НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри
___інформаційних систем і технологій
(назва кафедри)

_____ /_____Швиденко М.З.____/
(підпис) (ПІБ)

“___” _____ 2025 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА
на тему

«Інформаційна система для організації та моніторингу процесів у проєктному менеджменті»

Спеціальність 122 – «Комп’ютерні науки»

Гарант освітньої програми
____д.е.н., професор _____ _____ __Руденський Р.А.__
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Керівник бакалаврської кваліфікаційної роботи
_____ _____ _Золотуха Роман Андрійович_
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав _____ _____Ващук Олександр Володимирович___
(підпис) (ПІБ студента)

КИЇВ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри
_інформаційних систем і технологій___
(назва кафедри)

___Швиденко М.З., зав. каф., к.е.н. засл. проф
(науковий ступінь, вчене звання) (підпис) (ПІБ)
"___" _____ 2025 р.

З А В Д А Н Н Я
на виконання бакалаврської кваліфікаційної роботи студенту
Ващук Олександр Володимирович

Спеціальність 122 – «Комп'ютерні науки»

1. Тема бакалаврської кваліфікаційної роботи «Інформаційна система для організації та моніторингу процесів у проєктному менеджменті» затверджена наказом ректора НУБіП України від 16.12.2024 № 2246 "С"
2. Термін подання завершеної роботи на кафедру_____
(рік, місяць, число)
3. Вихідні дані до роботи: отримання відомостей про виконання призначених завдань, їх нюансів та комунікації між учасниками.
4. Перелік питань, що розглядаються:
   - Аналіз проблемної області
   - Моделювання предметної області
   - Проєктування програмної системи
   - Впровадження та експлуатація системи

Керівник кваліфікаційної роботи

д. філос. н. (спец. 122), ст. викл.		Золотуха Р.А.
(науковий ступінь та вчене звання)	(підпис)	(ПІБ)

Завдання прийняв до виконання		Ващук О.В.
	(підпис)	(ПІБ)

Дата отримання завдання "___"_____ 2025 р.

ЗМІСТ

ВСТУП

У сучасному цифровому середовищі ефективне управління проєктами має важливе значення для того, щоб організації підтримували продуктивність і ефективно дотримувались термінів. Оскільки команди ростуть, а проєкти стають складнішими, зростає потреба в спеціалізованих інструментах, які оптимізують планування, відстеження завдань і співпрацю.

Актуальність цього дослідження полягає у зростаючому попиті на гнучкі, масштабовані та зручні рішення для управління проєктами, які можна пристосувати до різноманітних організаційних потреб. Багато малих і середніх підприємств не мають доступу до дорогих платформ корпоративного рівня, тому важливо розробити альтернативу, яка б збалансувала функціональність і простоту.

Хоча існуючі рішення, такі як Jira та Asana, пропонують повну функціональність, вони часто дорогі, складні або їх важко налаштувати для конкретних потреб бізнесу. Це дослідження спрямоване на розробку інноваційної інформаційної системи для організації та моніторингу процесів управління проєктами.

Дослідження виконано з **метою** забезпечення покращення існуючих співпраці у сфері проєктного менеджменту розробивши інтуїтивно зрозуміле, економічно ефективне та ефективне рішення, призначене для команд, які шукають баланс між простотою та функціональністю.

Для досягнення визначеної мети потрібно поставити ряд завдань, виконання яких і приведе до бажаного результату. У рамках цього дослідження необхідно виконати аналіз сучасних програмних рішень у сфері управління проєктами, таких як Jira, Asana, Trello, щоб виявити їхні переваги, недоліки та межі застосування у різних організаційних контекстах. Важливо встановити особливості предметної області проєктного менеджменту в умовах малих і середніх підприємств, де ключовими вимогами є доступність, простота

використання та адаптивність системи. Слід проаналізувати потреби кінцевих користувачів, зокрема менеджерів проєктів і команд розробників, щодо функціоналу та зручності інтерфейсу. Після цього необхідно виявити основні функціональні та нефункціональні вимоги до майбутньої інформаційної системи, враховуючи специфіку командної роботи, контроль строків та моніторинг виконання завдань. На основі цього потрібно розробити архітектуру системи, моделі бази даних та алгоритми, що забезпечують ефективне створення проєктів, призначення завдань, комунікацію між учасниками та контроль за виконанням. Далі необхідно виконати програмну реалізацію інформаційної системи за допомогою технологій PHP, фреймворку Symfony та СУБД MySQL, інтегруючи сучасні веб-технології для створення зручного та інтуїтивно зрозумілого інтерфейсу. Завершальним етапом є проведення тестування, апробації програмного продукту в умовах, наближених до реального використання, а також оцінка його ефективності, стабільності роботи та відповідності початково поставленим вимогам.

Запропонована система розроблена, щоб задовольнити потребу в спрощеному та інтуїтивно зрозумілому інструменті управління проєктами. Це дозволяє користувачам:

- створювати проєкти та визначати ключові цілі;
- додавати завдання в проєкти, вказавши терміни виконання та рівні пріоритетності;
- призначати виконавців для завдань і відстежуйте прогрес у режимі реального часу;
- спілкуватися в рамках завдань, щоб забезпечити чітку та ефективну співпрацю;
- контролювати загальний хід проєкту, забезпечуючи своєчасне завершення робіт.

Технологічна основа цієї системи базується на надійній комбінації PHP і фреймворку Symfony, що забезпечує масштабованість і зручність обслуговування на сервері. MySQL служить основною базою даних для

управління структурованими даними, тоді як сучасні веб-технології використовуються для інтуїтивно зрозумілого та швидкого реагування на інтерфейс. Процес розробки дотримується методології Agile, що дозволяє безперервно вдосконалюватись на основі ітераційного зворотного зв’язку та тестування. Такий підхід гарантує, що система розвивається відповідно до потреб користувачів, зберігаючи гнучкість і адаптивність.

Об’єкт дослідження – процеси управління проєктами в малих та середніх організаціях, зокрема їх організація, контроль, моніторинг виконання завдань та взаємодія між учасниками проєктної команди.

Предмет дослідження – методи, інструменти та програмні засоби розробки інформаційної системи для організації та моніторингу процесів управління проєктами з використанням веб-технологій, що забезпечують автоматизацію ключових функцій проєктного менеджменту.

Розглядаючи зростаючий попит на ефективні та доступні інструменти управління проєктами, це дослідження представляє практичне та адаптоване рішення, яке може принести користь різним командам і організаціям. Отримані результати сприяють розвитку інформаційних систем і створюють основу для подальшого прогресу в розробці програмного забезпечення для управління проєктами.

Основна частина роботи складається з чотирьох розділів. У першому розділі здійснено аналіз проблемної області, де розглянуто сучасні інструменти управління проєктами, їх функціональні можливості та обмеження. Другий розділ присвячено моделюванню предметної області, зокрема опису функціональних вимог, побудові UML-діаграм і формуванню логічної структури системи. У третьому розділі описано проєктування програмної системи, що включає архітектуру, розробку бази даних та реалізацію ключових функцій. Четвертий розділ містить інформацію про впровадження та тестування системи, результати її апробації та оцінку відповідності поставленим вимогам.

1 АНАЛІЗ ПРОБЛЕМНОЇ ОБЛАСТІ

1.1 Опис предметної області

У сучасному бізнес-середовищі управління проєктами стає дедалі складнішим через зростання віддаленої роботи, багатофункціональних команд і різноманітних вимог до проєктів. Організації часто стикаються з проблемами координації зусиль, відстеження прогресу та забезпечення ефективного спілкування між членами команди. Традиційні методи управління проєктами, такі як електронні таблиці та електронна пошта, можуть призвести до дезорганізації, пропуску термінів і відсутності підзвітності, що зрештою перешкоджає загальній продуктивності. Як наслідок, існує нагальна потреба в надійних інформаційних системах, які можуть полегшити організацію та моніторинг процесів, пов’язаних із проєктом.

Існуючі програмні рішення для керування проєктами, такі як Jira та Asana, пропонують широкий спектр функцій, призначених для вирішення цих завдань. Однак вони часто мають обмеження, які можуть не відповідати потребам кожної організації. Наприклад, багато з цих інструментів розроблено для великих команд і проєктів, що призводить до непотрібної складності для невеликих організацій або конкретних типів проєктів. Крім того, вартість ліцензування та крутий процес навчання, пов’язані з деякими з цих платформ, можуть бути непомірно високими, особливо для малих та середніх підприємств (МСП). Це створює прогалину на ринку для більш доступних і адаптованих рішень, які можуть задовольнити організаційні розміри та складність проєктів [1].

Крім того, інтеграція комунікаційних інструментів у системи управління проєктами має важливе значення для оптимізації співпраці та мінімізації непорозумінь. Багатьом поточним рішенням або бракує адекватних комунікаційних функцій, або вони неефективно інтегруються з популярними комунікаційними платформами, через що командам важко узгоджуватися з

цілями проєкту. Таким чином, є можливість розробити інформаційну систему, яка не тільки спрощує керування завданнями, але й покращує комунікацію команди, що в кінцевому підсумку сприяє більш ефективному середовищу управління проєктами.

Виявляючи ці виклики, пропоноване дослідження спрямоване на створення комплексної інформаційної системи, адаптованої до потреб організацій, які прагнуть покращити свої процеси управління проєктами. Ця система використовуватиме сучасні технології для забезпечення зручного інтерфейсу, надійної функціональності та безперебійних комунікаційних функцій, що зрештою усуває недоліки існуючих рішень і допомагає командам ефективно досягати цілей проєкту.

Предметом цієї дисертації є розробка та впровадження інформаційної системи, спеціально розробленої для організації та моніторингу процесів управління проєктами. Ця область об’єднує принципи розробки програмного забезпечення, методології управління проєктами та інформаційні системи для створення інструменту, який підвищує ефективність і результативність діяльності, пов’язаної з проєктом.

За своєю суттю управління проєктами передбачає планування, виконання, моніторинг і закриття проєктів, що вимагає систематичного підходу для забезпечення досягнення цілей у межах визначених обмежень, таких як час, бюджет і ресурси. Ефективне управління проєктами вимагає чіткої комунікації, співпраці між членами команди та здатності адаптуватися до мінливих обставин. Ця інформаційна система спрямована на вирішення цих фундаментальних аспектів, надаючи користувачам централізовану платформу для керування різними компонентами проєкту, включаючи призначення завдань, відстеження прогресу, керування термінами та командну співпрацю [2].

Система включатиме основні функції, такі як створення проєкту, організація завдань, встановлення кінцевих термінів і розподіл ролей, щоб сприяти чіткій підзвітності. Крім того, інтегровані інструменти спілкування сприятимуть співпраці, дозволяючи членам команди ділитися оновленнями,

обговорювати виклики та надавати відгуки в режимі реального часу. Використання сучасних технологій, таких як PHP, Symfony та MySQL, гарантує, що система буде масштабованою, безпечною та зручною для користувача, що зробить її доступною для широкого кола користувачів, від невеликих команд до великих організацій.

