано коректність роботи окремих модулів, перевірено їхню взаємодію, протестовано усі основні сценарії користувацької поведінки та шляхи проходження логіки системи. Ретельно досліджено різні варіанти вхідних даних, що дозволило оцінити стійкість та адаптивність системи до різних ситуацій, зокрема – нештатних.

У цьому розділі представлено прикладну модель роботи платформи, що ілюструє логіку опрацювання запитів, трансформацію інформації та типові сценарії взаємодії користувача з системою. Усі перевірки здійснювались відповідно до моделі, сформованої ще на етапі проектування. Для структурованого опису архітектури та функціональності системи було розроблено такі діаграми:

- діаграма варіантів використання, яка ілюструє типові сценарії взаємодії користувача з системою;
- діаграма класів, що відображає структуру об’єктів і зв’язки між ними;
- діаграма компонентів, яка демонструє модульну побудову програмного забезпечення;
- діаграма станів, що описує можливі переходи між станами об’єктів залежно від дій користувача або подій системи;
- діаграма розгортання, яка показує топологію системи та взаємозв’язки між фізичними вузлами.

В процесі тестування було забезпечено охоплення усіх доступних методів, функціональних маршрутів та прикордонних умов. Такий підхід дозволяє впевнено стверджувати, що вебдодаток відповідає вимогам як у частині обробки даних, так і у забезпеченні зручності користування.

Саме детальне попереднє моделювання дозволило ще до початку безпосередньої реалізації знизити ризики архітектурних помилок, чітко визначити залежності між компонентами та оптимізувати подальший процес тестування.

Таким чином, результати тестування підтверджують технічну готовність програмного продукту до практичного застосування та свідчать про його відповідність сучасним вимогам до якості веборієнтованих інформаційних систем.

3.4 Встановлення та розгортання застосунку

Розгортання вебзастосунку «Customer Relationships and Management» (CRaM) здійснюється у декілька послідовних етапів і не потребує складної конфігурації, що дозволяє швидко почати його використання в будь-якому середовищі. Доступ до інтерфейсу системи надається за посиланням https://cram.io/, за умови попереднього запуску контейнерів Docker у локальному середовищі, оскільки вебдодаток спроектовано з урахуванням локального розгортання серверної частини.

Платформа є кросплатформною і може бути розгорнута на більшості сучасних операційних систем, включаючи Windows, Linux, macOS, а також мобільні операційні системи Android та iOS. Такий підхід дозволяє адаптувати застосунок до різних середовищ експлуатації без додаткових змін у кодовій базі або налаштуваннях.

Після успішного встановлення необхідних залежностей та запуску контейнерів Docker (у складі яких розміщені сервер Laravel, база даних MySQL та інші допоміжні сервіси), система стає доступною для використання через браузер. Відкриття початкової сторінки засвідчує коректність конфігурації, що підтверджується на рисунку 3.4.1.

Рисунок 3.4.1 – Вигляд початкової сторінки CRaM

Архітектурна схема розгортання системи на фізичних або віртуальних вузлах наочно відображена на відповідній діаграмі розгортання, що додається у додатку до пояснювальної записки. Ця діаграма демонструє топологію взаємодії між основними компонентами системи – клієнтською частиною, серверною логікою, базою даних та допоміжними службами.

Таким чином, процес встановлення та запуску CRaM є технічно прозорим, достатньо гнучким і не потребує специфічних знань від користувача, що робить платформу придатною для швидкої інтеграції в інфраструктуру підприємства будь-якого масштабу.

3.5 Інструкція з використання розробленого проекту

Для початку роботи з інформаційною системою «Customer Relationships and Management» (CRaM) користувачеві необхідно перейти за вказаним URL-адресом платформи. Першим кроком є авторизація в системі – доступ до функціоналу надається лише зареєстрованим користувачам. Самостійна реєстрація нових облікових записів не передбачена: створення облікового запису здійснюється виключно адміністраторами. Такий підхід дозволяє запобігти несанкціонованому

доступу сторонніх осіб, що підвищує загальний рівень інформаційної безпеки системи.

У разі введення некоректних облікових даних (електронної пошти або пароля), система виконує валідацію запиту та генерує відповідне повідомлення про помилку. Якщо ж авторизація відбувається успішно, користувача автоматично перенаправляє на головну сторінку – Dashboard, де зосереджено основну інформацію про його активність та ключові показники.

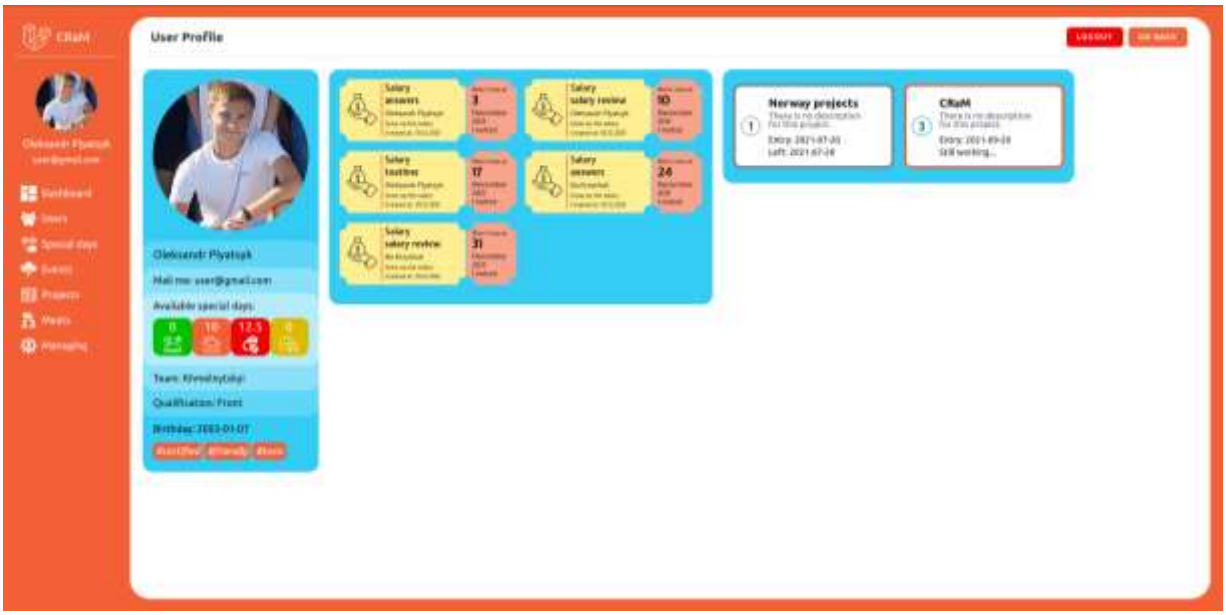


Рисунок 3.5.1 – Сторінка Dashboard

На інформаційній панелі користувач має змогу переглянути узагальнену статистику щодо своєї активності, включаючи кількість використаних особливих днів, заплановані зустрічі, а також події, що стосуються співробітників, підлеглих або всієї організації загалом. Крім цього, сторінка дозволяє швидко переходити до інших функціональних модулів системи.

У межах окремого розділу – сторінки «Special Days» – користувач отримує доступ до інформації щодо особливих днів. Це можуть бути запити на відпустку, роботу з дому, лікарняні або інші виняткові ситуації, що потребують погодження з керівництвом. Система забезпечує зручне візуальне представлення таких днів у вигляді таблиці або календаря. Користувач може створити новий запит, перевірити його статус або переглянути історію раніше поданих заявок.

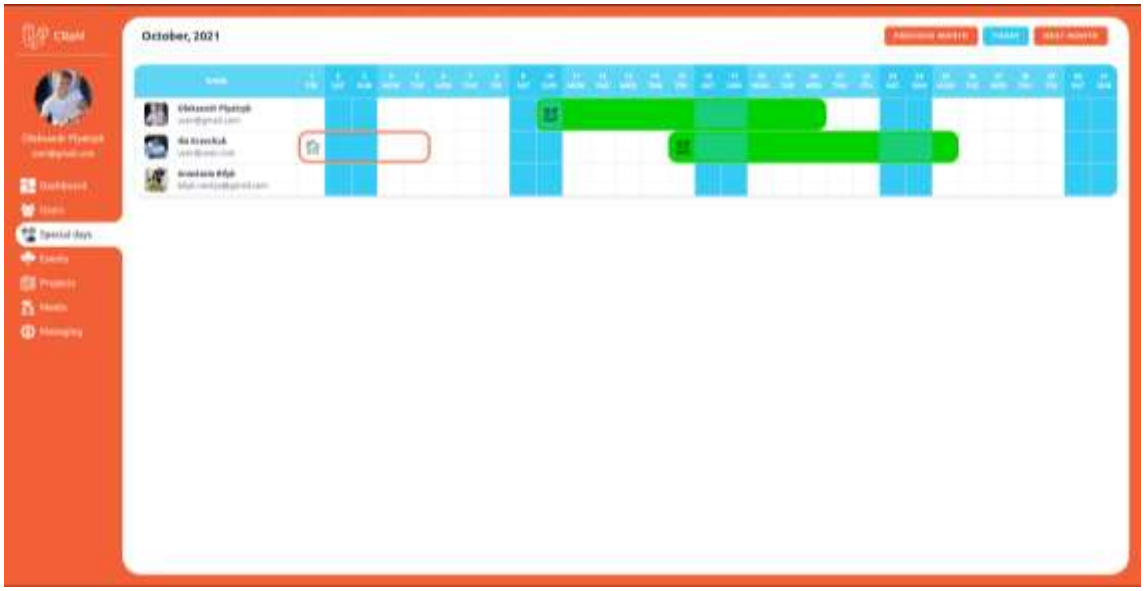


Рисунок 3.5.2 – Сторінка Special Days

Окрім цього, користувач має можливість переглядати інформацію про актуальні проекти, до яких він залучений. На відповідній сторінці відображаються назви проектів, кількість учасників, терміни реалізації, а також короткий опис цілей та призначення. Така функціональність дає змогу працівнику тримати фокус на своїх обов'язках у межах поточних задач та взаємодіяти з командою ефективно.

Інтерфейс системи побудовано з урахуванням принципів інтуїтивної навігації, тому навіть користувачі без технічної підготовки можуть швидко орієнтуватися в структурі платформи. Завдяки централізованому доступу до ключових функцій вебдодаток забезпечує повноцінну взаємодію між учасниками організаційного процесу в межах єдиного середовища.

Експлуатація системи CRaM є простою, інтуїтивно зрозумілою та не вимагає спеціальних навичок. Інтерфейс адаптовано таким чином, щоб користувач будь-якого рівня технічної підготовки міг швидко орієнтуватися в основному функціоналі та ефективно виконувати свої повсякденні завдання.

Для реалізації контрольованого доступу до функцій системи впроваджено механізм ACL (Access Control List) – список контролю доступу, що забезпечує диференційоване надання прав для різних ролей користувачів. Це дозволяє керівникам компаній гнучко налаштовувати рівні доступу до даних та функціоналу відповідно до посадових обов'язків співробітників.

Звичайний користувач системи має обмежений інтерфейс. Він може переглядати лише ті дані, що безпосередньо стосуються його облікового запису або до яких йому надано доступ відповідно до політики безпеки. Зокрема, редагування, видалення або модифікація об’єктів, що не належать до сфери його відповідальності, заборонені. Це забезпечує високий рівень захисту внутрішніх процесів і виключає ризик випадкових змін або несанкціонованих втручань.

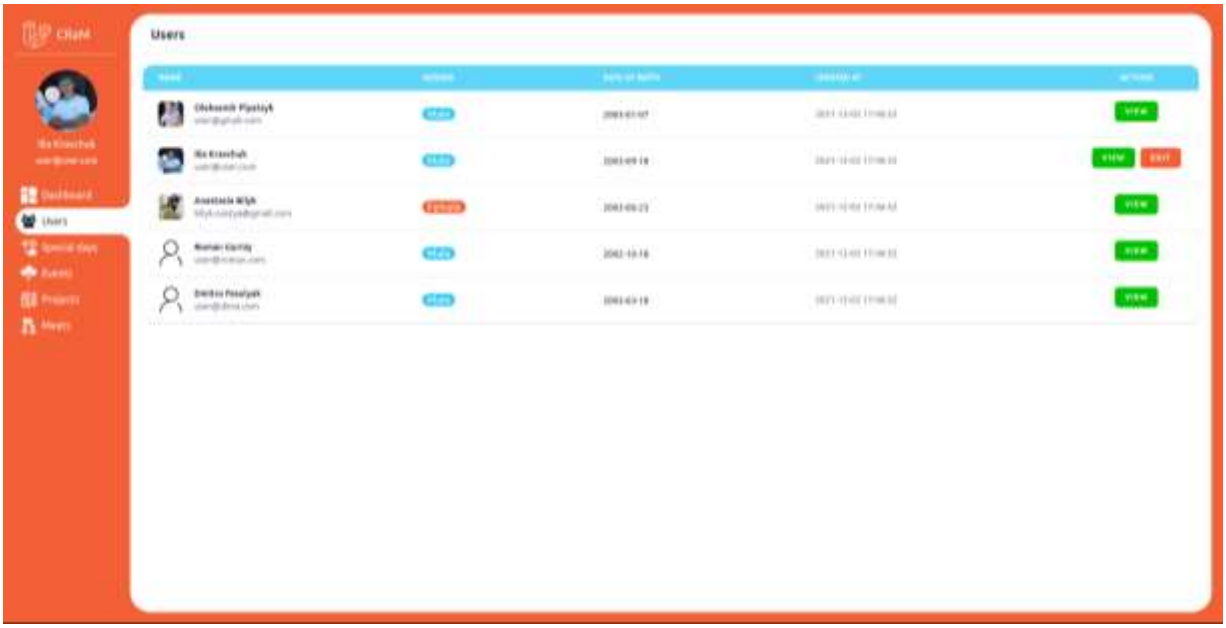


Рисунок 3.5.3 – Сторінка користувачів для звичайного користувача

На сторінці користувачів у звичайного співробітника доступна лише загальна інформація про колег – без можливості змінювати профілі або керувати іншими обліковими записами. Він може переглядати власні відомості та взаємодіяти з елементами, які безпосередньо стосуються його активностей у системі.

Рисунок 3.5.4 – Сторінка Projects для звичайного користувача

Аналогічно, у розділі проектів пересічний користувач бачить лише ті ініціативи, до яких він безпосередньо залучений. Відображається загальний опис, статус проекту, роль працівника в межах команди, але функції керування (редагування або додавання) відсутні.

Рисунок 3.5.5 – Сторінка зустрічей для звичайного користувача

У вкладці зустрічей користувач має доступ до графіка власних запланованих подій – як індивідуальних, так і колективних. Календар синхронізується з іншими модулями системи, що дозволяє формувати єдиний простір робочого планування.

Завдяки візуальній відокремленості дозволеного та обмеженого функціоналу, система забезпечує зручне користування без перевантаження інтерфейсу. Такий підхід дозволяє зберігати структурованість роботи з даними, підвищує безпеку та полегшує адміністрування платформи.

У процесі розробки вебзастосунку CRaM активно використовувалась система контролю версій Git, що забезпечила структуроване ведення коду та зручність у спільній роботі над проектом. Центральним інструментом зберігання коду виступив репозиторій, розміщений на платформі GitLab [8], яка є одним із найпоширеніших сервісів для розміщення Git-репозиторіїв та управління життєвим циклом програмного забезпечення.

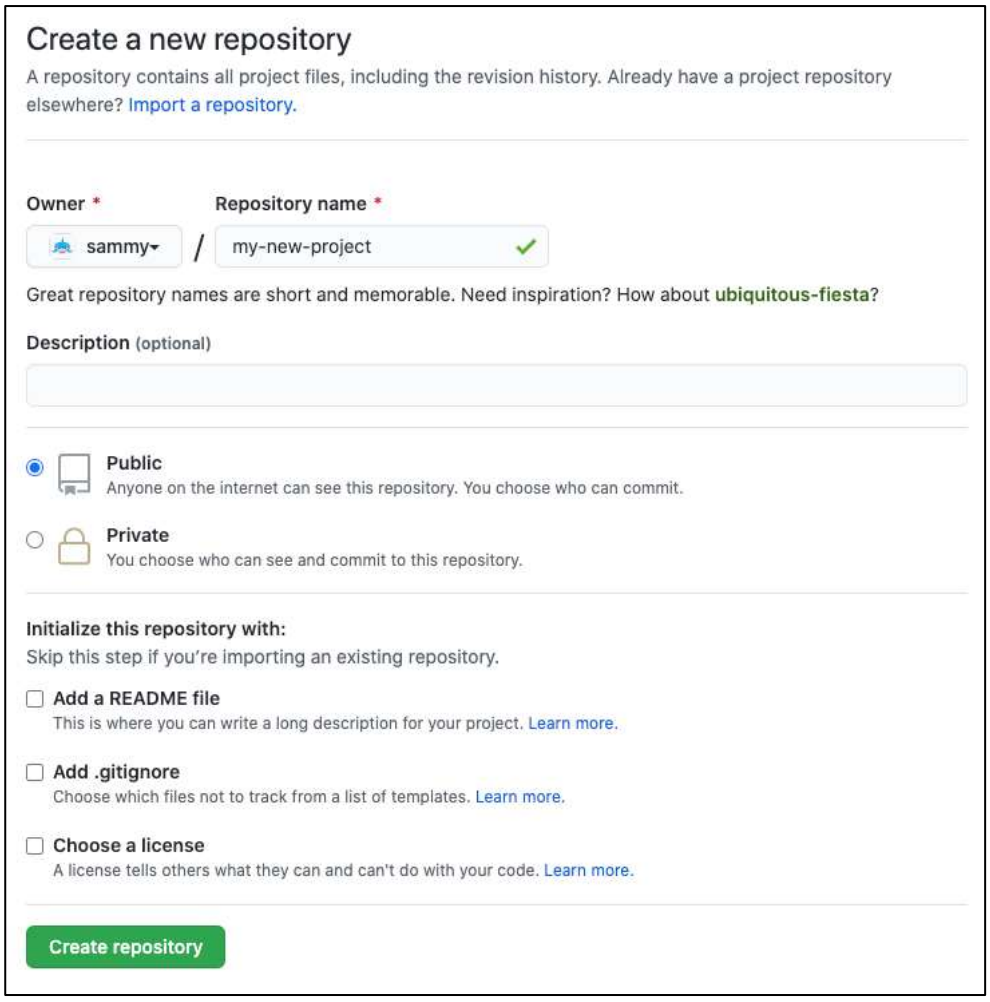


Рисунок 3.5.6 – Приклад створення репозиторію

GitLab дозволяє не лише зберігати історію змін, а й організовувати паралельну роботу над окремими модулями завдяки використанню гілок.

Розробники можуть створювати власні версії функціоналу, тестувати їх окремо від основного коду та, за потреби, повертатися до стабільних версій без ризику втрати важливих змін. У випадку помилок або відкату до попереднього стану проекту GitLab забезпечує повну прозорість та відстежуваність усіх змін.

Рисунок 3.5.7 – Вигляд ієрархії проекту

Сама система контролю версій дозволяє фіксувати кожен етап розробки як окремий знімок стану проекту (commit), що значно спрощує навігацію в історії

розробки. Можна переглядати, хто саме та коли вносив зміни, порівнювати версії між собою, або відновлювати попередні стани у разі помилки.

GitLab також підтримує обговорення змін, створення запитів на злиття (merge requests), рецензування коду, автоматичне тестування та безперервну інтеграцію. Усе це робить його ефективним середовищем для командної розробки – навіть на ранніх етапах, коли структуру проекту ще активно формують

GitLab є не лише хостингом для репозиторіїв коду, а й повноцінною платформою DevOps, що об’єднує інструменти для розробки, тестування, розгортання та моніторингу програмного забезпечення. У межах одного інтерфейсу розробники отримують доступ до засобів керування задачами, CI/CD пайплайнів, автоматичного тестування, вбудованої системи виявлення вразливостей та документації. Це суттєво спрощує координацію роботи над великими проектами та скорочує час на інтеграцію змін.

Завдяки гнучким налаштуванням прав доступу, GitLab дозволяє створювати ролі з різними рівнями повноважень – від звичайного спостерігача до адміністратора проекту. Це забезпечує контроль над кодовою базою та убезпечує від несанкціонованих змін. Крім того, платформа підтримує створення milestone-періодів, ведення backlog'у задач, коментування змін у реальному часі, що особливо зручно для командної взаємодії при паралельній розробці кількох функціональних модулів.

Візуалізація історії змін у проекті представлена у вигляді дерева гілок. Це дозволяє простежити весь шлях розвитку продукту: від початкової ініціалізації репозиторію до останніх оновлень та реалізації нових функцій.

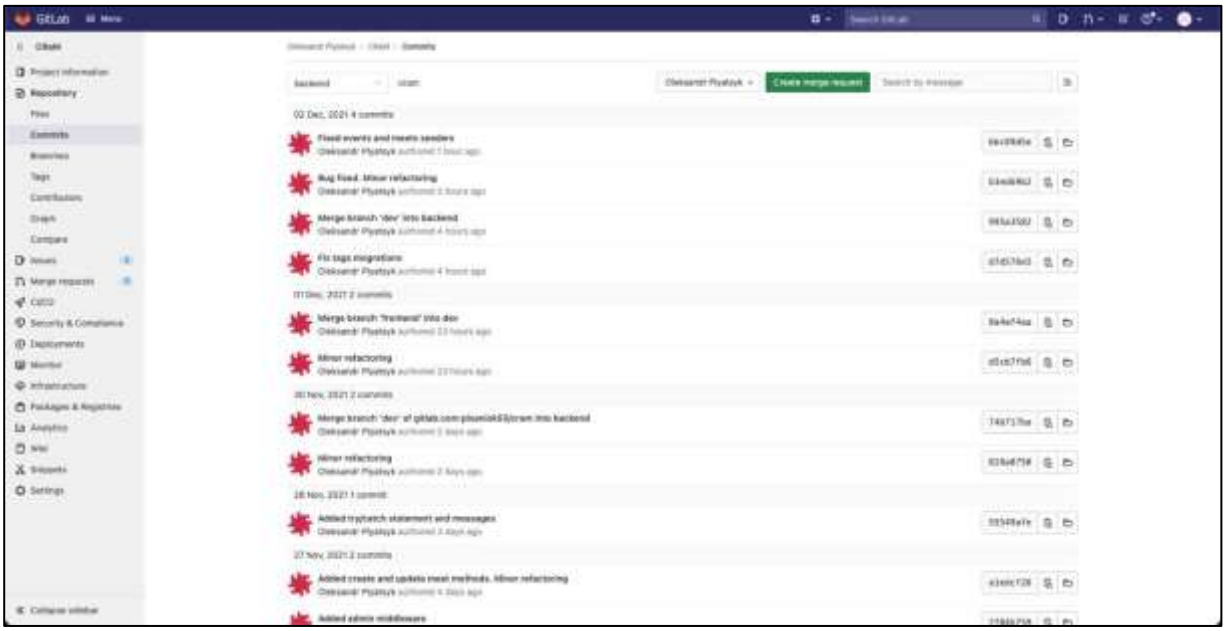


Рисунок 3.5.8 – Вигляд процесу розвитку програми, засобами GitLab

Основна гілка (зазвичай master або main) використовується для зберігання стабільного, працездатного коду. Для додавання нового функціоналу створюються окремі тематичні гілки, які після успішного тестування об'єднуються з основною. Цей підхід дозволяє дотримуватись принципів модульності, відокремлювати експериментальні зміни та зменшувати ризики при масштабуванні системи.

Завдяки використанню Git як основи системи керування версіями, проект залишається максимально адаптивним до змін, а також зручним для підтримки та розвитку. Усі ключові компоненти проекту були оформлені згідно з принципами чистого коду та документовані для подальшої роботи.

Для полегшення ознайомлення з логікою побудови та архітектурою системи, до репозиторію додано два файли README.md – один українською, інший англійською мовою. Вони містять інструкції щодо встановлення, запуску, а також опис структури коду та технологічного стеку.

3.6 Висновки до третього розділу

У процесі реалізації проекту було технічно обґрунтовано вибір стеку технологій, що забезпечив стабільну й масштабовану архітектуру вебзастосунку.

Як серверну основу використано PHP-фреймворк Laravel[9], що надає можливість гнучко реалізовувати REST API, застосовувати принципи MVC, а також керувати даними через ORM. У якості системи управління базами даних обрано MySQL[11], яка дозволяє ефективно зберігати структуровану інформацію про користувачів, події, проєкти та інші сутності. Всі таблиці бази даних були нормалізовані до третьої нормальної форми, що дозволило уникнути надлишковості даних та оптимізувати запити.

Розробка велась із застосуванням сучасного інструментарію: React.js[4] для клієнтської частини, TailwindCSS[13] для стилізації інтерфейсу та Webpack[15] для збирання проєкту. В процесі було впроваджено систему контейнеризації Docker[5], що забезпечує ізольоване середовище для сервісів і полегшує процес розгортання. Застосування Redux[17] дало змогу централізовано управляти станом інтерфейсу, а Axios[16] забезпечив зручний спосіб комунікації між фронтендом і сервером.

Для супроводу проекту використовувалась система контролю версій Git та хостинг репозиторію на GitLab[8], що дозволило чітко організувати історію змін, проводити тестування в гілках, об’єднувати функціонал без ризику пошкодження основної гілки проекту. Кожен компонент і сторінка були розроблені як модулі, що підвищує повторне використання коду та полегшує підтримку.

Особливу увагу було приділено перевірці функціональності системи. Проведено модульне та функціональне тестування ключових сутностей – авторизації, роботи з персоналом, створення особливих днів, перегляду зустрічей тощо. Для зручності користувача було реалізовано динамічні повідомлення, перевірку вхідних даних і механізми контролю доступу.

У підсумку розроблено надійну й масштабовану CRM-систему, що здатна автоматизувати значну частину внутрішніх процесів компанії, зменшити рутинне навантаження на менеджерів та забезпечити прозору взаємодію між усіма учасниками проектів. Система легко розширюється, відповідає сучасним вимогам до веброзробки та готова до використання в реальних умовах.

ВИСНОВКИ

Результатом виконання дипломної роботи є розроблена клієнтська частина вебзастосунку «Customer Relationships and Management» (CRaM), який призначений для автоматизації процесів управління персоналом у рамках діяльності IT-компанії. Проект реалізовано з дотриманням сучасних підходів до розробки програмного забезпечення, із застосуванням об’єктно-орієнтованого програмування, шаблону архітектури MVC та використанням таких технологій, як Laravel, React.js, MySQL, TailwindCSS і Less. Робота над системою розпочалась із дослідження предметної області, постановки задачі, збору вимог і побудови UML-діаграм у середовищі draw.io, що дало змогу системно підійти до проектування логіки взаємодії між модулями.

Фронтенд частину побудовано на компонентній моделі за допомогою React, що дало можливість реалізувати гнучкий, адаптивний інтерфейс користувача, а також оптимізувати повторне використання елементів. Система забезпечує чіткий розподіл прав доступу за ролями (звичайний користувач, адміністратор, супер-адміністратор), дозволяючи зручно керувати основними сутностями: користувачами, подіями, проектами, особливими днями, зустрічами. Завдяки інтеграції REST API забезпечено стабільну та надійну взаємодію між клієнтською та серверною частинами.

Окрема увага приділялась проектуванню процесу тестування. Систему перевіряли на всіх рівнях: модульному, інтеграційному, функціональному та приймальному. Було реалізовано перевірку коректності авторизації, роботи форм, валідації введених даних, реакції на типові помилки та взаємодії з базою даних. Сценарії тестування охоплювали як позитивні, так і негативні гілки логіки, що дозволило переконатися в надійності реалізованого функціоналу. Тестування підтвердило, що система функціонує відповідно до вимог, які були поставлені на етапі технічного завдання.