Крім того, інформаційна система спиратиметься на встановлені методології управління проєктами, такі як Agile та Waterfall, щоб гарантувати, що вона відповідає різноманітним потребам різних проєктів. Завдяки використанню найкращих практик і адаптації до різних робочих процесів система прагне забезпечити гнучке рішення, яке можна адаптувати до конкретних організаційних вимог.

Таким чином, предметна область цієї дисертації обертається навколо перетину управління проєктами та інформаційних систем, зосереджуючись на розробці програмного рішення, яке покращує організацію, моніторинг і виконання проєктів. За допомогою цього дослідження мета полягає в тому, щоб зробити внесок у вдосконалення інструментів управління проєктами, зрештою допомагаючи організаціям підвищити продуктивність і ефективніше досягати своїх цілей.

Детальний опис класів та атрибутів предметної області наведено у таблиці 1.1.

Таблиця 1.1

Опис атрибутів класів предметної області

Клас предметної області	**Атрибут**	**Опис**
Проєкт	Назва проєкту	Унікальна назва, що ідентифікує проєкт.
	Опис	Короткий опис цілей і завдань проєкту.
	Дата початку	Дата, коли проєкт розпочинається.
	Дата закінчення	Дата, до якої проєкт має бути завершено.
	Статус	Поточний стан проєкту (в процесі, завершено, відкладено тощо).
Завдання	Назва завдання	Унікальна назва завдання в рамках проєкту.
	Опис	Детальний опис завдання та його цілей.
	Відповідальна особа	Ім’я користувача або групи, відповідальної за виконання завдання.
	Дата виконання	Дата, до якої завдання має бути виконано.
	Пріоритет	Визначення важливості завдання (високий, середній, низький).
Користувач	Ім’я	Ім’я користувача, що використовує систему.
	Роль	Роль користувача в проєкті (менеджер, виконавець, спостерігач тощо).
	Контактна інформація	Дані для зв’язку (електронна пошта, телефон).

Ця таблиця допомагає зрозуміти основні класи предметної області, їх атрибути та їх значення в контексті системи управління проєктами.

1.2 Аналіз наявних інформаційних технологій

Розглянемо існуючі рішення предметної області.

Jira — це універсальний інструмент управління проєктами та відстеження проблем, створений Atlassian, широко визнаний своєю ефективністю в розробці програмного забезпечення та в інших секторах. Спочатку зосереджена на відстеженні помилок, Jira перетворилася на

комплексну платформу, яка підтримує гнучке управління проєктами, дозволяючи командам ефективно планувати, відстежувати та керувати своїми проєктами.

Однією з видатних особливостей Jira є підтримка гнучких методологій, таких як Scrum і Kanban. Платформа дозволяє командам створювати візуальні дошки, які спрощують керування завданнями, спрощуючи моніторинг прогресу та швидке виявлення вузьких місць. Дошки Scrum допомагають у плануванні спринту, тоді як дошки Kanban ідеально підходять для керування безперервними процесами доставки, сприяючи більш організованому робочому процесу [1].

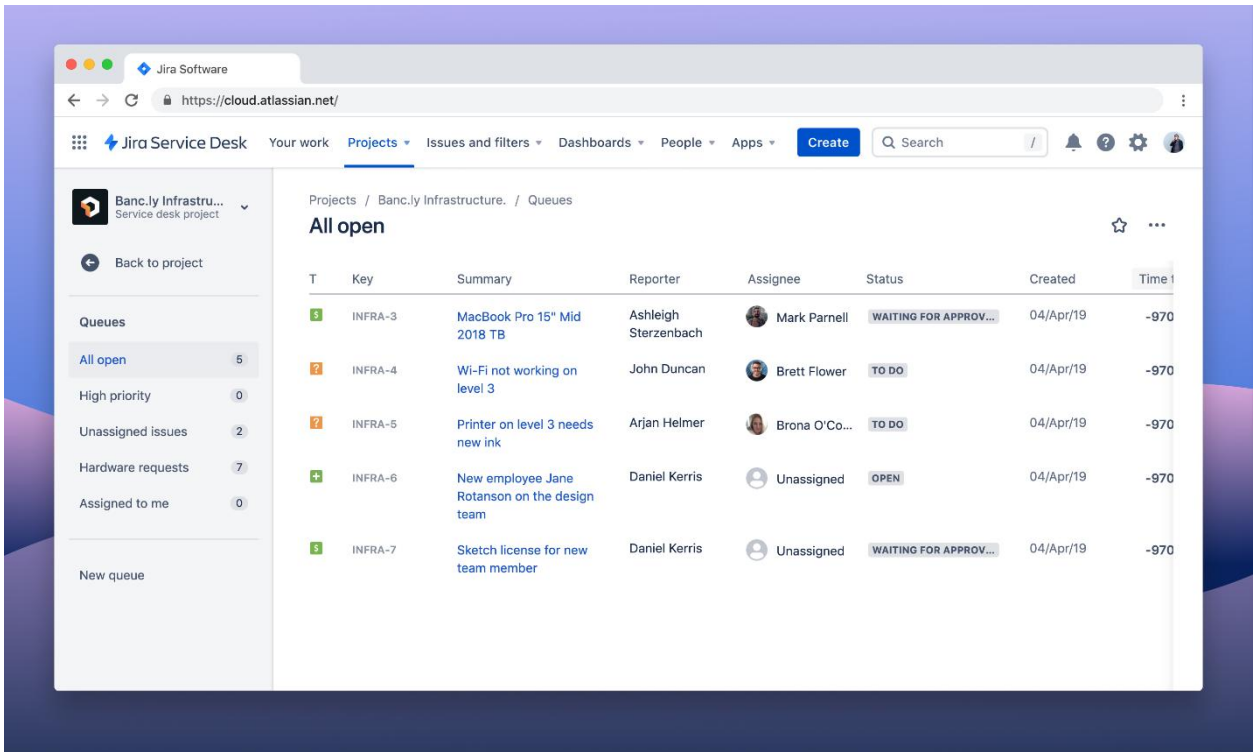

Рис. 1.1 Інформаційна система «Jira»

За своєю суттю Jira функціонує як система відстеження проблем, де користувачі можуть створювати, призначати та визначати пріоритети різних проблем, включаючи помилки, завдання та історії користувачів. Кожен випуск можна збагатити детальною інформацією, такою як описи, вкладення, коментарі та спеціальні поля, сприяючи вичерпній документації та співпраці між членами команди.

Налаштування є ключовим аспектом Jira, особливо щодо робочих процесів. Команди можуть адаптувати свої процеси, визначаючи конкретні

статуси робочого процесу, переходи та умови відповідно до своїх унікальних вимог. Ця гнучкість значно підвищує ефективність і допомагає оптимізувати управління проєктами.

На додаток до цих функцій, Jira пропонує надійні інструменти звітності та аналітики, які дозволяють командам ефективно контролювати свою продуктивність і прогрес. Користувачі можуть генерувати різноманітні звіти, наприклад про час спринту та швидкість, надаючи цінну інформацію, яка полегшує прийняття рішень на основі даних і постійні вдосконалення.

Можливості інтеграції Jira ще більше підвищують його корисність, оскільки він легко підключається до численних програм і інструментів сторонніх розробників, включаючи Atlassian Confluence та Bitbucket, а також різні інструменти CI/CD. Цей взаємозв’язок сприяє співпраці та оптимізує робочі процеси на різних платформах, полегшуючи спільну роботу команд [1].

Керування користувачами та дозволи також є життєво важливими компонентами Jira. Адміністратори можуть контролювати доступ до проєктів і інформації, гарантуючи, що члени команди мають відповідні дозволи, зберігаючи при цьому безпеку та захист конфіденційних даних. Ця функція заохочує співпрацю всередині команди без шкоди для цілісності даних.

Jira, доступна як у хмарному, так і в локальному варіантах розгортання, пропонує організаціям гнучкість відповідно до їхніх конкретних потреб. Хмарна версія (Jira Cloud) забезпечує легкий доступ, тоді як локальні рішення (Jira Server і Data Center) надають організаціям більше контролю над своїми даними.

Загалом Jira стала провідним інструментом управління проєктами, особливо в розробці програмного забезпечення, завдяки своєму надійному набору функцій і адаптованості до різних методологій. Оснащуючи команди необхідними інструментами для планування, відстеження та співпраці над проєктами, Jira підвищує організаційну ефективність, прозорість і результати проєкту. Його широкі можливості налаштування та можливості інтеграції роблять його придатним для різноманітних галузей, крім розробки програмного

забезпечення, дозволяючи командам адаптувати платформу до своїх конкретних вимог.

Розглянемо інше програмне рішення на ринку.

Asana — це популярний інструмент для управління проєктами та співпраці, який допомагає командам ефективно організовувати, відстежувати та керувати своєю роботою. Asana, запущена в 2012 році, отримала широке поширення в різних галузях завдяки своєму зручному інтерфейсу, гнучкості та надійним функціям, які задовольняють різноманітні потреби управління проєктами.

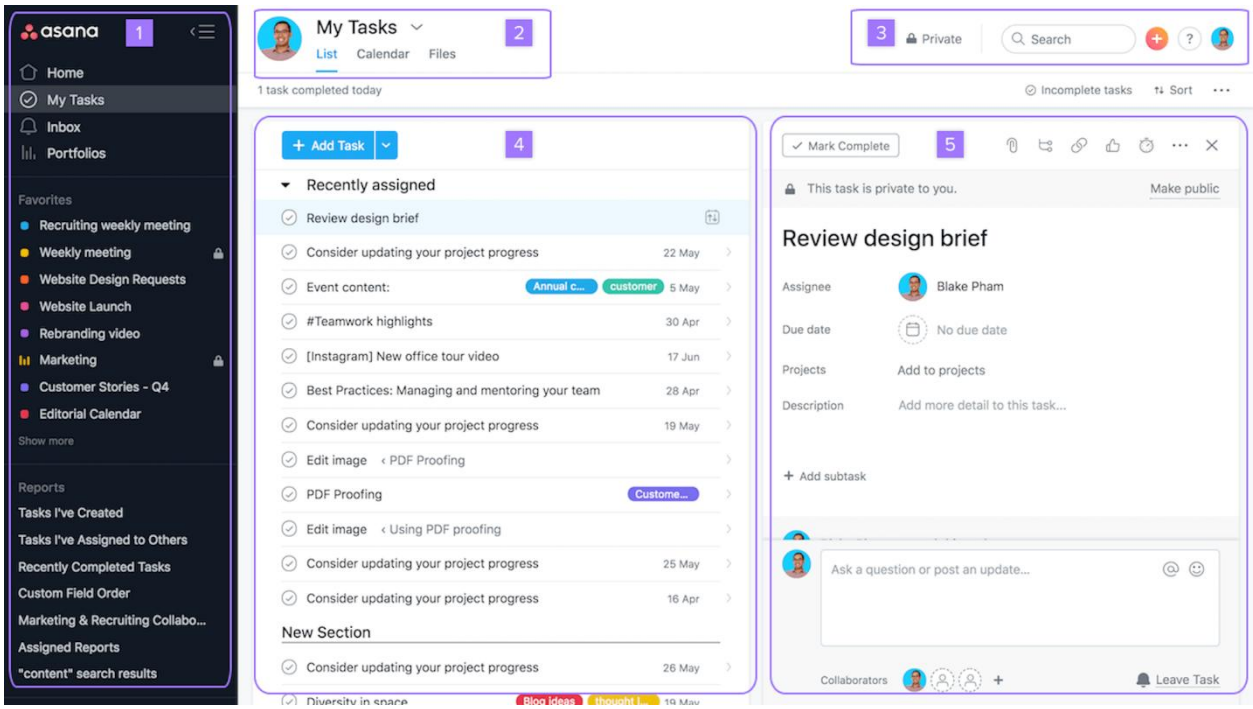

Рис. 1.2 Інформаційна система «Asana»

Однією з ключових сильних сторін Asana є її здатність полегшувати керування завданнями. Користувачі можуть створювати завдання, призначати їх членам команди, встановлювати крайні терміни та додавати докладні описи для забезпечення ясності. Платформа дозволяє класифікувати завдання за проєктами, дозволяючи командам підтримувати організовану структуру своєї роботи. Крім того, Asana підтримує використання підзавдань, які допомагають розбивати великі завдання на керовані компоненти, полегшуючи відстеження прогресу та розподіл обов'язків [2].

Asana пропонує різні режими перегляду для керування завданнями та проєктами, зокрема список, дошку, календар і часову шкалу. Перегляд списку забезпечує простий спосіб перегляду завдань у лінійному форматі, тоді як перегляд дошки використовує макет у стилі Канбан, що дозволяє командам візуалізувати свій робочий процес і переміщувати завдання на різних етапах. Перегляд календаря допомагає командам планувати та контролювати терміни, тоді як перегляд шкали часу пропонує перспективу, схожу на діаграму Ганта, забезпечуючи чіткий огляд графіків проєкту та залежностей.

Співпраця є центральною особливістю Asana, оскільки вона дозволяє членам команди спілкуватися безпосередньо в рамках завдань. Користувачі можуть залишати коментарі, вкладати файли та позначати колег, щоб полегшити обговорення та обмінюватися оновленнями. Ця співпраця в режимі реального часу підвищує прозорість і гарантує, що кожен залишається в курсі розвитку проєкту. Крім того, Asana інтегрується з численними програмами сторонніх розробників, включаючи Slack, Google Drive і Microsoft Teams, що дозволяє командам оптимізувати свої робочі процеси та підвищити продуктивність.

Asana також надає потужні інструменти звітності та відстеження, які допомагають командам контролювати свій прогрес і ефективність. Користувачі можуть створювати різноманітні звіти, як-от показники виконання проєктів, статуси завдань і робоче навантаження команди, що дозволяє приймати рішення на основі даних і постійно вдосконалюватися. Інформаційні панелі, що налаштовуються, дозволяють командам візуалізувати ключові показники та миттєво оцінювати продуктивність свого проєкту.

Зручний дизайн платформи та настроювані функції роблять її придатною для команд будь-якого розміру, від невеликих стартапів до великих підприємств. Asana дозволяє організаціям адаптувати свої робочі процеси, налаштовувати власні поля та створювати шаблони для повторюваних проєктів, гарантуючи, що інструмент відповідає їхнім конкретним потребам.

Підсумовуючи, Asana — це комплексне рішення для управління проєктами, яке дозволяє командам ефективно співпрацювати та ефективно

керувати своєю роботою. Завдяки надійним функціям керування завданнями, гнучким переглядам і можливостям інтеграції Asana підвищує продуктивність і прозорість, що робить її цінним інструментом для організацій, які прагнуть покращити свої процеси управління проєктами. Незалежно від того, чи використовується ви для простого відстеження завдань чи управління складними проєктами, Asana надає інструменти, необхідні командам для досягнення своїх цілей і досягнення успіху.

1.3 Постановка завдання

Основною метою цього проєкту є розробка інформаційної системи для організації та моніторингу процесів у управлінні проєктами. Ця система повинна надати командам ефективні інструменти для планування, виконання та контролю проєктів, що дозволить знизити ризики, пов'язані з недотриманням термінів та перевитратами ресурсів. Щоб досягти цього, необхідно створити надійну базу даних для зберігання важливої інформації, такої як деталі проєктів, завдання, терміни виконання, відповідальні особи та статуси виконання.

Розроблювана програмна система має на меті забезпечити користувачів швидким і легким доступом до важливої інформації про:

- структуру та організацію проєктів, включаючи визначення цілей і завдань;
- управління завданнями, включаючи їх розподіл, терміни виконання та статуси;
- моніторинг прогресу проєктів, що дозволяє вчасно виявляти проблеми та коригувати плани;
- можливість ефективного спілкування між членами команди та зацікавленими сторонами.

Система матиме інтуїтивно зрозумілий інтерфейс, що дозволить користувачам легко орієнтуватися в меню для управління проєктами та завданнями. Крім того, система включатиме функціональні можливості для

генерації звітів про статус проєктів, продуктивність команди та виконання завдань, що дозволить користувачам приймати обґрунтовані рішення для оптимізації процесів управління проєктами.

Впроваджуючи цю інформаційну систему, організації матимуть можливість оптимізувати свої робочі процеси, покращити спілкування в команді та, зрештою, досягати кращих результатів у реалізації проєктів.

1.4 Функціональні та нефункціональні вимоги

Таблиця 1.2

Функціональні вимоги

1	Система повинна дозволяти створення та керування обліковими записами користувачів з різними ролями (наприклад, керівник проєкту, член команди, зацікавлена сторона) і дозволами.
2	Користувачі повинні мати можливість створювати нові проєкти із зазначенням назви, опису, дати початку, дати завершення та цілей.
3	Система повинна дозволяти створювати, призначати та визначати пріоритети завдань у проєктах з можливістю додавання опису, термінів і файлів.

Продовження таблиці 1.2

4	Система має підтримувати комунікацію між членами команди – коментарі до завдань, обмін оновленнями, сповіщення про зміни.
5	Користувачі повинні мати можливість створювати звіти щодо продуктивності проєкту, команди та виконання завдань.
6	Система повинна підтримувати інтеграцію зі сторонніми додатками (календарі, хмарні сервіси зберігання).
7	Користувачі повинні мати можливість пошуку проєктів, завдань або документів з фільтрацією за статусом, пріоритетом або термінами.
8	Користувачі повинні мати можливість відстежувати хід виконання завдань і проєктів за допомогою статусів (не розпочато, виконується, завершено) та візуальних індикаторів (Kanban, прогрес-бари).

Таблиця 1.3

Нефункціональні вимоги

1	Інтерфейс системи має бути інтуїтивно зрозумілим та зручним для користувачів без необхідності тривалого навчання.
2	Система повинна ефективно обробляти велику кількість користувачів та проєктів одночасно без втрати продуктивності.
3	Архітектура повинна забезпечувати легке масштабування при збільшенні навантаження.
4	Система повинна забезпечувати безпеку даних: автентифікація, авторизація, шифрування, захист від витоку.
5	Висока доступність і надійність системи, мінімізація простоїв.
6	Простота обслуговування і оновлення, з можливістю безперешкодного додавання нових функцій.
7	Кросбраузерна та кросплатформна сумісність – підтримка ПК, планшетів, смартфонів.

Задовольняючи як функціональні, так і нефункціональні вимоги, інформаційна система прагне забезпечити комплексне та ефективне рішення для організації та моніторингу процесів управління проєктами, зрештою підвищуючи продуктивність команди та співпрацю.

1.5 Вимоги до інтерфейсу користувача

Інтерфейс користувача (UI) інформаційної системи, призначеної для організації та моніторингу процесів управління проєктами, має важливе значення для надання інтуїтивно зрозумілого та ефективного досвіду користувача. Щоб досягти цього, інтерфейс має надавати пріоритет інтуїтивно зрозумілій навігації, що дозволяє користувачам легко отримувати доступ до різних розділів системи, таких як проєкти, завдання, звіти та налаштування. Чуйний дизайн має вирішальне значення, гарантуючи плавну адаптацію

інтерфейсу користувача до різних розмірів екрана та пристроїв, включаючи настільні ПК, планшети та смартфони.

Таблиця 1.4

Вимоги до інтерфейсу користувача

№	Вимога до інтерфейсу користувача
1	Інтуїтивна навігація для легкого доступу до розділів: проєкти, завдання, звіти, налаштування.
2	Адаптивний (чуйний) дизайн для коректного відображення на ПК, планшетах і смартфонах.
3	Узгоджене компонування: єдина колірна схема, типографіка, стилі кнопок на всіх сторінках.
4	Зручні форми введення з підказками, чіткими мітками та повідомленнями про помилки для створення/редагування завдань і проєктів.
5	Чітка візуальна ієрархія: виділення важливої інформації (терміни, пріоритети) через заголовки, кольори, шрифти.
6	Візуальні елементи: діаграми Ганта, Kanban-дошки, індикатори прогресу для відображення статусу завдань/проєктів.
7	Потужний пошук і фільтрація за ключовими словами, статусом, виконавцем тощо.
8	Інтегровані інструменти співпраці: коментарі, згадки, сповіщення в деталях завдань.

Зосереджуючись на цих аспектах, інтерфейс користувача інформаційної системи покращить зручність використання, оптимізує процеси управління проєктами та, зрештою, підвищить задоволеність користувачів і продуктивність.

1.6 Висновки до 1 розділу

У першому розділі було здійснено всебічний аналіз предметної області, пов’язаної з організацією та моніторингом процесів управління проєктами. Розглянуто основні характеристики галузі, її потреби та специфіку, що дозволило чітко визначити актуальність створення інформаційної системи для автоматизації управлінських процесів.

Проведений огляд існуючих програмних рішень виявив як їх сильні сторони, так і недоліки, серед яких – обмежена функціональність, складний інтерфейс або недостатня адаптивність. Це дозволило виділити вимоги до майбутньої системи, що краще відповідатиме потребам користувачів.

У результаті було сформульовано постановку задачі, яка передбачає розробку інформаційної системи, що забезпечуватиме зручне управління проєктами, завданнями, звітністю та взаємодією між учасниками.