Розроблена система має низку важливих переваг: чітко структурована архітектура, компонентний інтерфейс, можливість масштабування,

централізована модель доступу, інтеграція через API та високий рівень адаптивності до змін. Вона підходить для використання в малих і середніх компаніях, а також передбачає розширення функціональності без суттєвих змін у кодовій базі. Проте існують і певні обмеження. На поточному етапі реалізація працює локально, що обмежує доступ до неї в середовищах без налаштованої інфраструктури. Також поки не реалізовані деякі заплановані модулі, зокрема сервіс нотаток, трекер часу та інтегрований чат. Мобільна оптимізація ще не завершена, а аналітичний функціонал представлено лише на базовому рівні.

Узагальнюючи, розроблений вебдодаток CRaM є завершеним, функціональним програмним продуктом, який демонструє відповідність технічному завданню, підтримує роботу з основними інформаційними об’єктами й забезпечує інтуїтивну взаємодію користувача з даними. Програмна система готова до практичного впровадження у внутрішні процеси компанії та має потенціал для подальшого розвитку.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. CRM платформа Bitrix24. URL: https://www.bitrix24.com/ (дата звернення: 13.02.2025).
2. CRM система. URL: https://terrasoft.ua/page/definition-crm/ (дата звернення: 30.03.2025).
3. Сервіс Draw.io. URL: https://www.app.diagrams.net/ (дата звернення: 04.05.2025).
4. Офіційна документація React. URL: https://reactjs.org/ (дата звернення: 08.05.2025).
5. ПЗ для контейнеризації додатків Docker. URL: https://docker.com/ (дата звернення: 08.04.2025).
6. Вебдодаток для створення діаграм Draw.io. URL: https://app.diagrams.net/ (дата звернення: 28.03.2025).
7. Вебсервіс для хостингу IT-проектів GitHub. URL: https://github.com/ (дата звернення: 07.03.2025).
8. Вебсервіс для хостингу IT-проектів GitLab. URL: https://about.gitlab.com/ (дата звернення: 21.02.2025).
9. PHP Framework Laravel. URL: https://laravel.com/ (дата звернення: 25.04.2025)
10. Laravel docs. URL: https://laravel.com/docs/9.x/ (дата звернення: 14.05.2025).
11. СУБД MySQL – Режим доступу: https://www.mysql.com/
12. СУБД MySQL. URL: https://www.mysql.com/ (дата звернення: 12.04.2025).
13. Інтегроване середовище розробки PhpStorm. URL: https://www.jetbrains.com/phpstorm/ (дата звернення: 10.04.2025).
14. CSS фреймворк tailwindCSS. URL: https://tailwindcss.com/ (дата звернення: 09.05.2025).
15. Інструмент для організації проектів в дошки Trello. URL: https://trello.com/ (дата звернення: 16.02.2025).

16. Збірник модулів Webpack. URL: https://webpack.js.org/ (дата звернення: 18.02.2025).
17. Axios бібліотека. URL: https://axios-http.com/docs/intro/ (дата звернення: 12.02.2025).
18. Сховище даних Redux. URL: https://redux.js.org/ (дата звернення: 22.03.2025).
19. Форматувальник файлів. URL: https://eslint.org/ (дата звернення: 11.04.2025).
20. Препроцесор .less. URL: https://lesscss.org/ (дата звернення: 27.04.2025).
21. JS бібліотека для роботи з масивами та об’єктами. URL: https://lodash.com/ (дата звернення: 25.04.2025).
22. JS бібліотека роботи з датами. URL: https://momentjs.com/ (дата звернення: 03.04.2025).
23. Learning React – Alex Banks & Eve Porcello, 2019р. (дата звернення: 26.04.2025).
24. Занурення в патерни проектування – Олександр Швець, 2021р. (дата звернення: 07.05.2025).
25. Martin Fowler. Patterns of Enterprise Application Architecture. – Addison-Wesley, 2002. (дата звернення: 02.05.2025).
26. Craig Walls. Spring in Action. – Manning Publications, 2018. (дата звернення: 17.03.2025).
27. PHP: The Right Way. URL: https://phptherightway.com (дата звернення: 08.05.2025).
28. Архітектура мікросервісів. URL: https://microservices.io/ (дата звернення: 19.05.2024).
29. UX/UI Принципи. URL: https://www.nngroup.com/articles/definition-user-experience/ (дата звернення: 20.05.2024).
30. Agile Manifesto. URL: https://agilemanifesto.org/ (дата звернення: 21.05.2024).

31. Scrum Guide 2020. URL: https://scrumguides.org/ (дата звернення: 22.05.2024).
32. Візуалізація даних з Chart.js. URL: https://www.chartjs.org/docs/latest/(дата звернення: 23.05.2024).
33. Стандарти API Microsoft. URL: https://learn.microsoft.com/en-us/azure/architecture/best-practices/api-design (дата звернення: 24.05.2024).
34. Docker Compose Docs. URL: https://docs.docker.com/compose/ (дата звернення: 14.05.2024).
35. Документація про JWT (JSON Web Token). URL: https://jwt.io/introduction/ (дата звернення: 15.05.2024).
36. UML Specification – Object Management Group. URL: https://www.omg.org/spec/UML (дата звернення: 19.03.2025).
37. Head First Design Patterns. URL: https://www.oreilly.com/library/view/head-first-design/9781492077992/ (дата звернення: 16.04.2025).
38. Clean Architecture. URL: https://www.oreilly.com/library/view/clean-architecture-a/9780134494272/ (дата звернення: 01.04.2025).
39. Understanding ECMAScript 6. URL: https://leanpub.com/understandinges6/read (дата звернення: 02.03.2025).
40. Fundamentals of Software Architecture. URL: https://www.oreilly.com/library/view/fundamentals-of-software/9781492043447/ (дата звернення: 04.03.2025).
41. JavaScript: The Definitive Guide. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide (дата звернення: 15.02.2025).

ДОДАТОК А

Код (лістинг) програми

А.1 – Початковий компонент App.jsx

```
import React from 'react'
import { Provider } from 'react-redux'
import { CookiesProvider } from 'react-cookie'
import { applyMiddleware, compose, createStore }
from 'redux'
import createSagaMiddleware from 'redux-saga'

import rootReducers from './redux/reducers'

import Router from './components/Router'

import { ToastContainer } from 'react-toastify'
import 'react-toastify/dist/ReactToastify.css'
import '/var/www/html/public/css/index.css'
import './utils/i18n'


import rootSaga from './redux/sagas/rootSaga'

const sagaMiddleware = createSagaMiddleware()

const composeEnhancers =
window.__REDUX_DEVTOOLS_EXTENSION_CO
MPOSE__ || compose
const store = createStore(rootReducers,

composeEnhancers(applyMiddleware(sagaMiddlewa
re))
)
sagaMiddleware.run(rootSaga)

const App = () => {
  return (
    <CookiesProvider>
      <Provider store={store}>
        <Router/>
        <ToastContainer/>
      </Provider>
    </CookiesProvider>
  )
}

export default App
```

А.2 –Компонент для управління сторінками Router.jsx

```
import React, { useEffect, useState } from 'react'
import { useCookies } from 'react-cookie'
import { BrowserRouter, Navigate, Route, Routes }
from 'react-router-dom'
import PropTypes from 'prop-types'
import { useDispatch, useSelector } from 'react-
redux'

import { setUser } from '../redux/actions/user'

import { COOKIE_USER_TOKEN, USER_ROLES }
from '../utils/constants'
import { getPagesRoutes } from '../utils/pagesRoutes'

import api from '../services/api'
import { routes } from '../utils/routes'
import Loader from './Loader'

const Router = () => {
  const user = useSelector(state => state.user.data)
  const [isLoading, setIsLoading] = useState(true)
  const dispatch = useDispatch()
  const [cookies] =
useCookies([COOKIE_USER_TOKEN])

  useEffect(() => {
    setTimeout(() => {
      if (cookies.COOKIE_USER_TOKEN) {
        api.get('/auth/me')
          .then(response =>
dispatch(setUser(response?.user)))
          .catch(() => console.warn('GUEST'))
          .finally(() => setIsLoading(false))
      } else {
        setIsLoading(false)
      }
    }, 1000)
  }, [])

  return (
    <>
      <Loader isLoading={isLoading}
isFullScreen/>

      {
        !isLoading &&
        <BrowserRouter>
          <Routes>
            {

getPagesRoutes(USER_ROLES[user?.role_id]).map(
page =>
                <Route
                  key={page.name}
                  path={page.path}
                  element={
                    <>
                      {

page?.role?.includes(USER_ROLES[user?.role_id]) ?

<page.PageComponent/> :
                        <Navigate
to={routes.notFound} replace/>
                      }
                    </>
                  }/>
              )
            }
          </Routes>
```

```
            </BrowserRouter>
          }
        </>
    )

}
```

```
Router.propTypes = {
    user: PropTypes.object,
}

export default Router
```

А.3 –Сторінка логінізації

```
import React, { useState } from 'react'
import { useTranslation } from 'react-i18next'
import { useNavigate } from 'react-router-dom'
import { useDispatch } from 'react-redux'
import { useCookies } from 'react-cookie'

import { setUser } from '../../redux/actions/user'

import Guest from '../../components/Layout/Guest.jsx'
import InputLabel from
'../../components/Form/InputLabel'
import Button from '../../components/Form/Button'
import TextLink from '../../components/TextLink'

import { routes } from '../../utils/routes'
import { COOKIE_USER_TOKEN } from
'../../utils/constants'
import api from '../../services/api'

const LogIn = () => {
    const { t } = useTranslation()
    const navigate = useNavigate()
    const dispatch = useDispatch()
    // eslint-disable-next-line no-unused-vars
    const [cookies, setCookie] =
useCookies([COOKIE_USER_TOKEN])

    const [email, setEmail] = useState('')
    const [password, setPassword] = useState('')
    const [remember, setRemember] = useState(false)

    const logInRequest = e => {
        e.preventDefault()

        const payload = {
            email,
            password
        }

        api.post('/auth/login', payload, { headers: {
'Accept': 'application/json' } })
            .then(response => {
                const authorization =
response?.authorisation
                if (authorization) {
                    const ttl = remember ?
                        60 * 60 * 24 * 7 : // 1 week from now
                        60 * response?.authorisation?.ttl // 1
hour from now
                    setCookie(COOKIE_USER_TOKEN,
authorization?.token, { path: '/', maxAge: ttl })
                }
                const userData = response?.user
                if (userData) {
                    new Promise(resolve => {
                        dispatch(setUser(userData))
                        resolve()
                    }).then(() => navigate(routes.user, {
replace: true }))
                }
                //
dispatch(addMessageTrigger(response?.message)) //
TODO saga trigger
            })
            .catch(err => {
                console.log(err)
            })
    }

    return (
        <Guest>
            <form onSubmit={logInRequest}>
                <div>
                    <InputLabel id="email" type="email"
className="block w-full" required autoFocus
                                autoComplete="email"
                                placeholder={t('Enter your
email...')}
                                value={email}
                                onChange={e =>
setEmail(e.target.value)}>
                        {t('Email')}
                    </InputLabel>
                </div>

                <div className="mt-4">
                    <InputLabel id="password"
type="password" className="block w-full" required
                                autoComplete="current-
password"
                                placeholder={t('Enter your
password...')}
                                value={password}
                                onChange={e =>
setPassword(e.target.value)}>
                        {t('Password')}
                    </InputLabel>
                </div>

                <div className="block mt-4">
                    <InputLabel id="remember_me"
type="checkbox" className="text-secondary"
                                onChange={e =>
setRemember(e.target.checked)}>
                        {t('Remember me')}
```

```
            </InputLabel>
          </div>

          <div className="flex items-center justify-
end mt-4">
            <TextLink className="text-sm"
to="#"> {/* TODO Forgot your password */}
              {t('Forgot your password?')}
            </TextLink>
            <Button type="submit" color="primary"
className="ml-3">
              {t('Log in')}
            </Button>
          </div>
        </form>
      </Guest>
    )
}

export default LogIn
```