На основі поставленої мети визначено набір функціональних та нефункціональних вимог, зокрема щодо надійності, продуктивності, масштабованості та безпеки системи.

Окремо розглянуто вимоги до інтерфейсу користувача, що охоплюють інтуїтивну навігацію, адаптивний дизайн, доступність, підтримку колаборації та персоналізацію. Ці аспекти мають на меті забезпечити позитивний користувацький досвід і сприяти ефективній роботі з системою.

Таким чином, проведений аналіз проблемної області та визначені вимоги створюють ґрунтовну основу для проєктування та реалізації інформаційної системи управління проєктами, що буде розглянуто в наступних розділах.

2 МОДЕЛЮВАННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

2.1 Об’єктне та функціональне моделювання

Уніфікована мова моделювання (UML) — це стандартизована мова моделювання, яка широко використовується в розробці програмного забезпечення для візуалізації, специфікації, конструювання та документування різноманітних артефактів програмної системи. Розроблений у 1990-х роках, UML пропонує повний набір графічних позначень і вказівок, які допомагають спростити моделювання складних програмних систем та їх компонентів. Це

полегшує розробникам, аналітикам і зацікавленим сторонам зрозуміти структуру та поведінку системи.

Однією з ключових переваг UML є його універсальність, що дозволяє застосовувати його протягом усього життєвого циклу розробки програмного забезпечення — від збору вимог до проєктування та впровадження. Використовуючи UML, команди можуть створювати чіткі та точні представлення системи, що значно покращує спілкування та співпрацю між зацікавленими сторонами [5].

UML включає різноманітні типи діаграм, кожна з яких відповідає різним аспектам моделювання системи. Ці діаграми зазвичай поділяють на дві основні категорії: структурні діаграми та діаграми поведінки. Структурні діаграми підкреслюють статичні елементи системи, надаючи розуміння її організації та зв’язків. Наприклад, діаграми класів ілюструють класи системи, включаючи їхні атрибути, методи та зв’язки, служачи основним інструментом для об’єктно-орієнтованого проєктування. З іншого боку, діаграми компонентів зображують організацію та залежності між програмними компонентами, дозволяючи розробникам візуалізувати архітектуру системи. Крім того, діаграми розгортання представляють фізичний розподіл артефактів між апаратними ресурсами, пояснюючи, як розгортаються програмні компоненти.

Навпаки, діаграми поведінки зосереджуються на динамічних аспектах системи, фіксуючи її взаємодії та процеси. Діаграми варіантів використання особливо корисні для зображення взаємодії між користувачами (акторами) і системою, підкреслюючи функціональні вимоги та різні варіанти використання, які підтримує система. Діаграми послідовності описують упорядковану за часом взаємодію між об’єктами, детально описуючи обмін повідомленнями під час конкретних випадків використання або сценаріїв. Між тим, діаграми діяльності ілюструють робочий процес системи, демонструючи послідовність дій і моменти прийняття рішень, щоб полегшити розуміння складних процесів.

Крім розробки програмного забезпечення, принципи UML можна застосовувати в різних сферах, таких як моделювання бізнес-процесів, системна

інженерія та проєктування баз даних. Його стандартизація означає, що UML широко визнаний і підтримується численними інструментами моделювання, що спрощує його прийняття та впровадження в проєкти для команд.

Підсумовуючи, UML служить потужною та адаптованою мовою моделювання, яка допомагає візуалізувати та документувати програмні системи. Надаючи багатий набір діаграм і нотацій, UML покращує спілкування зацікавлених сторін, покращує процес проєктування та, зрештою, сприяє успішній розробці складних програмних додатків.

2.1.1 Діаграма прецедентів. Діаграми варіантів використання є ключовим компонентом уніфікованої мови моделювання (UML), який візуально відображає взаємодію між користувачами (акторами) і системою. Вони забезпечують загальне уявлення про функціональність системи та ілюструють, як користувачі взаємодіють із різними функціями, що робить їх важливим інструментом для визначення функціональних вимог у розробці програмного забезпечення [5].

В основі діаграми варіантів використання знаходяться актори та варіанти використання. Актори представляють зовнішні об’єкти, які взаємодіють із системою, до яких можуть входити користувачі, інші системи або зовнішні пристрої. Кожен актор пов’язаний з одним або декількома варіантами використання, які представляють певні функції або завдання, які система може виконувати у відповідь на дії актора. Варіанти використання

описують цілі або завдання, яких актор прагне досягти під час взаємодії з системою, фіксуючи поведінку системи з точки зору користувача.

Відносини між акторами та варіантами використання зображені лініями, що з'єднують їх на діаграмі. Ці відносини можуть приймати різні форми, наприклад:

- асоціація: проста лінія, що з'єднує актора з варіантом використання, вказуючи, що актор взаємодіє з цим варіантом використання;
- include: зв'язок, де варіант використання включає поведінку іншого варіанту використання, представляючи залежність між ними. Це корисно для розбиття складних процесів на простіші компоненти, які можна багаторазово використовувати;
- extend: зв'язок, який дозволяє варіанту використання розширювати поведінку іншого варіанту використання за певних умов. Це зазвичай використовується для представлення необов'язкової або умовної функціональності.

Діаграми варіантів використання служать кільком важливим цілям у процесі розробки програмного забезпечення. Вони допомагають зацікавленим сторонам, включаючи розробників, керівників проєктів і клієнтів, отримати чітке розуміння функціональності системи та взаємодії з користувачем. Візуально представляючи вимоги, діаграми варіантів використання полегшують спілкування між членами команди та забезпечують основу для подальшого аналізу та проєктування [6].

Спроєктована діаграма прецедентів представлена на рис.2.1.

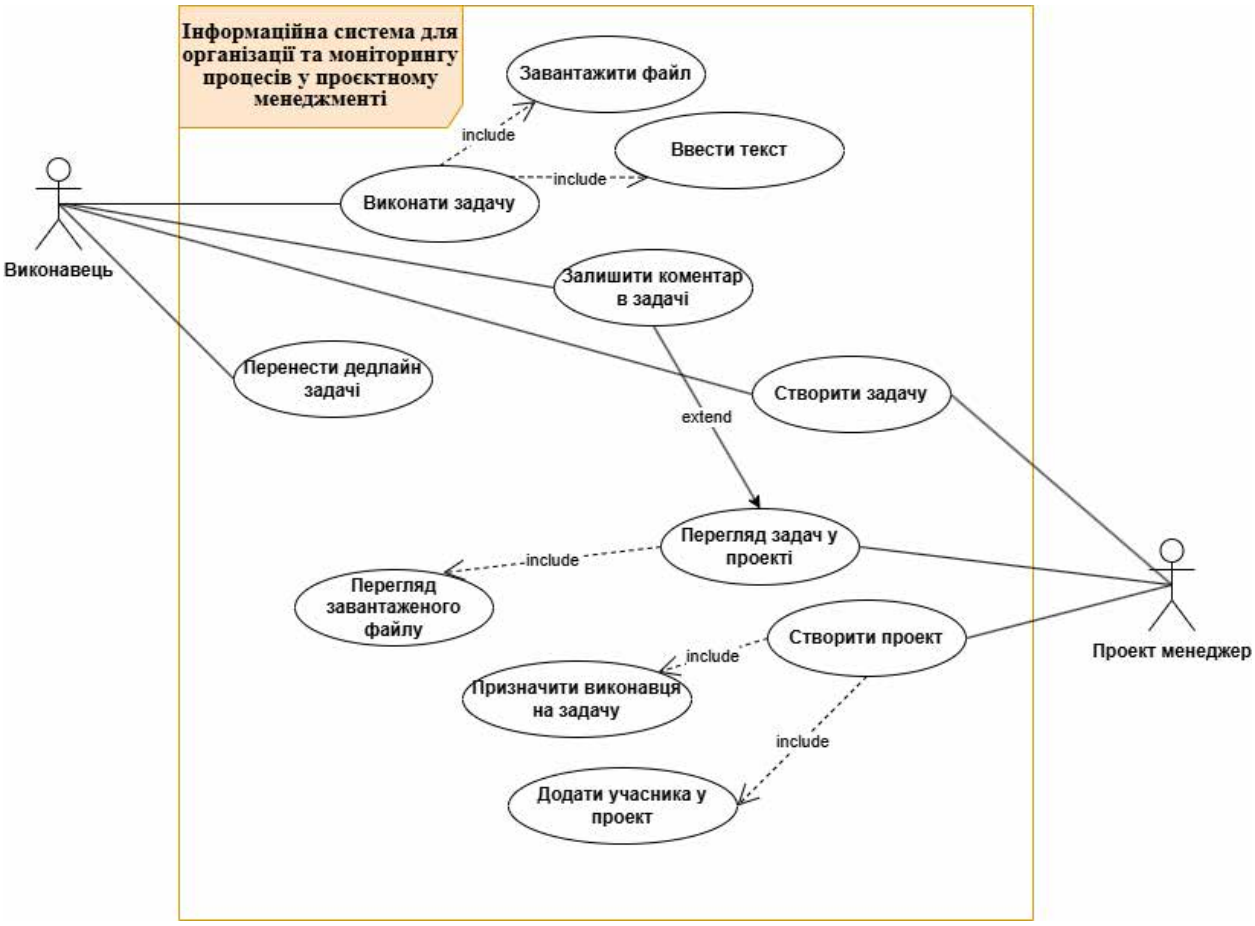

Рис. 2.1 Діаграма прецедентів

Створена діаграма прецедентів містить акторів:

- “Виконавець”;
- “Проєкт менеджер”.

Актор «Виконавець» включає такі прецеденти:

- виконати задачу;
- завантажити файл;
- ввести текст;
- залишити коментар в задачі;
- перенести дедлайн;
- створити задачу.

Актор «Проєкт менеджер» включає такі прецеденти:

- створити задачу;
- створити проєкт;
- призначити виконавця на задачу;

- додати учасника в проєкт;
- перегляд завантаженого файлу.

Прецеденти певним чином залежать одне від одного.

Прецедент “Виконати задачу” доповнюється іншими такими прецедентами як “Завантажити файл” та “Ввести текст”.

Розглянемо детальніше вищеописані прецеденти.

Сценарій використання: «Виконати задачу».

Актор: Виконавець

Передумова: Виконавець увійшов в інформаційну систему і має доступ до проєкту та завдання, призначеного йому. Завдання створено і доступне до виконання.