A.4 –Сторінка користувача

```
import React, { useEffect, useState } from 'react'
import PropTypes from 'prop-types'
import { useSelector } from 'react-redux'
import { Link, useParams } from 'react-router-dom'
import moment from 'moment'
import { isEmpty, isUndefined } from 'lodash'

import App from '../../components/Layout/App'
import Header from '../../components/Layout/Header'

import LogOut from '../../components/LogOut'
import ActionLink from
'../../components/ActionLink'
import Svg from '../../components/Svg'
import Image from '../../components/User/Image'

import { routes } from '../../utils/routes'
import api from '../../services/api'

const Index = () => {
  const loggedUser = useSelector(state =>
state.user.data)
  const [user, setUser] = useState({})
  const [meets, setMeets] = useState()
  const [projects, setProjects] = useState()
  const { id } = useParams()

  const specialDays = [
    {
      bgColor: 'bg-green',
      type: 'vacation'
    },
    {
      bgColor: 'bg-orange',
      type: 'home'
    },
    {
      bgColor: 'bg-red',
      type: 'sick'
    },
    {
      bgColor: 'bg-yellow-dark',
      type: 'business_trip'
    },
  ]

  useEffect(() => {
    api.get(routes.userById.replace(':id', id ??
loggedUser.id))
      .then(response => {
        setUser(response?.user)
      })
      .catch(() => setUser({}))

    api.get(routes.meetsByUserId.replace(':id', id ??
loggedUser.id))
      .then(response => {
        setMeets(response?.meets)
      })
      .catch(() => setMeets([]))

    api.get(routes.projectsByUserId.replace(':id', id
?? loggedUser.id))
      .then(response => {
        setProjects(response?.projects)
      })
      .catch(() => setProjects([]))
  }, [loggedUser, id])

  // TODO implement user.status (0 - disabled)
  return (
    <App isLoaded={!isEmpty(user)}>
      {
        !isEmpty(user) &&
        <>
          <Header label="User Profile">
            {(id === loggedUser.id ||
isUndefined(id)) &&
              <LogOut/>
            }

            <ActionLink to={routes.userList}
color="primary">
              Go back
            </ActionLink>
          </Header>

          <div className={`w-full py-5 flex
${meets?.length > 1 && projects?.length > 1 ?
'justify-between' : ''}`}>
            <div className="w-1/6">
              <div className="bg-secondary
rounded-2xl p-4">
                <Image className="object-
cover rounded-full" borderClassName="rounded-
full" path={user.image}/>
                <div className="pt-2 pb-6 px-4
bg-secondary-light rounded-2xl rounded-b-none -m-4
```

```
mt-4 text-lg">
                        {user.name}
                    </div>
                    <div className="pt-2 pb-6 px-4
bg-secondary-lighter rounded-2xl rounded-b-none -
m-4 mt-0">
                        {'Mail me: '}
                        <div className="inline-block
after:block after:w-0 after:h-[1px] after:bg-black
after:duration-300 hover:after:w-full">
                            <a
href={`mailto:${user.email}`}>
                                {user.email}
                            </a>
                        </div>
                    </div>
                    <div className="z-30 relative
pt-2 bg-secondary-lightest rounded-2xl p-4 -mx-4 mt-
0">
                        <p className="pb-2">
                            {'Available special days:'}
                        </p>
                        <div className="flex text-
white">

{specialDays.map((specialDay, ind) =>
                                <div className={`w-1/4
relative box-border pt-[25%] ${specialDay.bgColor}
rounded-xl`}
                                     key={ind}>
                                    <div
className="absolute top-0 bottom-0 left-0 right-0 p-
1 pt-0 text-center text-lg">

{user[specialDay.type] ?? 0}
                                        <div className="p-
2.5 pt-0 -mt-0.5">
                                            <Svg
name={specialDay.type}/>
                                        </div>
                                    </div>
                                </div>
                            )}
                        </div>
                    </div>
                    <div className="z-20 relative
pt-6 pb-2 px-4 bg-secondary-lighter rounded-2xl
rounded-t-none -m-4">
                        {
                            user.team ?
                                `Team: ${user.team}` :
                                `Not a part of any team`
                        }
                    </div>
                    <div className="z-10 relative
pt-6 pb-2.5 px-4 bg-secondary-light rounded-2xl
rounded-t-none -m-4">
                        {
                            user.type ?
                                `Qualification:
${user.type}` :
                                `Not qualified yet`
                        }
                    </div>
                    <div className="z-10 relative
pt-6 pb-2.5 px-4 rounded-2xl -m-4">
                        {`Birthday: ${user.dob}`}
                    </div>
                    {
                        user.tags &&
                        <div className="pt-4 flex
flex-wrap text-white">
                            {

user.tags.split(',').map((tag, ind) =>
                                    <div className="bg-
primary rounded-lg px-1 py-0.5 mr-1 mb-1"
key={ind}>
                                        <i>#{tag}</i>
                                    </div>
                                )
                            }
                        </div>
                    }
                </div>
            </div>

            {
                meets &&
                <div className="max-w-
[640px]">
                    <div className="bg-secondary
rounded-2xl ml-4 p-4 pb-0">
                        {
                            meets.length > 0 ?
                                <div className="flex
flex-wrap justify-between gap-4">
                                    {meets.map((meet, ind)
=>
                                        <div
className={`ticket-small ${ind % 2 === 1 ? 'pl-4' :
''}`} key={ind}>
                                            <div
className="ticket-small-main">
                                                <span
className="ticket-small-top"/>
                                                <span
className="ticket-small-bottom"/>
                                                <div
className="ticket-small-content flex-row items-
center">
                                                    <div
className="h-[57px] min-w-[57px]">
                                                        {/*TODO:
Dynamically get icon from ${meet}*/}
                                                        <Svg
className="h-full" name="salary_review"/>
                                                    </div>

                                                    <div
className="border-l border-l-[2px] border-primary-
light px-[4px] max-w-2/3 min-h-full flex flex-col
justify-center">
                                                        <div
className="font-semibold text-sm">
```

```
{meet.name}
                                        </div>
                                        <div
className="line-clamp-1 font-bold text-sm">

{meet.type}
                                        </div>
                                        <div
className="py-1 text-
2xs">{meet.initiator_user_name}</div>
                                        <div
className="line-clamp-2 mb-1 text-3xs">

{meet.description}
                                        </div>
                                        <div
className="text-3xs">

{`Created at:
${moment(meet.created_at).format('D.M.YYYY')}`}
                                        </div>
                                    </div>
                                </div>
                                <div
className="ticket-small-part">
                                    <span
className="ticket-small-top"/>
                                    <span
className="ticket-small-bottom"/>
                                    <div
className="ticket-small-content flex-col pt-1">
                                    <span
className="text-3xs">
                                        {'Won\'t miss
at:'}
                                    </span>
                                        <h3
className="font-bold text-xl leading-tight -mb-
0.5">

{moment(meet.meet_date).format('Do')}
                                        </h3>
                                        <h4
className="text-xs">

{moment(meet.meet_date).format('MMM')}
                                        </h4>
                                        <h4
className="text-2xs leading-2">

{moment(meet.meet_date).format('YYYY')}
                                        </h4>
                                        <span
className="text-2xs">

{/*{meet.users_id.split(',').length} invited TODO
how many invited*/}
                                            ∞ invited
                                    </span>
                                    </div>
                                </div>
                            </div>
                        )}
                    </div>
                </div>
                :
                <div className="w-full
pb-4">
                    <div className="mx-
auto rounded-4xl bg-white border border-gray-200">
                        <div className="p-
6">
                            {'There are no
meets for this user!'}
                        </div>
                    </div>
                </div>
            }
        </div>
    </div>
}

{
    projects &&
    <div className="max-w-
[554px]">
        <div className="bg-secondary
rounded-2xl ml-4 p-4 pb-0">
            {
                projects.length > 0 ?
                    <div className="flex
flex-wrap">
                        {projects.map((project,
ind) =>
                            //TODO
projects.description link
                            <Link
to={routes.projectById.replace(':id', project.id)}

className={`${projects.length <= 1 ? 'min-w-
[253px]' : 'w-1/2'} mb-4 ${ind % 2 === 1 ? 'pl-4' :
''}`}
                                  key={ind}>
                                <div
className={`${project.is_active ? '' : 'grayscale'}`}>
                                    <div
className="bg-white rounded-xl border-4 border-
primary py-3 px-2 min-h-full max-w-full flex flex-
col justify-between text-sm">
                                        <div
className="max-w-full pl-1">
                                            <h2
className="text-lg font-bold">

{project.project_name}
                                            </h2>
                                            <h3
className="font-medium">
                                                {`Role:
${project.project_role}`}
                                            </h3>
                                            <p
className="mt-2">

{`Joined: ${project.date_from}`}
                                            </p>
```

```
                                        <p
className="">
                                            {

project.date_to ?

`Left: ${project.date_to}` :

'Still working...'
                                            }
                                        </p>
                                    </div>
                                    <div
className="mt-2 pl-1">
                                        <p
className="text-gray-600 leading-none line-clamp-
2">
                                            {

project.description ??
                                                'There
is no description for this project.'
                                            }
                                        </p>
                                    </div>
                                </div>
                            </div>
                        </Link>
                    )}
                </div>
                :
                <div className="w-full
```

```
pb-4">
                        <div className="mx-
auto rounded-4xl bg-white border border-gray-200">
                            <div className="p-
6">
                                {'There are no
projects for this user!'}
                            </div>
                        </div>
                    </div>
                }
            </div>
        </div>
    }

    {/*{{-- TODO: show gender --}}*/}
    {/*{{-- <p>{{__('Gender') . ': ' . $user-
>gender}}</p> --}}*/}
                </div>
            </>
        }
    </App>
  )
}

Index.propTypes = {
  user: PropTypes.object
}

export default Index
```

A.5 –Сторінка редагування користувача

```
import React, { useEffect, useState } from 'react'
import { forEach, isEmpty } from 'lodash'
import { useNavigate, useParams } from 'react-
router-dom'

import App from '../../components/Layout/App'
import Header from '../../components/Layout/Header'

import ActionLink from
'../../components/ActionLink'
import Button from '../../components/Form/Button'
import InputLabel from
'../../components/Form/InputLabel'
import SelectLabel from
'../../components/Form/SelectLabel'

import { routes } from '../../utils/routes'
import api from '../../services/api'

const Edit = () => {
  const navigate = useNavigate()
  const [userData, setUserData] = useState({})
  const [payload, setPayload] = useState({})
  const [submitType, setSubmitType] = useState('')
  const { id } = useParams()

  useEffect(() => {
    api.get(routes.userEdit.replace(':id', id))
      .then(response => {
        setUserData(response?.user)
      })
      .catch(() => setUserData({}))
  }, [])

  const editRequest = async e => {
    e.preventDefault()

    const formData = new FormData()
    forEach(payload, (value, key) =>
formData.append(key, value))
    formData.append('_method', 'PUT')

    await api.post(routes.userById.replace(':id', id),
formData, { headers: { 'Content-Type':
'multipart/form-data' } })
      .then(response => console.log(response))

    if (submitType === 'EditLogin') {
      // navigate() TODO Login after Edit
    }

    navigate(routes.userList)
  }
```

```
  return (
    <App isLoaded={!isEmpty(userData)}>
      {
        !isEmpty(userData) &&
        <>
          <Header label={`Edit ${payload.name
?? userData.name}'s profile`}>
            <ActionLink to={routes.userList}
color="primary">
              {'Go back'}
            </ActionLink>
          </Header>
          <div className="w-1/2 py-5">
            <form onSubmit={editRequest}>
              <div>
                <InputLabel id="name"
type="text" className="block w-full" required
autoFocus

autoComplete="username"
                            placeholder={'Edit User
name...'}
                            value={payload.name ??
userData.name}
                            onChange={e =>
setPayload({ ...payload, name: e.target.value })}>
                  {'Name'}
                </InputLabel>
              </div>

              <div className="flex">
                <div className="mt-4 pr-4 w-
1/2">
                  <SelectLabel id="gender"
className="block mt-1 w-full" required

value={`${payload.gender ?? userData.gender}`}
                              onChange={e =>
setPayload({ ...payload, gender: e.target.value })}
                              options={[ // TODO
Get genders dynamically
                                {
                                  value: '',
                                  label: 'Select user
gender',

                                  disabled: true,
                                  hidden: true
                                },
                                { value: '1', label:
'Male' },
                                { value: '0', label:
'Female' },
                              ]}>
                    {'Gender'}
                  </SelectLabel>
                </div>

                <div className="mt-4 w-1/2">
                  <InputLabel id="dob"
type="date" className="block w-full" required
                              autoComplete="bday"
                              value={payload.dob ??
userData.dob}
                              onChange={e =>
setPayload({ ...payload, dob: e.target.value })}>
                    {'Date of Birth'}
                  </InputLabel>
                </div>
              </div>

              <div className="mt-4 pr-4 w-
1/2">
                  <SelectLabel id="role"
className="block mt-1 w-full" required
                              value={`${payload.role
?? userData.role}`}
                              onChange={e =>
setPayload({ ...payload, role: e.target.value })}
                              options={[ // TODO Get
roles dynamically
                                {
                                  value: '',
                                  label: 'Select user
role',
                                  disabled: true,
                                  hidden: true
                                },
                                { value: '1', label:
'User' },
                                { value: '2', label:
'Admin' },
                              ]}>
                    {'Role'}
                  </SelectLabel>
                </div>

                <div className="mt-4">
                  <InputLabel id="image"
type="file" className="block w-full" // TODO
image afterload
                              autoComplete="photo"
                              onChange={e =>
setPayload({ ...payload, image: e.target.files[0] })}>
                    {'Image'}
                  </InputLabel>
                </div>

                <div className="mt-4">
                  <InputLabel id="email"
type="email" className="block w-full" required
                              autoComplete="email"
                              placeholder={'New
email...'}
                              value={payload.email ??
userData.email}
                              onChange={e =>
setPayload({ ...payload, email: e.target.value })}>
                    {'Email'}
                  </InputLabel>
                </div>

                <div className="mt-4">
                  <InputLabel id="password"
type="password" className="block w-full"
                              autoComplete="new-
password"
```

```
                                onChange={e =>
setPayload({ ...payload, password: e.target.value
})}>
                            {'Password'}
                        </InputLabel>
                    </div>

                    {/*<div className="mt-4"> TODO
pass confirm*/}
                    {/*    <InputLabel
id="password_confirmation" type="password"
className="block mt-1 w-full"*/}
                    {/*                autoComplete="new-
password"*/}
                    {/*                onChange={e =>
setPayload({ ...payload, password_confirmation:
e.target.value })}>*/}
                    {/*        {'Confirm Password'}*/}
                    {/*    </InputLabel>*/}
                    {/*</div>*/}

                    <div className="flex items-center
justify-end mt-4">
                        <Button color="green"
onClick={() => setPayload({})} className="ml-3"

disabled={isEmpty(payload)}>
                            {'Restore Defaults'}
                        </Button>

                        {/*TODO Login AS user*/}
                        {/*<Button type="submit"
color="secondary" onClick={() =>
setSubmitType('EditLogin')}*/}
                        {/*        className="ml-3"*/}
                        {/*
disabled={isEmpty(payload)}>*/}
                        {/*    {'Edit & Login'}*/}
                        {/*</Button>*/}

                        <Button type="submit"
color="primary" className="ml-3"
disabled={isEmpty(payload)}>
                            {'Edit'}
                        </Button>
                    </div>
                </form>
            </div>
        </>
    }
</App>
  )
}

export default Edit
```

A.6 –Сторінка перегляду користувачів

```
import React, { useState, useEffect } from 'react'
import moment from 'moment'
import { isEmpty } from 'lodash'

import App from '../../components/Layout/App'
import Header from '../../components/Layout/Header'
import ActionLink from
'../../components/ActionLink'
import ModalDelete from
'../../components/Modal/ModalDelete'
import Image from '../../components/User/Image'

import { routes } from '../../utils/routes'
import api from '../../services/api'

const List = () => {
  const [users, setUsers] = useState([])
  const [deleteUser, setDeleteUser] = useState({})

  useEffect(() => {
    if (isEmpty(deleteUser)) {
      Promise.all([
        api.get(routes.user)
          .then(response => {
            setUsers(response?.users)
          })
          .catch(() => setUsers([]))
      ])
    }
  }, [deleteUser])

  return (
    <App isLoaded={!isEmpty(users)}>
      {
        !isEmpty(users) &&
        <>
          <Header label="Users">
            {/*@if(Auth::user()->role_as == 1)
TODO ACL*/}
            <ActionLink to={routes.userCreate}
color="primary">
              Create User
            </ActionLink>
            {/*@endif*/}
          </Header>

          <div className="my-4 shadow
overflow-hidden border-b border-gray-200 rounded-
2xl">
            <table className="min-w-full divide-
y divide-gray-200">
              <thead className="bg-secondary-
light text-white">
                <tr className="text-xs font-bold
uppercase">
                  {
                    ['Name', 'Gender', 'Date of
birth', 'Type', 'Team'].map((th, ind) =>
                      <th className="px-6 py-3
text-left" key={ind}>
                        {th}
                      </th>
```

```
                        )
                    }
                    <th className="px-6 py-3 text-
center w-[10%]">
                        Actions
                    </th>
                </tr>
                </thead>
                <tbody className="bg-white
divide-y divide-gray-200">
                {
                    users.map((user, ind) =>
                        <tr key={ind}>
                            <td className="px-6 py-4
w-1/4">
                                <div className="flex
items-center">
                                    <div className="flex-
shrink-0 h-10 w-10">
                                        <Image
className="h-10 w-10 rounded-lg"
borderClassName="rounded-lg"
path={user.image}/>
                                    </div>
                                    <div className="ml-
4">
                                        <div
className="text-sm font-medium text-gray-900">
                                            {user.name}
                                        </div>
                                        <div
className="text-sm text-gray-500">
                                            {user.email}
                                        </div>
                                    </div>
                                </div>
                            </td>
                            <td className="px-6 py-
4">
                                <div className={'px-2
inline-flex leading-5 rounded-full text-white ' +
                                    (user.gender === 'Male'
? 'bg-secondary' : 'bg-primary')}>
                                    {user.gender}
                                </div>
                            </td>
                            <td className="px-6 py-4
text-sm text-gray-500">

{moment(user.dob).format('LL')}
                            </td>
                            <td className="px-6 py-4
text-sm text-gray-900">
                                {user.type}
                            </td>
                            <td className="px-6 py-4
text-sm text-gray-900">
                                {user.team}
                            </td>
                            <td className="px-6 py-4
text-right flex justify-center">
                                <ActionLink
to={routes.userById.replace(':id', user.id)}
color="green">
                                    {'View'}
                                </ActionLink>

                                {/*@if(Auth::user()-
>role_as == 1)*/} {/*TODO ACL*/}
                                {/*    @if(Auth::id() ==
$user->id)*/}
                                {/*        <x-button
disabled :color="'secondary'">*/}
                                {/*            {{
__('LogIn')}}*/}
                                {/*        </x-button>*/}
                                {/*    @else*/}
                                {/*TODO Login as*/}
                                {/*<ActionLink
to="route('users.login-as') + user.id"
color="secondary">*/}
                                {/*    {'Log In'}*/}
                                {/*</ActionLink>*/}
                                {/*    @endif*/}

                                <ActionLink
to={routes.userEdit.replace(':id', user.id)}
color="primary">
                                    {'Edit'}
                                </ActionLink>

                                {/*@if(Auth::id() ==
$user->__get(\Modules\User\Entities\Users::ID))
TODO ACL*/}
                                {/*    <x-button disabled
:color="'red'">*/}
                                {/*
{{__('Delete')}}*/}
                                {/*    </x-button>*/}
                                {/*@else*/}
                                {/*<x-button
className="user__openModal" value="{{ 'profile/' .
$user->__get(\Modules\User\Entities\Users::ID) . ':' .
$user->__get(\Modules\User\Entities\Users::NAME)
. '\'s' }}" :color="'red'">*/}
                                {/*    {{__('Delete')}}*/}
                                {/*</x-button>*/}
                                {/*@endif*/}

                                <ActionLink onClick={()
=> setDeleteUser(user)} color="red">
                                    {'Delete'}
                                </ActionLink>

                                {/*@elseif(Auth::user()-
>role_as == 0 && $user->id == Auth::user()-
>getAuthIdentifier()) TODO ACL*/}
                                {/*    <x-link href="{{
route('users.profile.edit', ['id' => $user->id]) }}"
:color="'primary'">*/}
                                {/*        {{__('Edit')}}*/}
                                {/*    </x-link>*/}
                                {/*@endif*/}
                            </td>
                        </tr>
                    )
```

```
                    }
                    </tbody>
                </table>
                {/* TODO pagination */}
                {/*<div className="links-
wrapper">*/}
                {/*    {{ $users->links()}}*/}
                {/*</div>*/}
            </div>

            <ModalDelete entity={deleteUser}
entitySetter={setDeleteUser}
deleteApiPath={`/user/${deleteUser.id}`}/>
        </>
      }
    </App>
  )
}

export default List
```

A.7 – Сторінка реєстрації користувача

```
import React, { useState } from 'react'
import { forEach } from 'lodash'
import { useNavigate } from 'react-router-dom'

import Button from '../../components/Form/Button'
import InputLabel from
'../../components/Form/InputLabel'
import SelectLabel from
'../../components/Form/SelectLabel'

import App from '../../components/Layout/App'
import Header from '../../components/Layout/Header'

import ActionLink from
'../../components/ActionLink'

import { routes } from '../../utils/routes'
import api from '../../services/api'

const Register = () => {
  const navigate = useNavigate()
  const [payload, setPayload] = useState({
    name: '',
    gender: '',
    dob: '',
    role: '',
    image: null,
    email: '',
    password: '',
    // password_confirmation: '', TODO pass
confirm
  })
  const [submitType, setSubmitType] = useState('')

  const registerRequest = async e => {
    e.preventDefault()

    const formData = new FormData()
    forEach(payload, (value, key) =>
formData.append(key, value))

    await api.post(routes.user, formData, { headers:
{ 'Content-Type': 'multipart/form-data' } })
      .then(response => console.log(response))

    if (submitType === 'RegisterLogin') {
      // navigate() TODO Login after register
    }
    navigate(routes.userList)
  }

  return (
    <App isLoaded={true}>
      <Header label="User Register">
        <ActionLink to={routes.userList}
color="primary">
          Go back
        </ActionLink>
      </Header>

      <div className="w-1/2 py-5">
        <form onSubmit={registerRequest}>
          <div>
            <InputLabel id="name" type="text"
className="block w-full" required autoFocus
                        autoComplete="username"
                        placeholder={'New User
name...'}
                        value={payload.name}
                        onChange={e => setPayload({
...payload, name: e.target.value })}>
              {'Name'}
            </InputLabel>
          </div>

          <div className="flex">
            <div className="mt-4 pr-4 w-1/2">
              <SelectLabel id="gender"
className="block w-full" required
                           value={payload.gender}
                           onChange={e =>
setPayload({ ...payload, gender: e.target.value })}
                           options={[ // TODO Get
genders dynamically
                             {
                               value: '',
                               label: 'Select user
gender',
                               disabled: true,
                               hidden: true
                             },
                             { value: '1', label: 'Male'
},
                             { value: '0', label:
'Female' },
```

```
                        ]}>
                    {'Gender'}
                  </SelectLabel>
                </div>

                <div className="mt-4 w-1/2">
                  <InputLabel id="dob" type="date"
className="block w-full" required
                              autoComplete="bday"
                              value={payload.dob}
                              onChange={e =>
setPayload({ ...payload, dob: e.target.value })}>
                    {'Date of Birth'}
                  </InputLabel>
                </div>
              </div>

              <div className="mt-4 pr-4 w-1/2">
                <SelectLabel id="role"
className="block mt-1 w-full" required
                             value={payload.role}
                             onChange={e => setPayload({
...payload, role: e.target.value })}
                             options={[ // TODO Get roles
dynamically
                               {
                                 value: '',
                                 label: 'Select user role',
                                 disabled: true,
                                 hidden: true
                               },
                               { value: '1', label: 'User' },
                               { value: '2', label: 'Admin'
},
                             ]}>
                  {'Role'}
                </SelectLabel>
              </div>

              <div className="mt-4">
                <InputLabel id="image" type="file"
className="block w-full" required //TODO image
afterload
                            autoComplete="photo"
                            onChange={e => setPayload({
...payload, image: e.target.files[0] })}>
                  {'Image'}
                </InputLabel>
              </div>

              <div className="mt-4">
                <InputLabel id="email" type="email"
className="block w-full" required
                            autoComplete="email"
                            placeholder={'New email...'}
                            value={payload.email}
                            onChange={e => setPayload({
...payload, email: e.target.value })}>
                  {'Email'}
                </InputLabel>
              </div>

              <div className="mt-4">
                <InputLabel id="password"
type="password" className="block w-full" required
                            autoComplete="new-
password"
                            value={payload.password}
                            onChange={e => setPayload({
...payload, password: e.target.value })}>
                  {'Password'}
                </InputLabel>
              </div>

              {/*<div className="mt-4"> TODO pass
confirm*/}
              {/*    <InputLabel
id="password_confirmation" type="password"
className="block w-full" required*/}
              {/*                autoComplete="new-
password"*/}
              {/*
value={payload.password_confirmation}*/}
              {/*                onChange={e =>
setPayload({ ...payload, password_confirmation:
e.target.value })}>*/}
              {/*        {'Confirm Password'}*/}
              {/*    </InputLabel>*/}
              {/*</div>*/}

              <div className="flex items-center
justify-end mt-4">
                {/*TODO Login as*/}
                {/*<Button type="submit"
color="secondary" onClick={() =>
setSubmitType('RegisterLogin')}*/}
                {/*        className="ml-3">*/}
                {/*    {'Register & Login'}*/}
                {/*</Button>*/}

                <Button type="submit"
color="primary" className="ml-3">
                  {'Register'}
                </Button>
              </div>
            </form>
          </div>
        </App>
    )
}

export default Register
```