Основний сценарій:

1. виконавець переходить на інформаційну панель інформаційної системи, де він може переглядати всі завдання, призначені йому в різних проєктах;
2. виконавець визначає завдання, яке йому потрібно виконати, зі свого списку завдань. Кожне завдання відображається з відповідними деталями, включаючи назву завдання, опис, термін виконання та рівень пріоритету;
3. виконавець клацає назву завдання, щоб отримати доступ до деталей завдання, які включають детальний опис, будь-які вкладені файли та коментарі інших членів команди або керівника проєкту;
4. виконавець переглядає вимоги до завдання та будь-які надані додаткові ресурси, щоб переконатися, що він розуміє, що потрібно зробити;
5. виконавець починає роботу над завданням. За потреби вони можуть завантажувати відповідні документи, додавати коментарі або співпрацювати з іншими членами команди. Якщо виникнуть запитання чи потрібні роз’яснення, виконавець може зв’язатися з

іншими членами команди безпосередньо через функцію обміну повідомленнями системи;

6. після виконання завдання виконавець повертається на сторінку деталей завдання та позначає завдання як «Виконане», вибравши відповідну опцію в системі;
7. виконавець також може мати можливість надати відгук або коментарі щодо виконання завдання, що може бути корисним для подальших довідок або оцінок;
8. після позначення завдання як виконаного виконавець може додавати будь-які остаточні документи або результати, пов’язані із завданням, для перегляду керівником проєкту;
9. система оновлює статус завдання на «Виконано», і сповіщення можуть надсилатися керівнику проєкту та відповідним зацікавленим сторонам, інформуючи їх про завершення завдання.

Альтернативні потоки:

1. якщо під час виконання завдання виконавець стикається з труднощами, він може залишити коментарі в деталях завдання або звернутися за допомогою до керівника проєкту;
2. якщо виконавець розуміє, що не може виконати завдання до встановленого терміну, він може оновити статус завдання на «Виконується» та надати коментар із поясненням затримки, що забезпечить кращий зв’язок і керування проєктом.

Цей сценарій описує кроки, які виконує виконавець для виконання завдання в інформаційній системі, підкреслюючи важливість зв’язку, документації та управління робочим процесом у процесах управління проєктом.

Розглянемо інший сценарій.

Сценарій використання: «Створити проєкт».

Актор: Проєкт менеджер

Передумова: Менеджер проєкту ввійшов в інформаційну систему та має необхідні дозволи для створення нового проєкту.

Основний потік:

1. менеджер проєкту переходить до розділу управління проєктами інформаційної системи, де він може переглянути список існуючих проєктів і варіанти створення проєкту;
2. менеджер проєкту натискає кнопку «Створити новий проєкт», яка відкриває форму створення проєкту;
3. у формі створення проєкту менеджер проєкту заповнює необхідні дані;
4. менеджер проєкту переглядає введену інформацію для забезпечення точності та повноти. Якщо будь-які обов’язкові поля відсутні або неправильні, система запропонує їм виправити проблеми, перш ніж продовжити;
5. після перевірки всієї інформації керівник проєкту натискає кнопку «Створити проєкт», щоб надіслати форму;
6. система обробляє запит, створює новий проєкт у базі даних і генерує унікальний ідентифікатор проєкту для використання в майбутньому;
7. менеджеру проєкту відображається повідомлення про підтвердження, яке вказує на те, що проєкт успішно створено. Система також може надавати опції для перегляду щойно створеного проєкту або повернення до списку проєктів;
8. менеджер проєкту тепер може перейти на сторінку деталей проєкту, де він може детальніше визначити завдання, встановити терміни та керувати ресурсами, пов’язаними з новим проєктом.

Альтернативні потоки:

1. якщо керівник проєкту вирішить скасувати процес створення проєкту в будь-який момент, він може натиснути кнопку «Скасувати», яка повертає його до попереднього екрана без збереження будь-яких змін;

2. якщо керівник проєкту стикається з помилкою під час подання (наприклад, проблеми з сервером або помилки перевірки), система відобразить відповідне повідомлення про помилку, що дозволить керівнику проєкту виправити інформацію та спробувати створити проєкт знову.

У цьому сценарії описано кроки, які виконує менеджер проєкту для створення нового проєкту в інформаційній системі, наголошуючи на важливості точного введення даних і ефективного налаштування проєкту в процесі управління проєктом.

2.1.2 Діаграма послідовності. Діаграми послідовності — це тип діаграми взаємодії в уніфікованій мові моделювання (UML), яка ілюструє, як об’єкти взаємодіють у певному сценарії з часом. Вони використовуються для моделювання динамічної поведінки системи, показуючи послідовність повідомлень, якими обмінюються різні компоненти або об’єкти, задіяні в конкретному випадку використання або процесі. Діаграми послідовності є особливо цінними для візуалізації потоку керування та даних, полегшуючи розуміння того, як працює система та як різні частини співпрацюють для досягнення спільної мети [7].

У центрі діаграми послідовності знаходяться учасники, які зазвичай представлені у вигляді вертикальних пунктирних ліній, відомих як лінії життя. Кожна лінія життя відповідає об'єкту або актору, який взаємодіє з системою. Взаємодії зображені горизонтальними стрілками між лініями життя, що представляють повідомлення, надіслані від одного учасника до іншого. Ці повідомлення можуть включати виклики методів, відповіді або сигнали, і вони розташовані в хронологічному порядку зверху вниз, демонструючи потік зв’язку в часі.

Діаграми послідовності дають кілька переваг у процесі розробки програмного забезпечення. Вони сприяють чіткій комунікації між членами

команди, візуально представляючи складні взаємодії, полегшуючи розуміння того, як компоненти співпрацюють. Крім того, діаграми послідовності служать цінною документацією для поведінки системи, допомагаючи зацікавленим сторонам зрозуміти функціональність системи та її компонентів.

Крім того, ці діаграми можуть допомогти у виявленні потенційних проблем або неефективності в процесі взаємодії, дозволяючи розробникам оптимізувати робочі процеси та покращити дизайн системи. Розбиваючи сценарії на детальні послідовності, команди можуть більш ефективно аналізувати вимоги, перевіряти вибір дизайну та гарантувати, що система відповідає потребам користувачів.

Діаграма послідовностей, зображена на рис. 2.2, містить такі об’єкти:

- “Виконавець”;
- “Проєкт менеджер”;
- “Проєкт”;
- “Задача”.

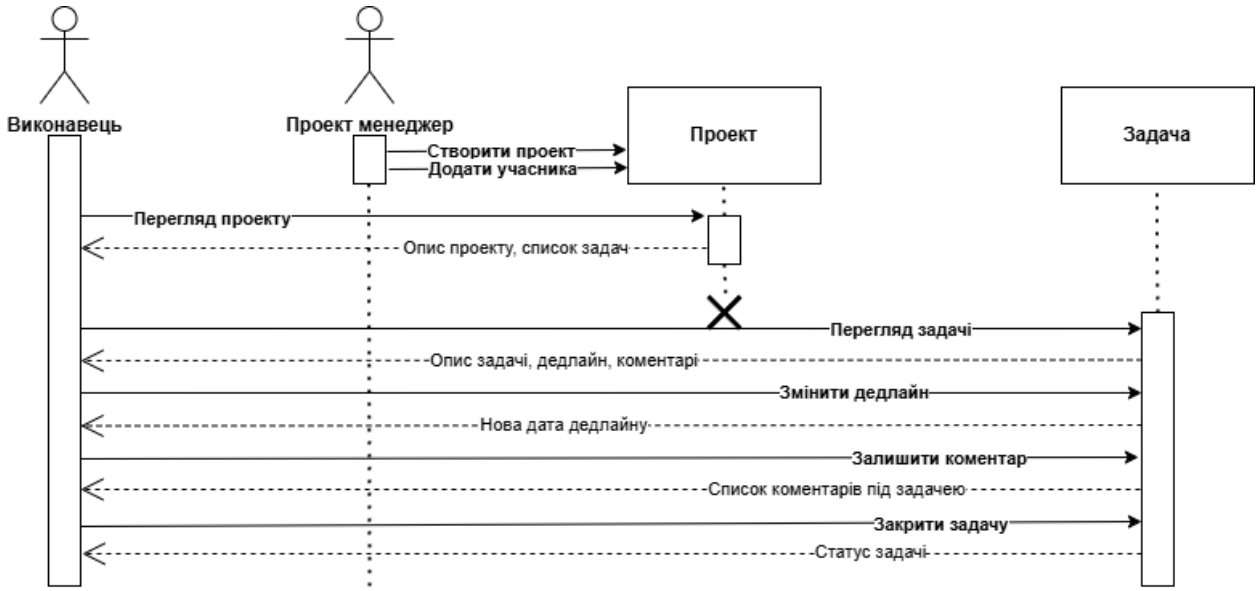

Рис. 2.2 Діаграма послідовності

Таким чином, діаграми послідовності є важливим інструментом в UML для моделювання динамічних взаємодій між об’єктами в системі. Ілюструючи потік повідомлень і час взаємодії, діаграми послідовності

покращують розуміння, покращують комунікацію та підтримують ефективне проєктування та розробку системи.

2.1.3 Діаграма активності. Діаграми діяльності — це тип діаграм поведінки в уніфікованій мові моделювання (UML), які представляють потік керування та даних у системі. Вони використовуються для моделювання динамічних аспектів системи шляхом ілюстрації послідовності дій, дій і рішень, які мають місце під час певного процесу або робочого циклу. Діаграми діяльності особливо корисні для візуалізації складних процесів, фіксації потоку роботи та визначення паралельних і умовних дій [8].

В основі діаграми діяльності – дії, які представляють завдання або дії, що виконуються в рамках процесу. Ці види діяльності з’єднані стрілками, які вказують на потік контролю від однієї діяльності до іншої. Діаграма також може включати вузли прийняття рішень, які представляють точки в робочому процесі, де необхідно прийняти рішення, що веде до різних гілок процесу на основі умов.

Діаграми діяльності пропонують кілька переваг у процесі розробки програмного забезпечення. Вони забезпечують чітке візуальне представлення робочих процесів, полегшуючи зацікавленим сторонам розуміння складних процесів і взаємодій. Розбиваючи процес на окремі дії, команди можуть визначити вузькі місця, надмірності та області, які потрібно вдосконалити.

Ці діаграми є особливо цінними під час збору вимог, оскільки вони допомагають охопити функціональні можливості системи та прояснити взаємодію між різними компонентами. Діаграми діяльності також можуть служити документацією для процесів, сприяючи спілкуванню між членами команди та забезпечуючи довідку для майбутнього розвитку та обслуговування.

Розроблена діаграма активності представлена на рис. 2.3.