ДОДАТОК Б

Презентаційний матеріал

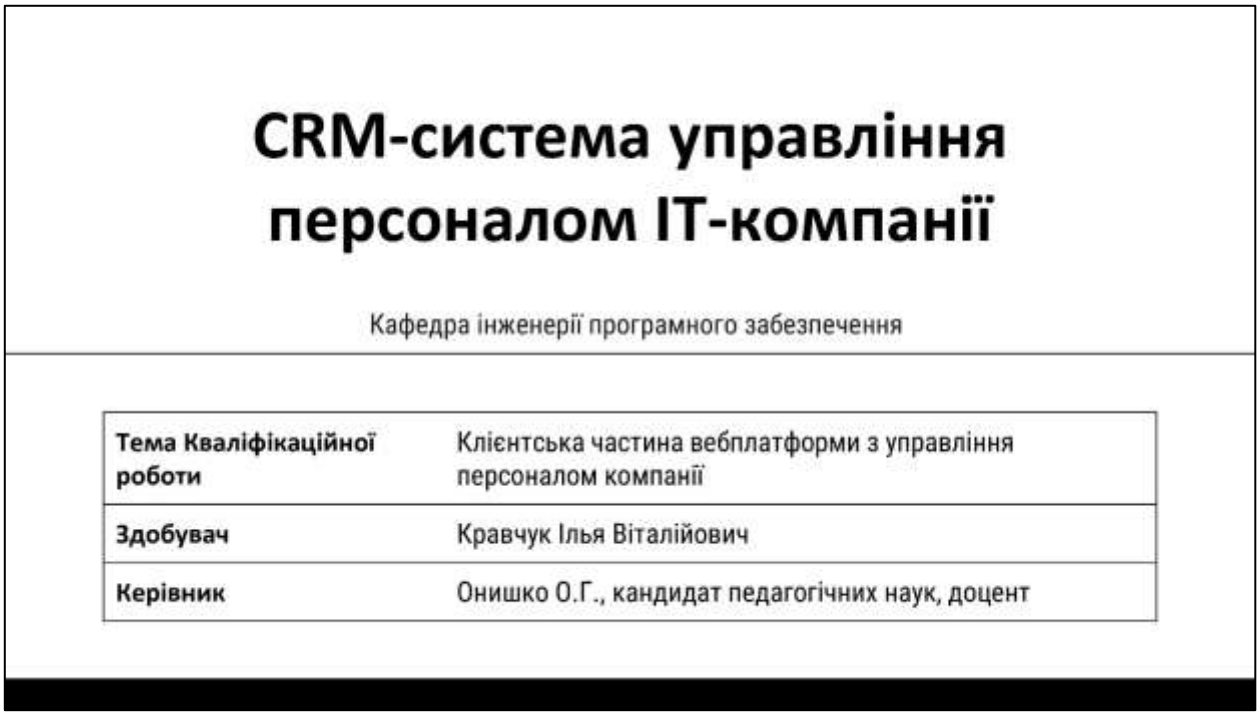


Рисунок Б.1 – Слайд №1

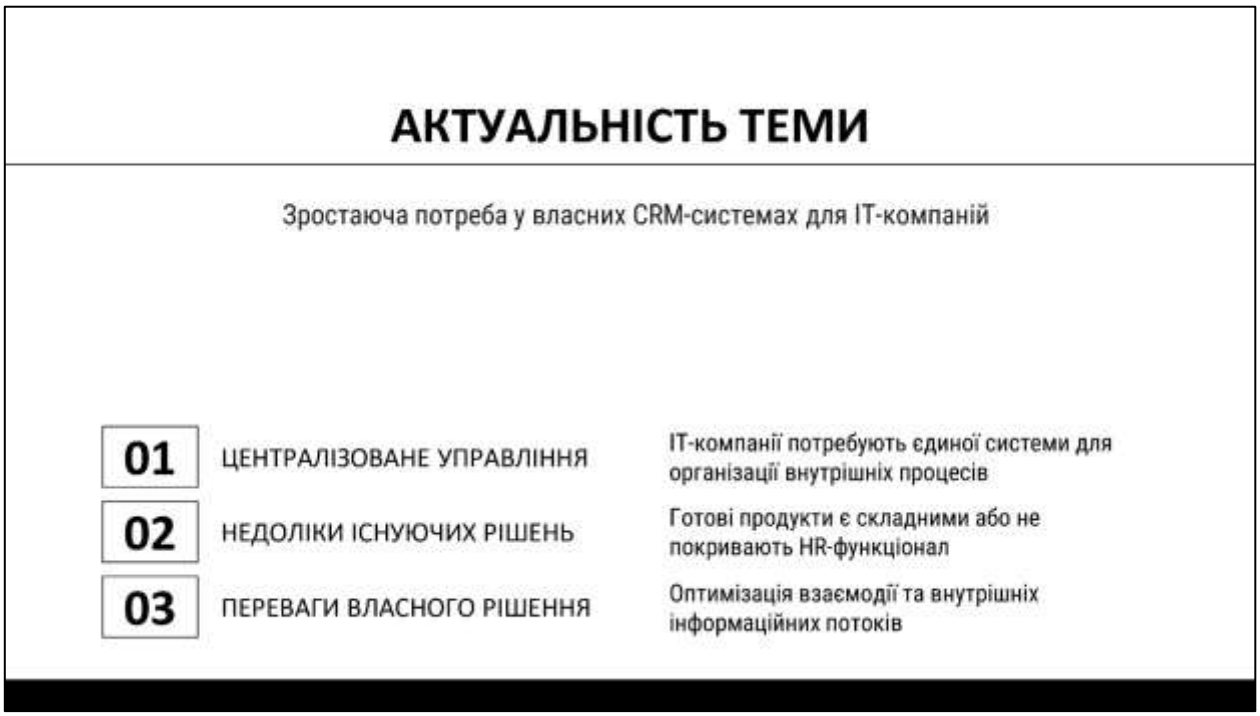


Рисунок Б.2 – Слайд №2

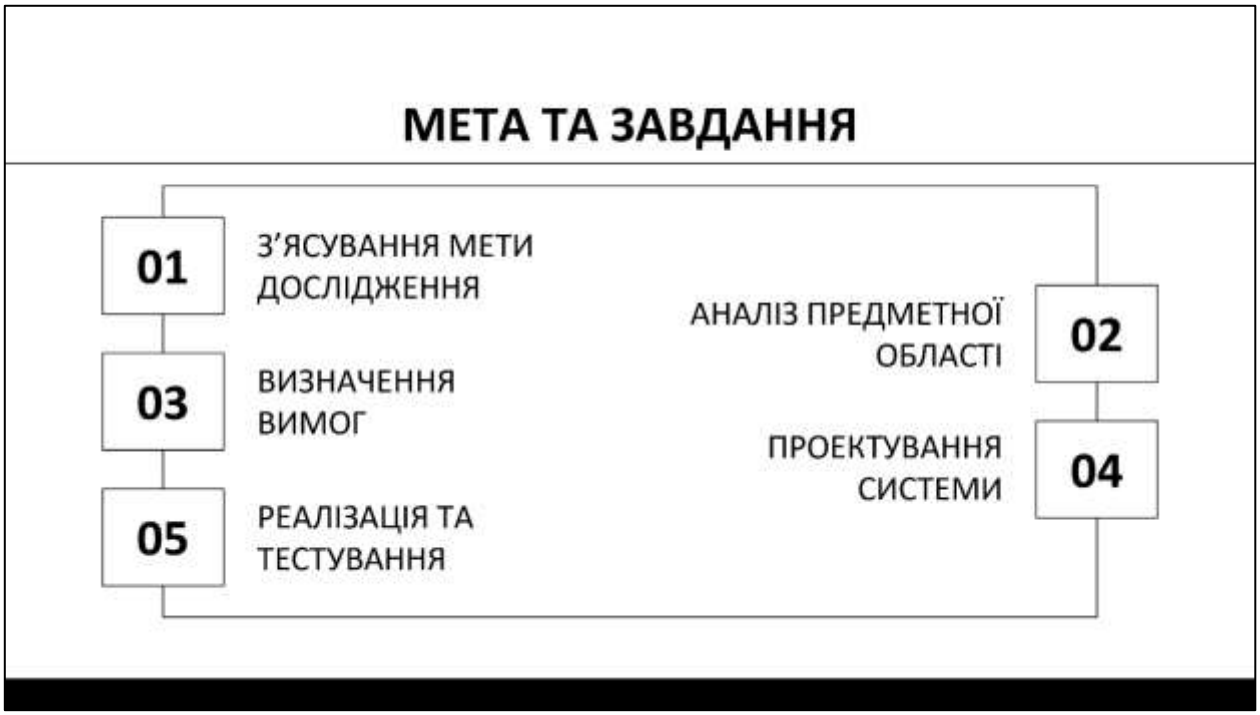


Рисунок Б.3 – Слайд №3

Рисунок Б.4 – Слайд №4

Рисунок Б.5 – Слайд №5

Рисунок Б.6 – Слайд №6

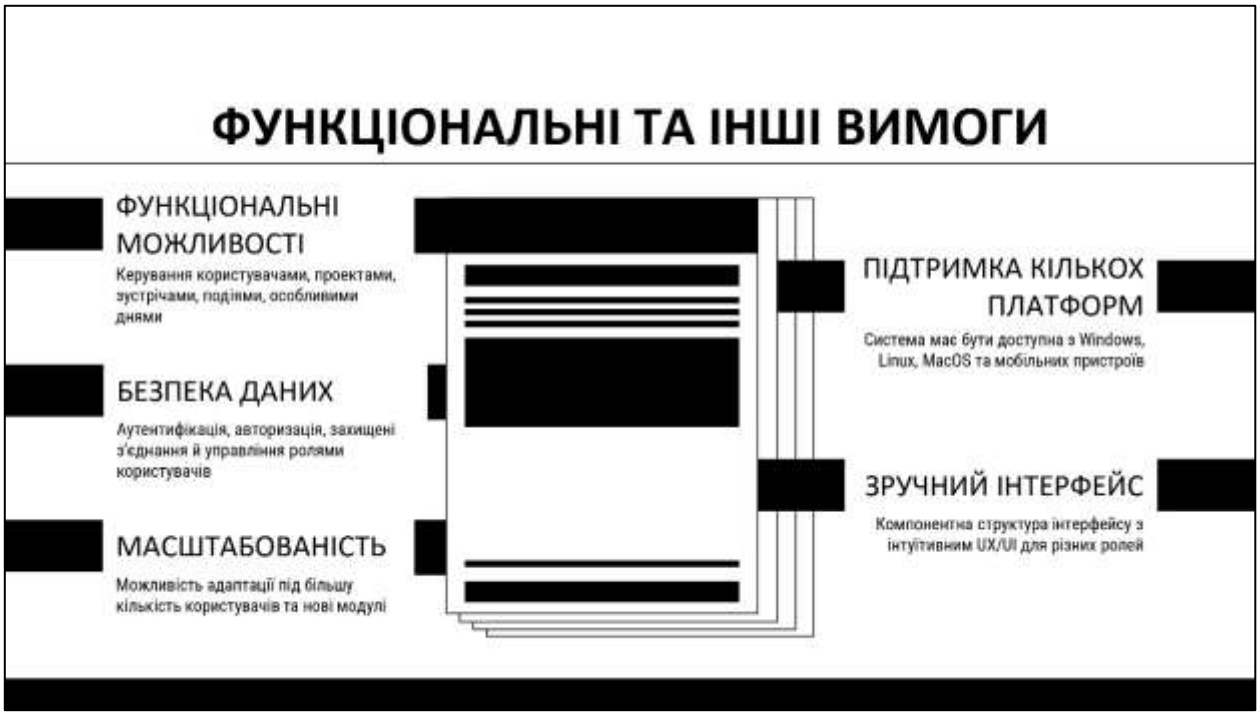


Рисунок Б.7 – Слайд №7

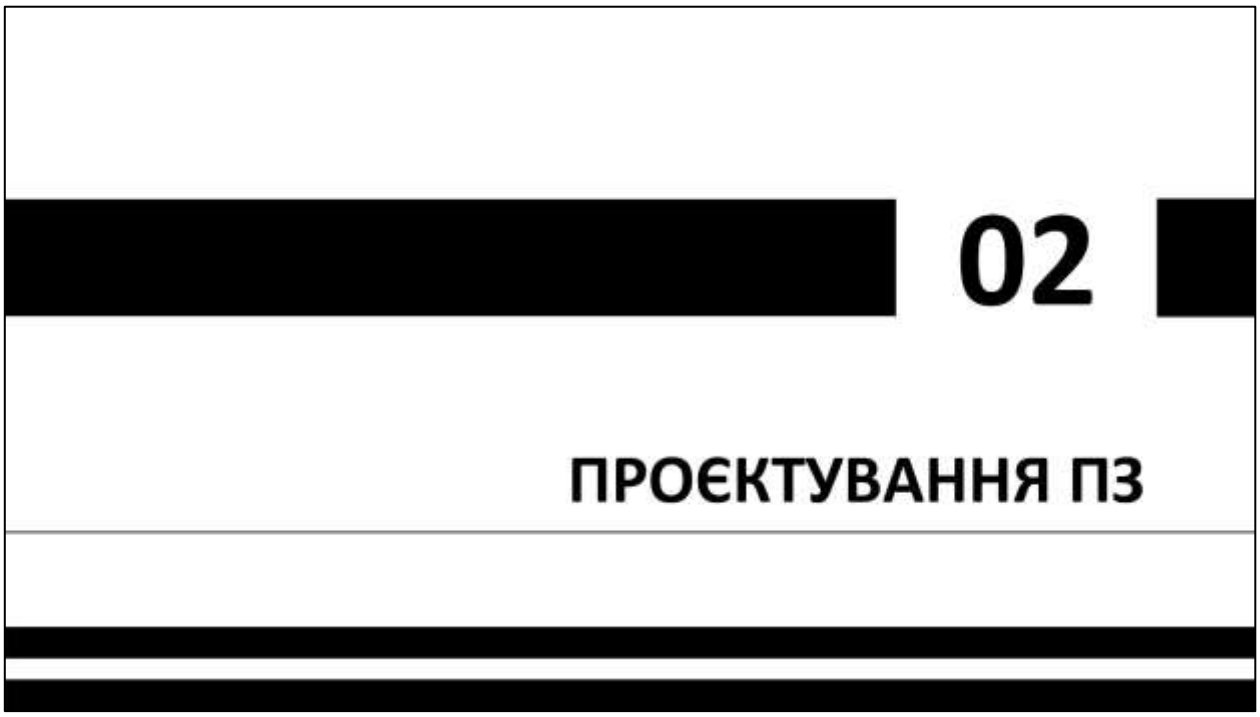


Рисунок Б.8 – Слайд №8

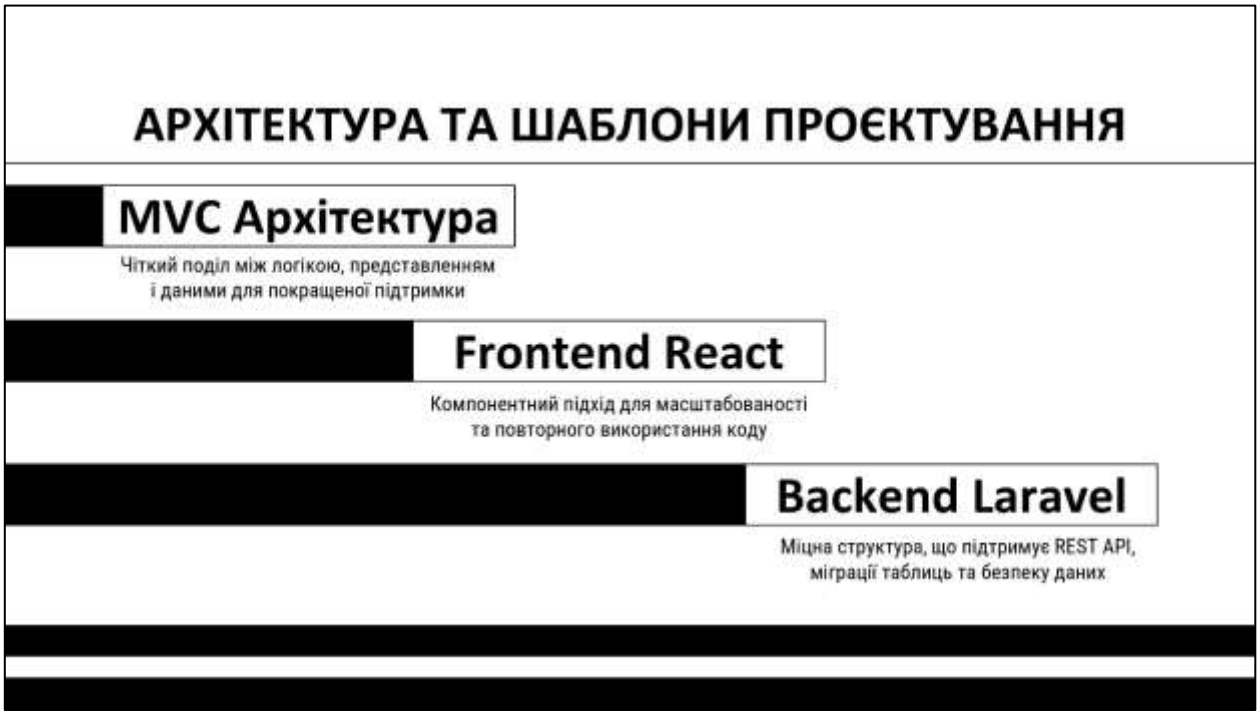


Рисунок Б.9 – Слайд №9

Рисунок Б.10 – Слайд №10

Рисунок Б.11 – Слайд №11

Рисунок Б.12 – Слайд №12

Рисунок Б.13 – Слайд №13

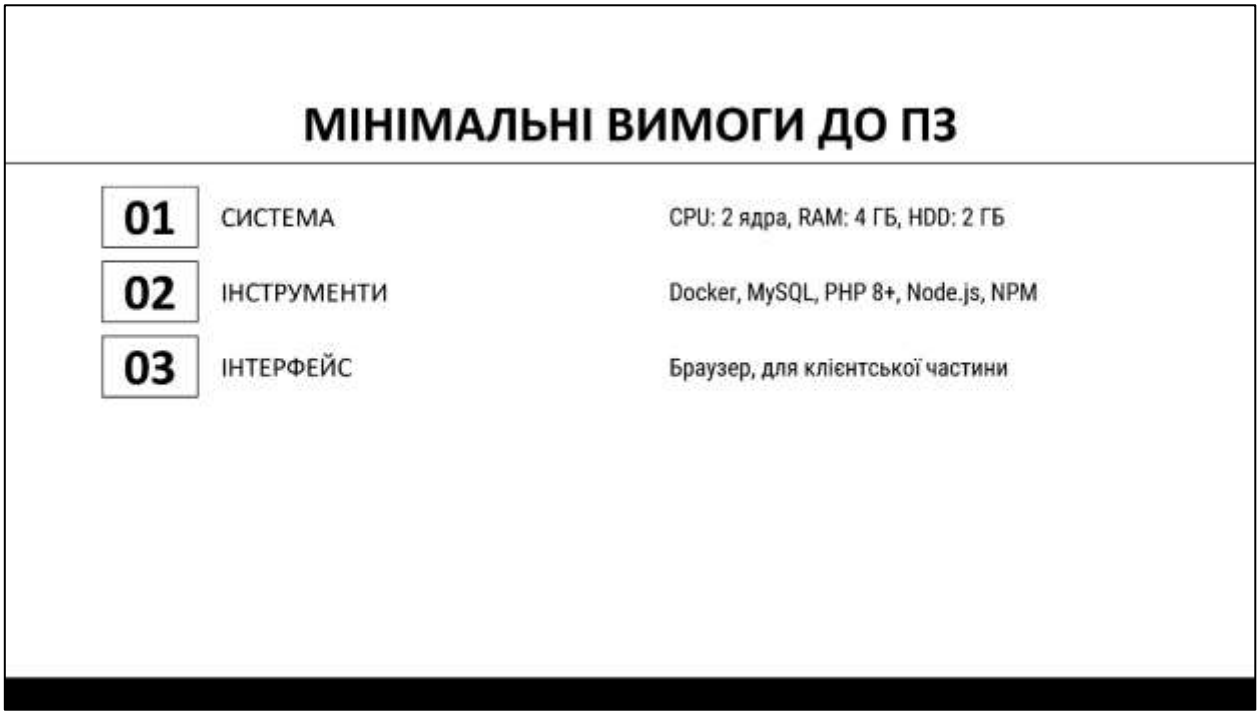


Рисунок Б.14 – Слайд №14

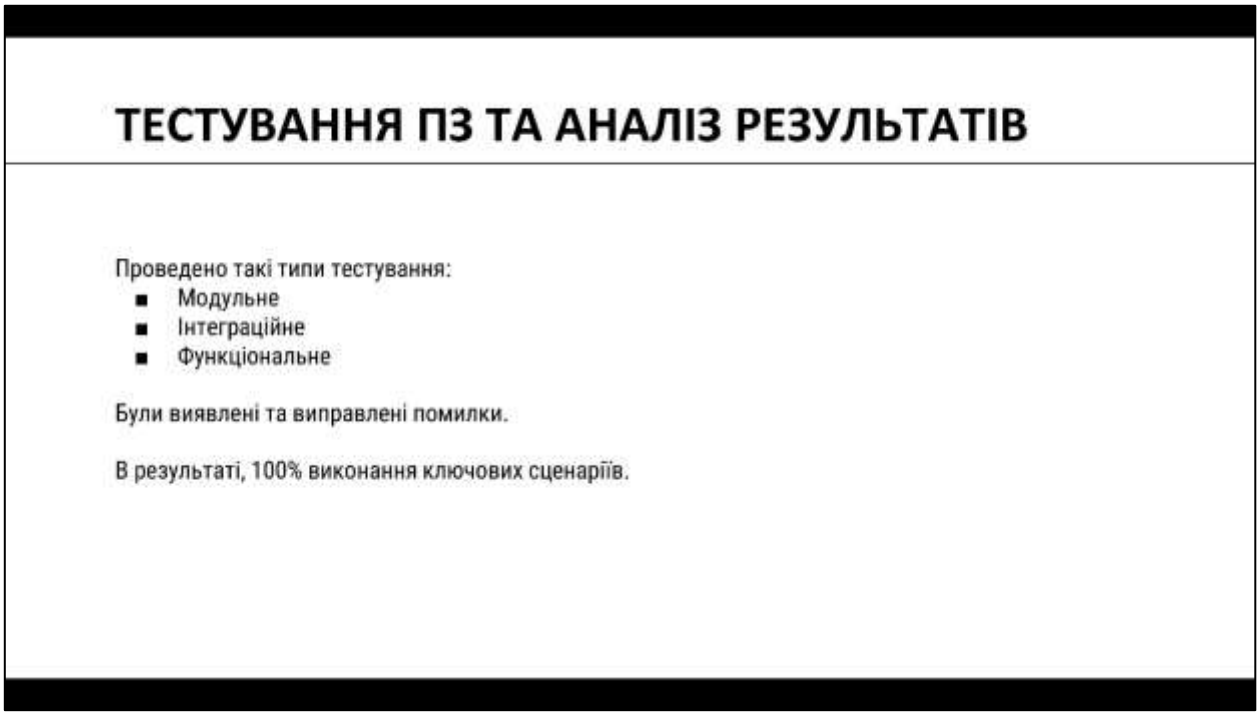
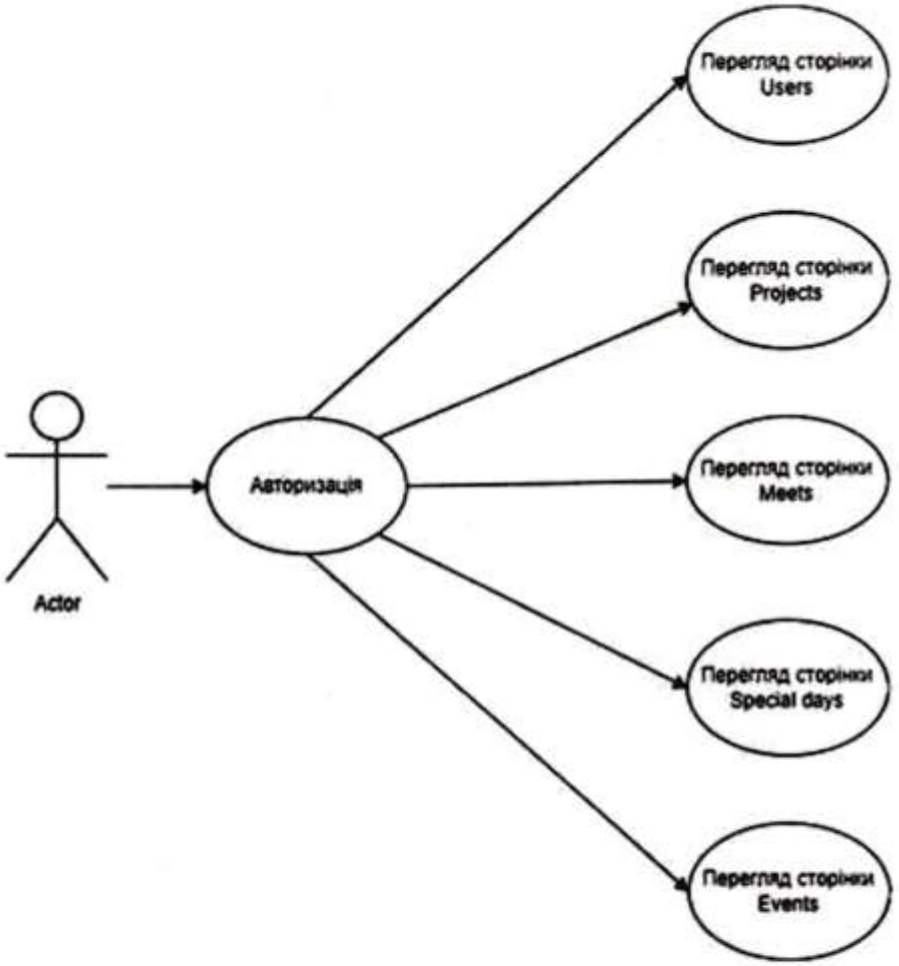


Рисунок Б.15 – Слайд №15

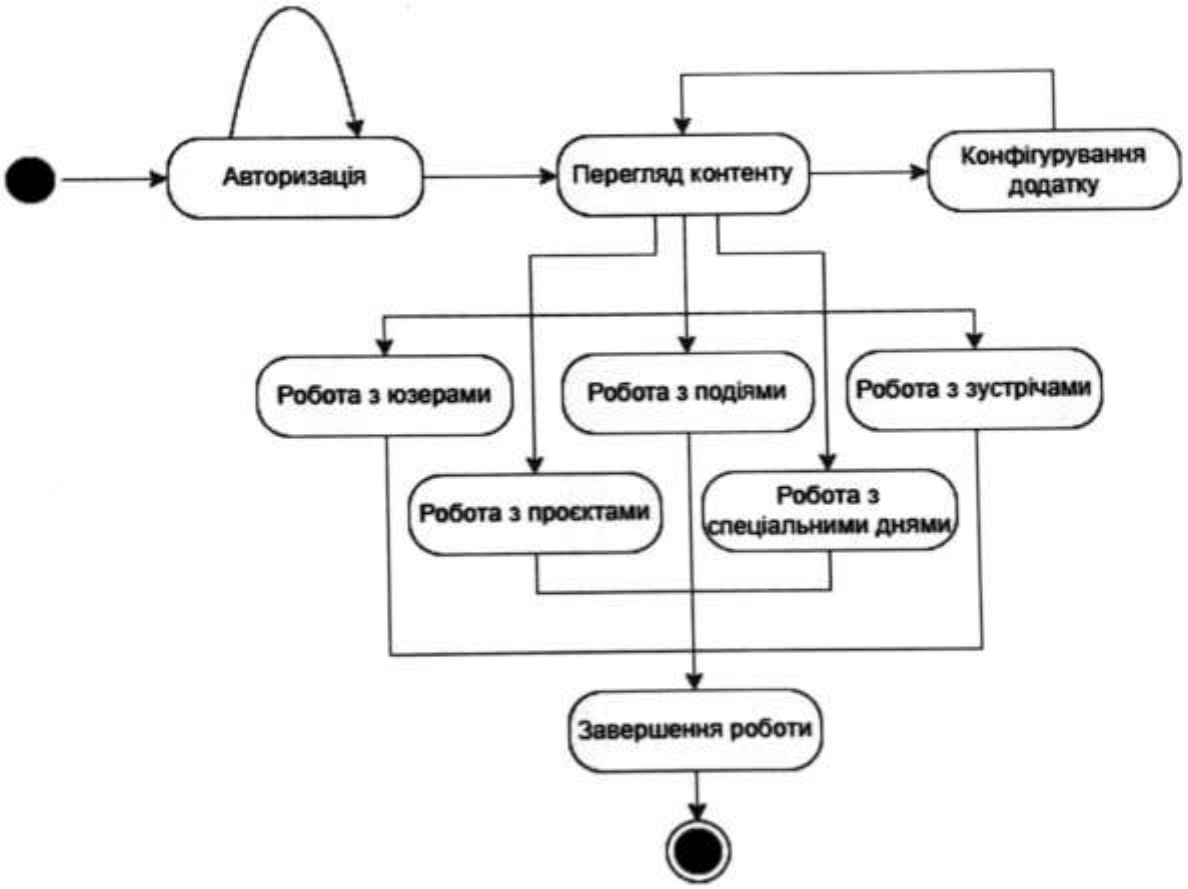
Рисунок Б.16 – Слайд №16

Рисунок Б.17 – Слайд №17

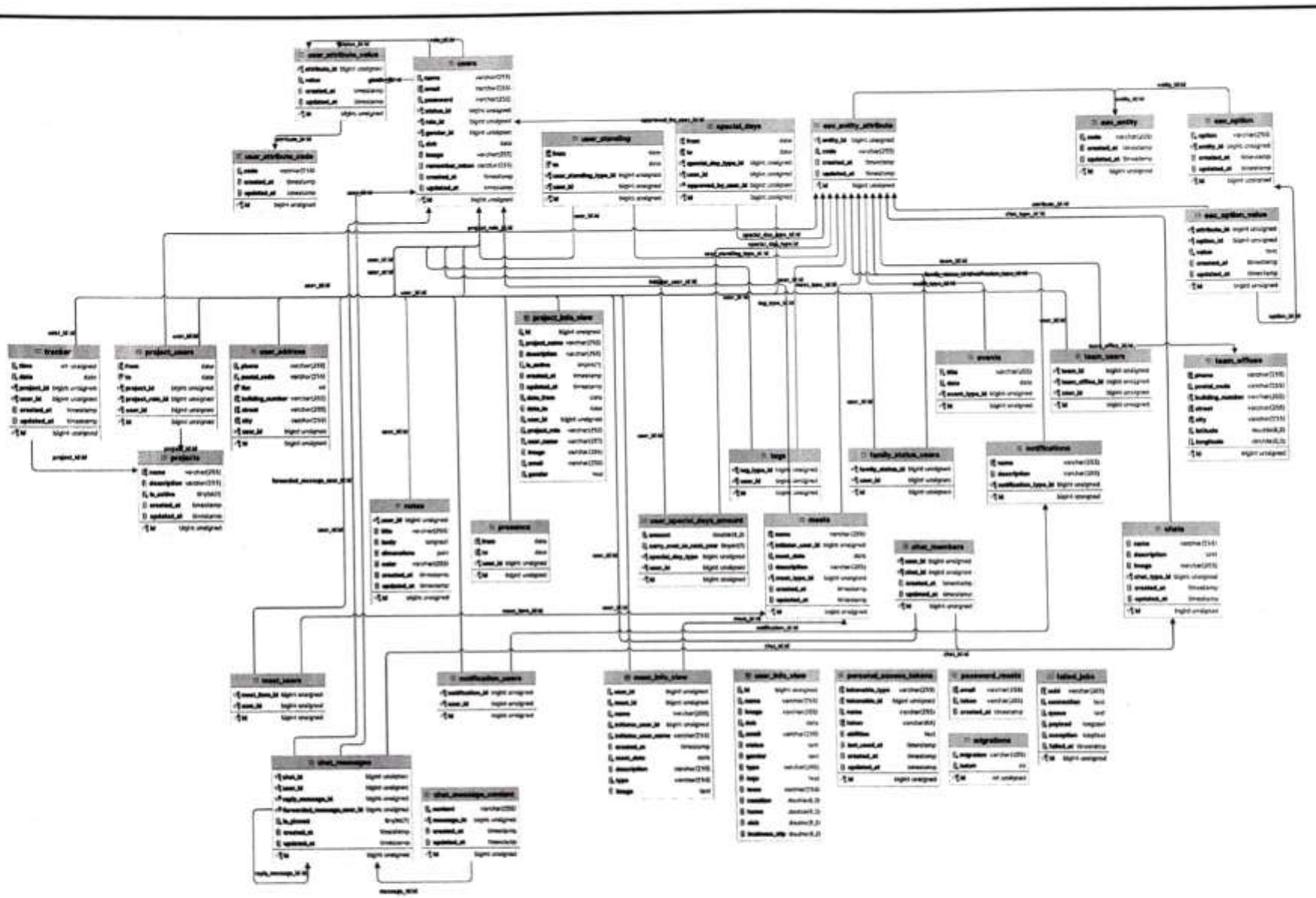
ГРАФІЧНА ЧАСТИНА



Змін.	Аркуш	№ докум.	Підпис	Дата	КПП.2201124.01.03.Е8		
Виконав		Крайчук І.В.		12.06	Діаграма використання, UseCase diagram	Літ	Аркуш 1 / Аркушів 5
Керівник		Онишко О.Г.		12.06			
Н. контр.		Яшина О.М.		12.06		ХНУ, ІПЗс-22-1	
Зав. каф.		Бедратюк Л.П.		12.06			