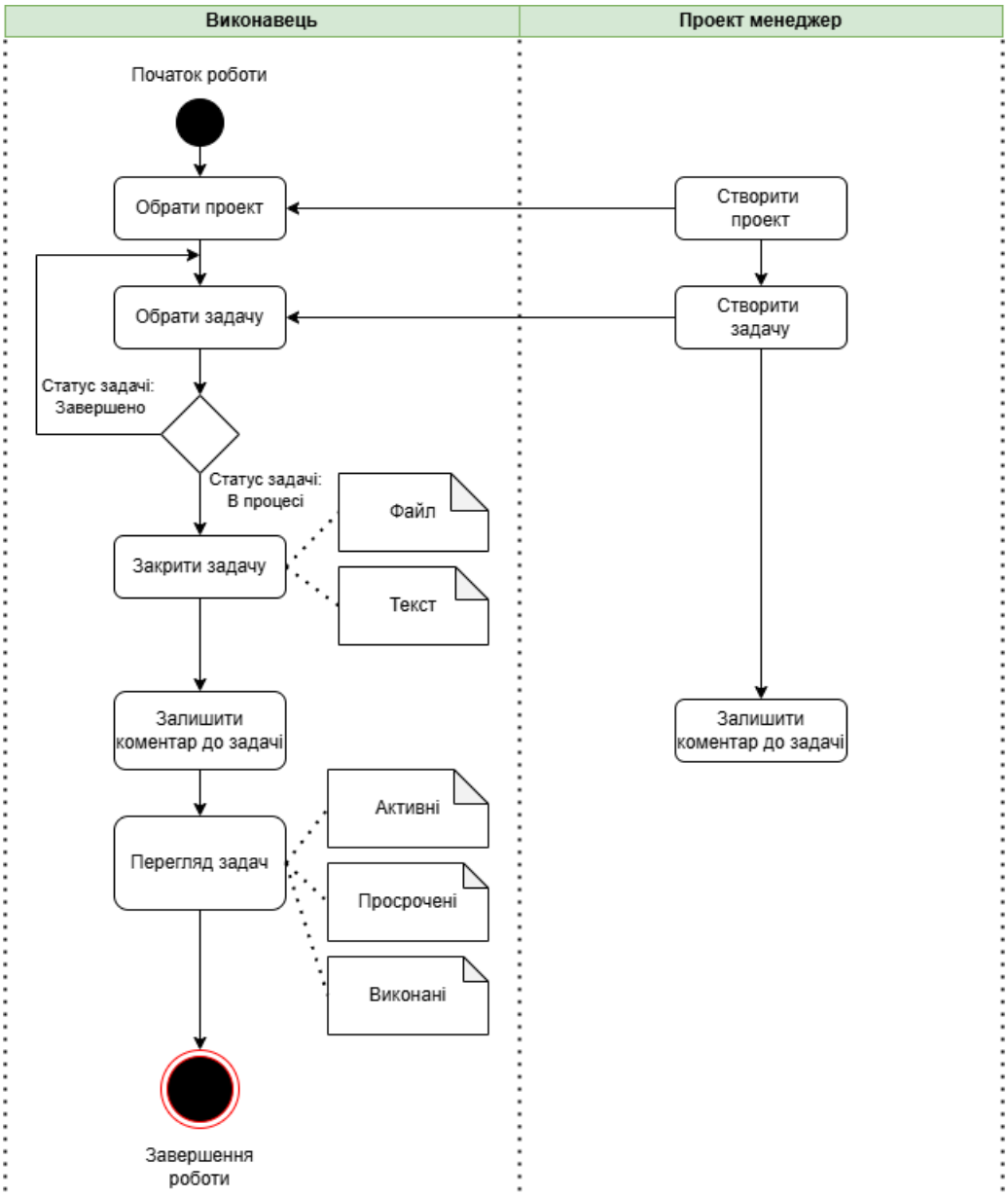

Рис. 2.3 Діаграма активності

Таким чином, діаграми діяльності є потужним інструментом в UML для моделювання динамічного потоку діяльності в системі. Ілюструючи послідовність завдань, точки прийняття рішень і паралельні процеси, діаграми діяльності покращують розуміння, підтримують оптимізацію процесів і полегшують ефективне проєктування та розробку системи.

2.1.4 Функціональна діаграма. Розробка, керована функціями (FDD) — це ітеративна та поетапна методологія розробки програмного забезпечення, яка зосереджена на своєчасному наданні відчутних робочих функцій

програмного забезпечення. FDD наголошує на співпраці між членами команди та зацікавленими сторонами, що дозволяє чітко спілкуватися та узгоджувати цілі проєкту. Одним із ключових компонентів FDD є використання спеціальних діаграм для візуалізації та керування функціями протягом усього процесу розробки [8].

Діаграми FDD відіграють вирішальну роль у представленні структури, поведінки та зв’язків функцій у програмній системі. Вони допомагають командам отримувати та передавати важливу інформацію про функціональні можливості системи, що полегшує планування, відстеження прогресу та забезпечує відповідність вимогам бізнесу.

Діаграми FDD надають кілька переваг у процесі розробки програмного забезпечення. Вони покращують комунікацію між членами команди та зацікавленими сторонами, надаючи чіткі візуальні представлення функцій та їхніх взаємозв’язків. Це допомагає переконатися, що всі мають спільне розуміння цілей і вимог проєкту.

Крім того, діаграми FDD полегшують краще планування та відстеження прогресу. Розбиваючи проєкт на керовані функції та візуалізуючи їхні зв’язки, команди можуть визначати пріоритети роботи, визначати залежності та ефективніше контролювати прогрес.

На даній діаграмі(Рис. 2.4) проаналізовано функціонал системи.

Дана діаграма на показує основні процеси нашої системи:

- управління користувачами;
- задача;
- проєкт;
- комунікація.

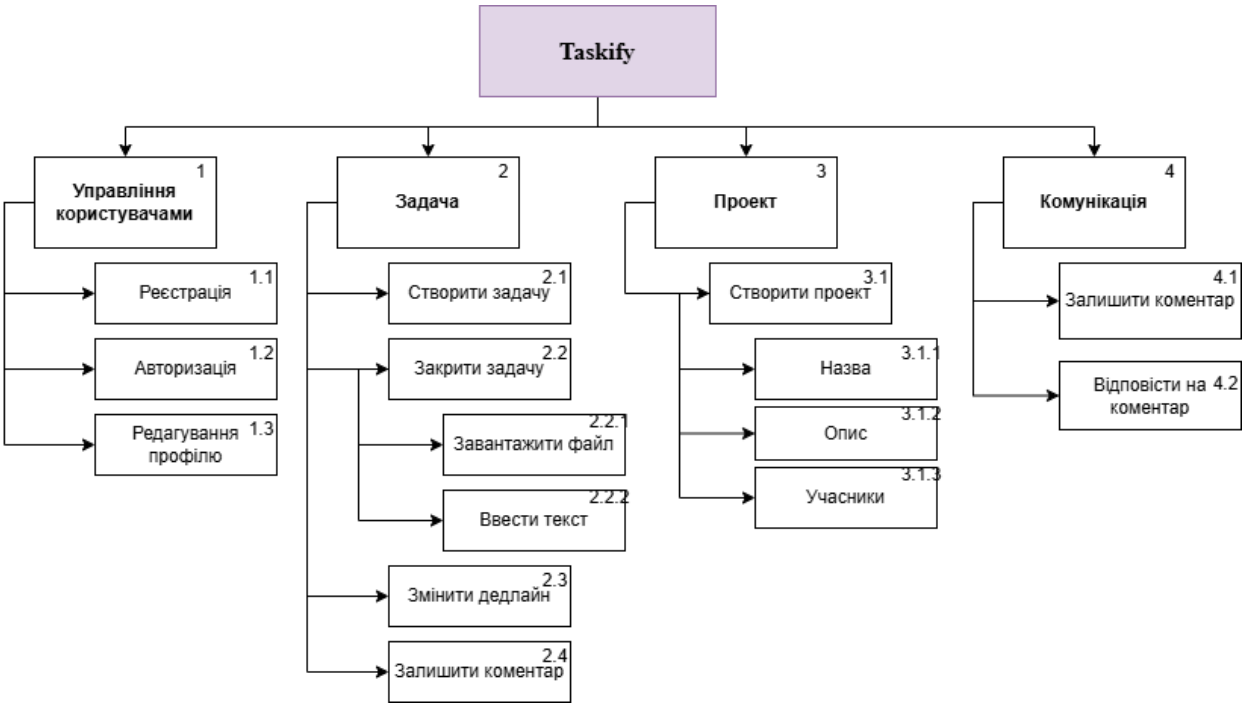

Рис. 2.4 FDD-діаграма

Перший процес “Управління користувачами” включає в себе:

1. реєстрація;
2. авторизація;
3. редагування профілю.

Другий процес “задача” включає в себе:

1. створити задачу;
2. закрити задачу;
3. завантажити файл;
4. ввести текст;
5. змінити дедлайн;
6. залишити коментар.

Третій процес “Проєкт” включає в себе:

1. створити проєкт;
2. назва;
3. опис;
4. учасники.

Четвертий процес “Комунікація” включає в себе:

1. залишити коментар;

2. відповісти на коментар.

Підводячи підсумок, можна сказати, що діаграми розробки, керовані функціями, є важливими інструментами для візуалізації та керування функціями в програмній системі. Використовуючи різні типи діаграм, наприклад списки функцій, діаграми класів, діаграми послідовності та діаграми компонентів, FDD допомагає командам ефективно спілкуватися, ефективно планувати та постачати високоякісне програмне забезпечення, яке відповідає потребам користувачів.

2.2 Абстракції предметної області

Абстракції предметної області — це спрощене представлення концепцій, сутностей і зв’язків у межах конкретної області, що забезпечує базове розуміння, яке інформує про проєктування та реалізацію інформаційної системи. У контексті управління проєктами ці абстракції допомагають ідентифікувати та визначити ключові компоненти, якими система керуватиме та з якими взаємодіятиме, сприяючи ефективному аналізу та комунікації протягом усього процесу розробки.

В основі цих абстракцій знаходяться сутності, які представляють основні об’єкти в межах домену управління проєктом. Загальні сутності включають проєкти, завдання, користувачів і команди. Проєкт означає унікальну ініціативу з конкретними цілями, часовими рамками та розподілом ресурсів, тоді як завдання являють собою окремі частини роботи, необхідні для досягнення цілей проєкту. Користувачі — це люди, які взаємодіють із системою, включаючи керівників проєктів, членів команди та зацікавлених сторін, а команди складаються з груп користувачів, які співпрацюють над різними проєктами [9].

Атрибути — це характеристики, які описують кожну сутність, надаючи подробиці про її стан. Наприклад, проєкт може мати такі атрибути, як назва, опис, дата початку, дата завершення та рівень пріоритету. Ці атрибути

допомагають уточнити важливу інформацію, пов’язану з кожною сутністю, забезпечуючи ефективне керування та відстеження.

Відносини ілюструють, як ці сутності взаємодіють одна з одною в рамках управління проєктом. Наприклад, проєкт може мати кілька пов’язаних завдань, встановлюючи зв’язок «один до багатьох». Крім того, конкретне завдання може бути призначено окремому користувачеві, підкреслюючи зв’язок між завданнями та користувачами, відповідальними за їх виконання. Користувачі також можуть бути частиною кількох команд, демонструючи спільний характер управління проєктами.