Змін.	Аркуш	№ докум.	Підпис	Дата	КПІІ.2201124.01.03.Е8		
Виконав		Кравчук І.В.		10.06	Діаграма станів, State diagram	Літ	Аркуш 2 / Аркушів 5
Керівник		Онищко О.Г.		10.06			
Н. контр.		Яшина О.М.		10.06		ХНУ, ІПЗс-22-1	
Зав. каф.		Бедратюк Л.П.		10.06			



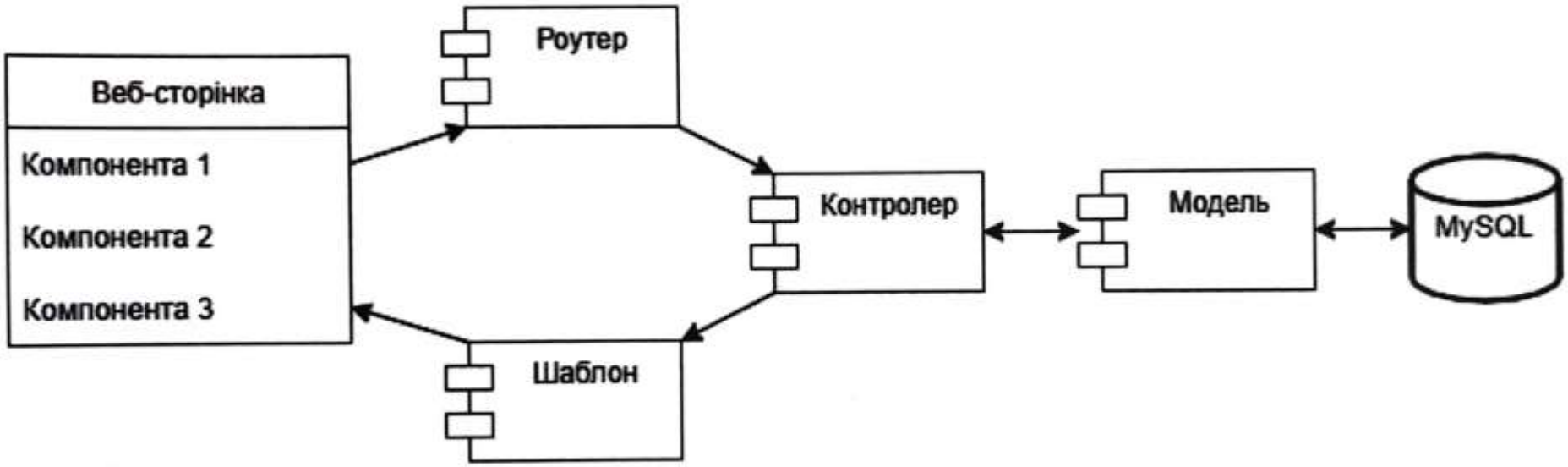
Зміл.	Аркуш	№ докум.	Підпис	Дата
Виконав		Кравчук І.В.		10.06
Керівник		Онишко О.Г.		10.06
Н. контр.		Яшина О.М.		10.06
Зав. каф.		Бедратюк Л.П.		10.06

КПП.2201124.01.03.Е8

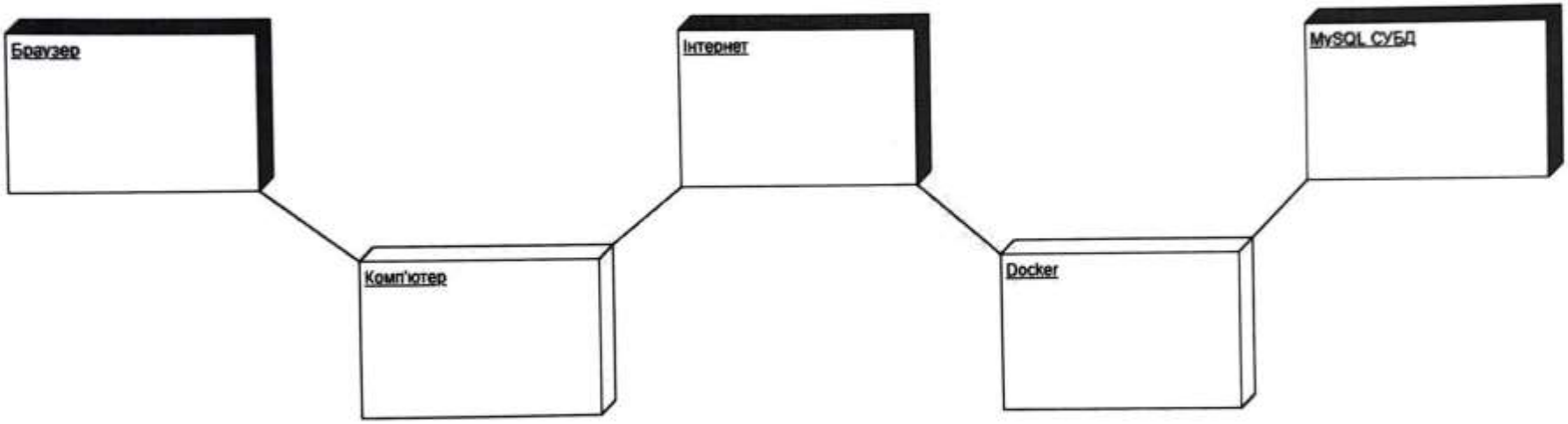
Діаграма класів,
Class diagram

Літ	Аркуш	Аркушів
	3	5

ХНУ, ІПЗс-22-1



Змін.	Аркуш	№ докум.	Підпис	Дата	КПП.2201124.01.03.Е8		
Виконав		Кравчук І.В.		10.06	Діаграма компонентів, Component diagram	Літ	Аркуш 4 / Аркушів 5
Керівник		Онишко О.Г.		10.06			
Н. контр.		Яшина О.М.		10.06		ХНУ, ІПЗс-22-1	
Зав. каф.		Бедратюк Л.П.		10.06			



Змін.	Аркуш	№ докум.	Підпис	Дата	КПІП.2201124.01.03.Е8		
Виконав		Кравчук І.В.		10.06	Діаграма розгортання, Deployment diagram	Літ	Аркуш 5 / Аркушів 5
Керівник		Онищко О.Г.		10.06			
Н. контр.		Яшина О.М.		10.06		ХНУ, ІПЗс-22-1	
Зав. каф.		Бедратюк Л.П.		10.06			

СУПРОВІДНІ ДОКУМЕНТИ

Завідувачу кафедри ІПЗ
проф. Бедратюку Л.П
студента групи ІПЗс-22-1
Кравчука І.В.
Прізвище, ініціали

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 (а). Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на наявність академічного плагіату оповіщений (а) 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 студента

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

освітнього ступеня «Бакалавр»

Дипломник Кравчук Ілья Віталійович

Тема Клієнтська частина вебплатформи з управління персоналом компанії

Спеціальність 121 – Інженерія програмного забезпечення

Обсяг кваліфікаційної роботи:

Кількість листів креслень 5 ; кількість сторінок записки 104

1. Короткий зміст пояснювальної записки та прийнятих рішень. У кваліфікаційній роботі розглянуто процес розробки клієнтської частини CRM-системи управління персоналом IT-компанії. У записці послідовно описано результати аналізу предметної області, оцінено наявні аналоги, сформульовано функціональні та нефункціональні вимоги до системи. Обґрунтовано вибір архітектурного підходу (MVC) та сучасного технологічного стеку, включно з фреймворками Laravel і React, а також інструментами Docker, MySQL, TailwindCSS. Здійснено проектування інтерфейсу та логіки роботи, реалізовано ключові модулі (керування персоналом, подіями, проектами, зустрічами, особливими днями), проведено тестування, яке підтвердило стабільну роботу веб-застосунку відповідно до поставлених вимог.

2. Висновок про відповідність роботи поставленому завданню. Кваліфікаційна робота повністю відповідає сформульованому завданню. Усі етапи — від аналізу предметної області до розробки, тестування та оформлення — виконано з урахуванням технічного завдання. Реалізований функціонал охоплює всі необхідні модулі, передбачені у вимогах, що підтверджує повну відповідність роботи поставленим цілям.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки та передових методів роботи. У вступі визначено актуальність, мету, завдання та предмет дослідження. У першому розділі проаналізовано предметну область, досліджено аналоги,

сформульовано вимоги до системи. У другому — спроєктовано архітектуру, базу даних, діаграми UML, інтерфейс. У третьому — реалізовано функціонал за допомогою Laravel, React, Docker, MySQL, проведено тестування. Робота базується на сучасних технологіях та підходах до розробки.

4. Позитивні сторони роботи. Тематика кваліфікаційної роботи є актуальною з огляду на зростаючі потреби IT-компаній у внутрішньому цифровому інструменті для ефективного управління персоналом. Робота вирізняється глибоким аналізом предметної області, порівнянням із популярними CRM-рішеннями, продуманим архітектурним підходом та використанням сучасних технологій (Laravel, React, Docker, MySQL). Особлива увага приділена зручності для кінцевого користувача, масштабованості системи й практичному впровадженню у реальні умови.

5. Негативні сторони роботи. Застосунок не реалізує розширену систему управління відділами та не має функціоналу бази знань для зберігання важливої документації.

6. Оцінка графічного оформлення та пояснювальної записки. Графічне оформлення виконано відповідно до теми та завдання кваліфікаційної роботи та подано у вигляді діаграм і рисунків. Пояснювальна записка оформлена згідно вимог чинних стандартів.

7. Відгук про кваліфікаційну роботу в цілому. Кваліфікаційна робота заслуговує позитивної оцінки. Матеріал пояснювальної записки структурований, послідовний, чіткий та простий, що дозволяє чітко зрозуміти викладений матеріал у рамках тематики проектування.

8. Інші зауваження. У деяких розділах обсяг тексту досить великий, можна розглянути варіант розбиття на менші логічні блоки.

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

10. РЕЦЕНЗЕНТ Кисоч Юрій, к.т.н., зав. кафедри кібербезпеки, ХНУ

9.06.25
Дата

Підпис

Anti-Plagiarism (UA) v-15.281 Educational

The maximum coincidence with one document 2.0%

Dictionaries check: en_US, ru_RU, ua_UA. **Errors in the documents: 9%**

	Document		Sum coincidence on the DB	
	Symbols	Lexemes	Symbols	Lexemes
ID: 243713 Title: БКР Клієнтська частина вебплатформи з управління персоналом компанії Added in a DB: 2025-06-05 Authors: Кравчук І.В. Heads: Онишко О.Г. канд. пед. наук, доцент Consultants: Opponents:	85941	1288	3110 (4%)	48 (4%)

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: Кравчук Ілья

Співавтор:

Назва: БКР Клієнтська частина вебплатформи з управління персоналом компанії

Науковий керівник:

Підрозділ: Кафедра інженерії програмного забезпечення

Коефіцієнт подібності 1: 1.7%

Коефіцієнт подібності 2: 0%

Мікропробіли: 0

Заміна букв: 15

Інтервали: 0

Білі знаки: 1

Дата створення звіту: 2025-06-04 14:26:15.0

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:

10.06.2025
Дата

Експерт

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: «Клієнтська частина вебплатформи з управління персоналом компанії»
Автор: Кравчук Ілья Віталійович
Освітня програма: Освітньо-професійна програма «Інженерія програмного забезпечення»
Спеціальність: 121 – Інженерія програмного забезпечення
Науковий керівник: Онишко Оксана Григорівна, канд. пед. 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 перевірки на плагіат Anti-Plagiarism виявлено схожість з деякими документами у частині загальновживаних обов язкових словосполучень у стандартних бланках, у структурі змісту, назвах розділів/підрозділів, рамках форм, у назвах та URL-адресах публікацій переліку джерел посилання:
2) запозичення. виявлені у тексті роботи є фрагментарними.
Максимальний обсяг запозичень. визначений системою Anti-Plagiarism, складає 2.0%, що з урахуванням наведених обгрунтувань, відповідає характеру теми і свідчить на користь кваліфікаційної роботи.

Дата 10.06.25

Завідувач кафедри  Леонід БЕДРАТЮК

Гарант освітньої програми  Леонід БЕДРАТЮК

Керівник кваліфікаційної роботи  Оксана ОНИШКО