Процеси представляють робочі процеси або дії, які визначають, як сутності взаємодіють протягом певного часу для досягнення бажаних результатів. В управлінні проєктами типові процеси включають створення та планування проєктів, призначення завдань, моніторинг прогресу та оцінку продуктивності проєкту. Ці процеси допомагають оптимізувати роботу та гарантують, що всі члени команди розуміють свої ролі в системі.

Варіанти використання відіграють вирішальну роль у фіксуванні взаємодії між користувачами та системою. Вони описують конкретні функції, необхідні для підтримки домену управління проєктами, наприклад створення нових проєктів, призначення завдань, оновлення статусів завдань і створення звітів про хід. Ці випадки використання надають чіткі вказівки щодо розробки системи, гарантуючи, що вона ефективно відповідає потребам користувачів [9].

Встановлюючи ці абстракції, групи розробників можуть отримати структуроване розуміння домену управління проєктами. Ці фундаментальні знання не тільки керують розробкою та впровадженням інформаційної системи, але й гарантують, що вона ефективно відповідає вимогам користувачів і підтримує ефективні процеси управління проєктами. Зрештою, абстракції предметної області спрощують складні концепції, полегшуючи аналіз вимог, проєктування архітектури системи та полегшують спілкування із зацікавленими сторонами протягом життєвого циклу розробки Для системи, що розробляється, були розроблені конкретні абстракції, які зображені на малюнку 2.5.

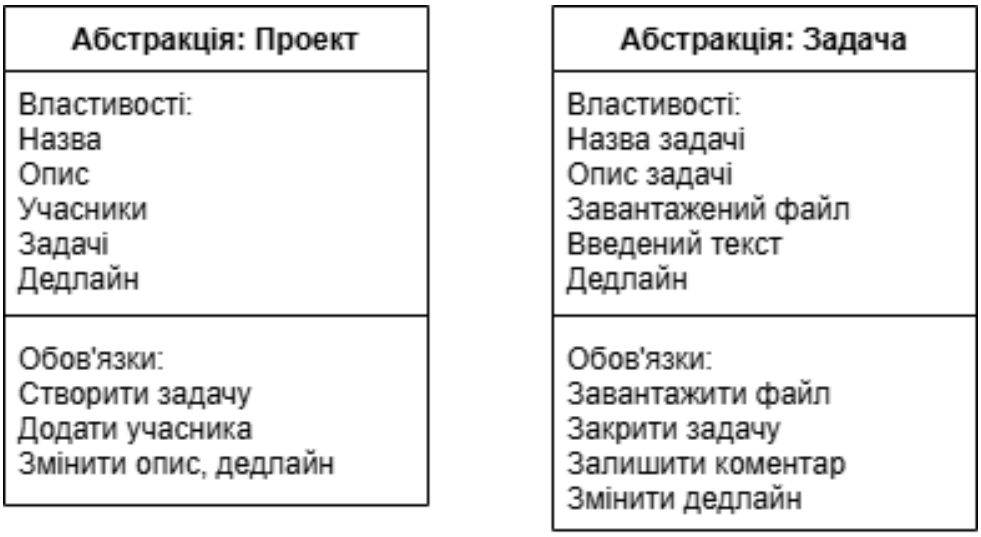

Рис. 2.5 Абстракції «Проєкт» та “Задача”

2.3 Діаграма класів

Діаграми класів є фундаментальним компонентом уніфікованої мови моделювання (UML), який використовується для моделювання статичної структури системи. Вони забезпечують візуальне представлення класів системи, їхніх атрибутів, методів і зв’язків між ними. Діаграми класів є важливими в об’єктно-орієнтованому проєктуванні та розробці, оскільки вони допомагають розробникам і зацікавленим сторонам зрозуміти архітектуру системи та взаємодію різних компонентів.

В основі діаграми класів знаходяться класи, які є схемами для створення об’єктів. Кожен клас представляє певну сутність або концепцію в системі та зазвичай зображується у вигляді прямокутника, поділеного на три частини: верхня частина містить назву класу, середня частина містить список атрибутів (властивостей) класу, а нижня частина детально описує методи (функції або операції), які може виконувати клас [10].

Діаграми класів виконують кілька важливих завдань у процесі розробки програмного забезпечення. Вони надають чітке візуальне уявлення про структуру системи, полегшуючи розробникам, аналітикам і зацікавленим сторонам розуміння компонентів та їх взаємодії. Визначаючи класи, атрибути,

методи та зв’язки, діаграми класів сприяють ефективній комунікації між членами команди та служать цінним довідником протягом життєвого циклу розробки.

Крім того, діаграми класів відіграють вирішальну роль на етапі проєктування розробки програмного забезпечення. Вони допомагають ідентифікувати та уточнювати вимоги, керувати впровадженням класів і гарантувати, що система дотримується об’єктно-орієнтованих принципів. Моделюючи статичну структуру системи, діаграми класів також допомагають визначити потенційні недоліки конструкції, надмірності та області для вдосконалення.

Було створено власну діаграму класів за об'єктно орієнтованим підходом для даного веб-сайту (рис. 2.6).

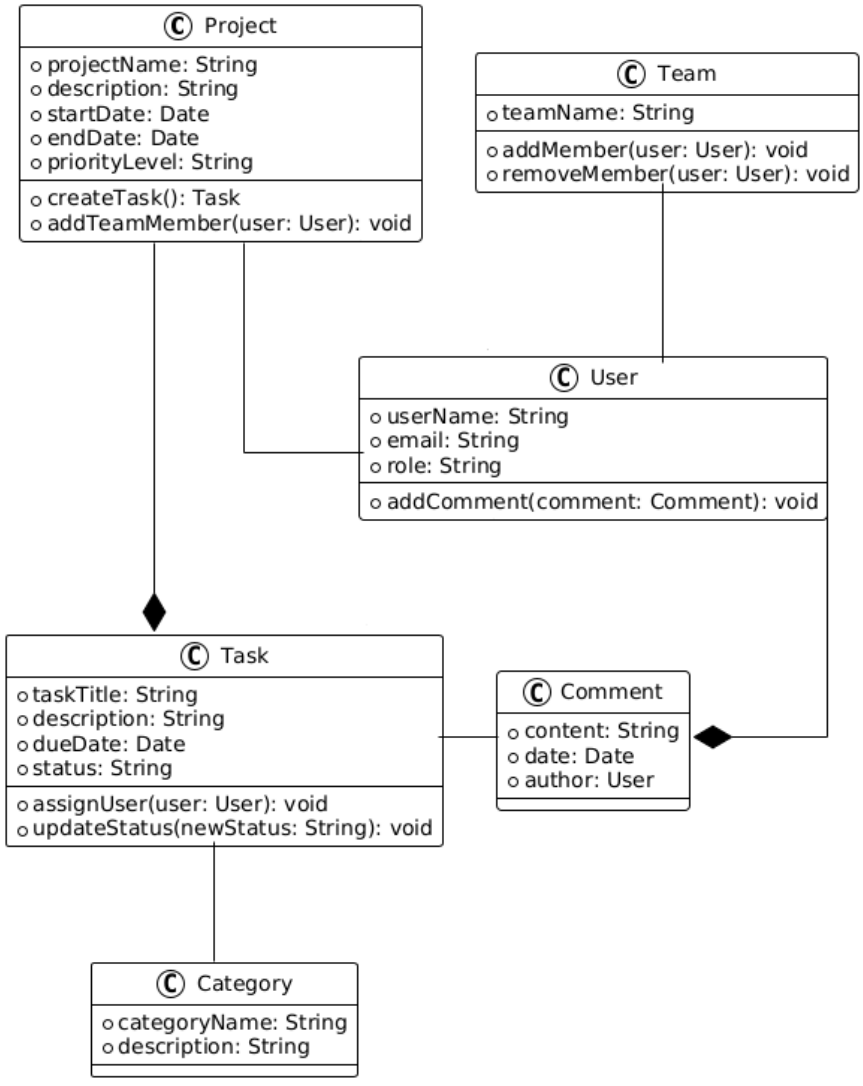

Рис. 2.7 Діаграма класів

Таким чином, діаграми класів є життєво важливим інструментом в UML для моделювання статичної структури системи. Ілюструючи класи, їхні атрибути, методи та зв’язки, діаграми класів покращують розуміння, покращують комунікацію та підтримують ефективне проєктування та розробку проєктів об’єктно-орієнтованого програмного забезпечення [11].

2.4 Логічна модель даних

ERwin — це потужний інструмент моделювання даних і проєктування баз даних, який широко використовується в області управління даними та розробки баз даних. Він надає повний набір функцій, які дозволяють користувачам створювати, візуалізувати та керувати моделями даних, полегшуючи проєктування та впровадження складних систем баз даних. ERwin особливо популярний серед архітекторів даних, адміністраторів баз даних і розробників завдяки своїй здатності оптимізувати процес розробки бази даних і покращити співпрацю між членами команди.

Однією з основних сильних сторін ERwin є його здатність полегшувати як логічне, так і фізичне моделювання даних. Користувачі можуть створювати логічні моделі даних, які представляють організацію та структуру даних, не заглиблюючись у специфіку реалізації бази даних. Це високорівневе подання допомагає зацікавленим сторонам зрозуміти взаємозв’язки та обмеження об’єктів даних, полегшуючи визначення вимог і розробку ефективної схеми бази даних [12].

Окрім логічного моделювання, ERwin також підтримує фізичне моделювання даних, що дозволяє користувачам визначати фактичну реалізацію бази даних. Це включає визначення типів даних, стратегій індексування та інших конфігурацій, що стосуються бази даних. ERwin генерує сценарії SQL на основі фізичної моделі, що дозволяє розробникам створювати та підтримувати бази даних ефективно.

ERwin пропонує зручний графічний інтерфейс, який спрощує процес розробки моделей даних. Користувачі можуть легко створювати сутності, атрибути та зв’язки за допомогою функції перетягування, а інструмент автоматично генерує відповідні візуальні представлення. Цей візуальний аспект має вирішальне значення для розуміння складних структур даних і забезпечення того, щоб усі зацікавлені сторони були на одній сторінці.

Співпраця — ще одна ключова функція ERwin. Інструмент підтримує контроль версій і командну співпрацю, дозволяючи кільком користувачам працювати над тією самою моделлю даних одночасно. Це особливо корисно у великих проєктах, де для точного моделювання даних важливий внесок від різних зацікавлених сторін. ERwin також надає функції для документування моделей даних, включаючи можливість додавати анотації та описи до сутностей і зв’язків, що покращує спілкування та розуміння між членами команди [13].

Крім того, ERwin інтегрується з різними системами керування базами даних (СУБД), що дозволяє користувачам легко імпортувати та експортувати моделі даних. Ця сумісність гарантує узгодження моделей даних з існуючими базами даних і сприяє бездоганній інтеграції в загальну екосистему керування даними.

Таким чином, ERwin — це комплексний інструмент моделювання даних, який спрощує процес проєктування, візуалізації та керування системами баз даних. Підтримуючи як логічне, так і фізичне моделювання даних, пропонуючи зручний інтерфейс і полегшуючи співпрацю між членами команди, ERwin дозволяє організаціям створювати ефективні та ефективні рішення для баз даних, які відповідають їхнім потребам управління даними. Його можливості інтеграції ще більше підвищують його цінність, роблячи його популярним вибором для професіоналів обробки даних у різних галузях.

Логічна модель системи представлена на рис. 2.8.

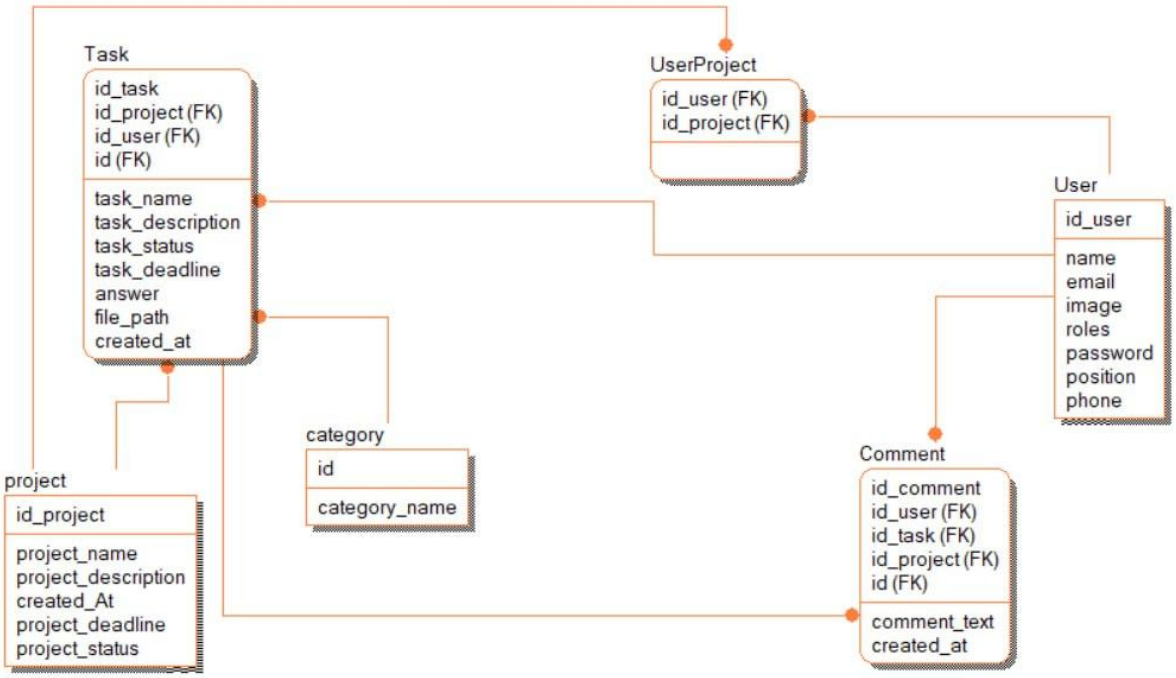

Рис. 2.8 ER-діаграма

Логічна модель складається з таких сутностей:

Task

1. id_task: унікальний ідентифікатор для завдання;
2. id_project (FK): зовнішній ключ, що посилається на пов’язаний проєкт;
3. id_user (FK): Зовнішній ключ, який посилається на користувача, якому призначено завдання;
4. task_name: Назва завдання;
5. task_description: Детальний опис завдання;
6. task_status: поточний статус завдання (наприклад, очікує на розгляд, виконується, завершено);
7. task_deadline: Кінцевий термін виконання завдання;
8. відповідь: Поле для будь-яких відповідей, пов’язаних із завданням;
9. file_path: Шлях до будь-яких файлів, пов’язаних із завданням;
10. created_at: Позначка часу, яка вказує, коли було створено завдання.

UserProject

1. id_user (FK): зовнішній ключ, що посилається на користувача;

2. id_project (FK): Зовнішній ключ, що посилається на проєкт, до якого призначено користувача.

Project

1. id_project: унікальний ідентифікатор проєкту;
2. project_name: Назва проєкту;
3. project_description: Детальний опис проєкту;
4. created_at: Позначка часу, що вказує, коли проєкт було створено;
5. project_deadline: Кінцевий термін завершення проєкту;
6. project_status: поточний статус проєкту (наприклад, активний, завершений).

Category

1. id: унікальний ідентифікатор для категорії;
2. category_name: назва категорії.

User

1. id_user: унікальний ідентифікатор користувача;
2. name: ім'я користувача;
3. email: адреса електронної пошти користувача;
4. зображення: зображення профілю користувача;
5. ролі: Ролі, призначені користувачу (наприклад, адміністратор, учасник);
6. пароль: Пароль для автентифікації користувача;
7. позиція: позиція користувача в організації;
8. phone: Номер телефону користувача.

Comment

1. id_comment: унікальний ідентифікатор для коментаря;
2. id_user (FK): зовнішній ключ, який посилається на користувача, який зробив коментар;
3. id_task (FK): зовнішній ключ, що посилається на завдання, з яким пов’язаний коментар;

4. id_project (FK): зовнішній ключ, який посилається на проєкт, пов’язаний із коментарем;
5. id (FK): інший ідентифікатор для додаткового посилання;
6. comment_text: текстовий вміст коментаря;
7. created_at: Позначка часу, що вказує, коли було створено коментар.

Від завдання до проєкту: завдання пов’язане з одним проєктом (зв’язок «один до багатьох»).

Завдання для користувача: завдання можна призначити одному користувачеві (відношення «один до багатьох»).

UserProject: ця сутність представляє зв’язок «багато-до-багатьох» між користувачами та проєктами, вказуючи, які користувачі беруть участь у яких проєктах.

Коментар до завдання: коментар пов’язаний з одним завданням (зв’язок «один до багатьох»).

Коментар для користувача: коментар робить один користувач (відношення «один до багатьох»).

Коментар до проєкту: коментар може бути пов’язаний з одним проєктом (зв’язок «один до багатьох»).

Завдання за категоріями: хоча це явно не показано у витягнутому тексті, завдання можна класифікувати, що забезпечує кращу організацію та фільтрацію.

Ця схема бази даних забезпечує структурований підхід до керування проєктами, завданнями, користувачами та їх взаємодією, забезпечуючи ефективну організацію та моніторинг процесів в інформаційній системі.

2.5 Висновки до 2 розділу

У другому розділі було проведено детальне моделювання предметної області системи управління проєктами. Спочатку наведено загальні відомості, що визначають контекст і межі досліджуваної системи.

В межах об’єктного та функціонального моделювання ідентифіковано ключові об’єкти, їх властивості та взаємодії, що дозволило чітко описати основні функції та процеси системи. Це дало змогу розбити систему на логічні компоненти та встановити їх зв’язки.

Розроблена діаграма класів відобразила структуру основних сутностей, їх атрибутів і зв’язків, що є фундаментом для побудови бази даних та реалізації програмної логіки.

В результаті сформовано логічну модель даних, яка визначає структуру збереження інформації, враховуючи зв’язки між об’єктами та забезпечуючи цілісність даних.

3 ПРОЄКТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

3.1 Вибір системи управління базою даних та її реалізація

Система керування реляційною базою даних (RDBMS) — це спеціалізована програмна програма, призначена для керування базами даних у структурованому форматі за допомогою рядків і стовпців. У RDBMS дані організовані в таблиці, кожна з яких представляє різні сутності або об’єкти в системі. Ці таблиці складаються із записів (рядків) і полів (стовпців), що забезпечує систематичний підхід до зберігання та пошуку даних.

Однією з визначальних особливостей RDBMS є її здатність встановлювати зв’язки між таблицями. Наприклад, таблицю клієнтів можна зв’язати з таблицею замовлень, що дозволяє відображати складні взаємодії даних. Цей реляційний аспект дозволяє користувачам зрозуміти, як різні частини даних пов’язані між собою.

Як правило, RDBMS використовує структуровану мову запитів (SQL) як стандартну мову для запитів і обробки даних. SQL надає користувачам інструменти для виконання різних операцій, включаючи вставку, оновлення,

видалення та отримання даних із бази даних. Підтримка SQL дозволяє розробникам і аналітикам ефективно взаємодіяти з базою даних.

Цілісність даних є важливою характеристикою RDBMS, що забезпечується застосуванням властивостей ACID, які означають атомарність, узгодженість, ізоляцію та довговічність. Ці властивості гарантують, що транзакції бази даних обробляються надійно, зберігаючи валідність бази даних і запобігаючи аномалії даних. Крім того, RDBMS забезпечує цілісність даних через обмеження, такі як первинні ключі, які забезпечують унікальність, і зовнішні ключі, які підтримують посилальну цілісність між пов’язаними таблицями [14].

Нормалізація є ще одним важливим аспектом RDBMS. Він включає в себе організацію даних в окремі таблиці та визначення зв’язків для мінімізації надмірності та покращення цілісності даних. Цей структурований підхід допомагає оптимізувати керування даними та підвищує ефективність роботи з ними.

Системи RDBMS підтримують одночасний доступ, дозволяючи кільком користувачам читати та записувати дані одночасно. Ця багатокористувацька здатність гарантує, що різні користувачі можуть взаємодіяти з базою даних без шкоди для цілісності чи узгодженості даних.

Кілька добре відомих прикладів RDBMS включають MySQL, систему з відкритим кодом, яка широко використовується для веб-додатків; PostgreSQL, відомий своєю розширюваністю та сумісністю з SQL; Oracle Database, потужний комерційний варіант, який часто використовується в корпоративних налаштуваннях; Microsoft SQL Server, який часто використовується в бізнес-середовищі; і SQLite, легка файлова система, ідеальна для мобільних додатків і вбудованих систем [15].

РСУБД застосовується в різних галузях, таких як бізнес-додатки для керування даними клієнтів, замовленнями та інформацією про співробітників; фінансові системи для відстеження операцій і рахунків; системи управління

контентом для зберігання статей і коментарів користувачів; а також сховища даних для агрегування та аналізу великих обсягів інформації з різних джерел.

Таким чином, система управління реляційною базою даних (RDBMS) служить життєво важливим інструментом для зберігання, керування та отримання структурованих даних. Його здатність встановлювати зв’язки, забезпечувати цілісність даних і підтримувати складні запити робить його важливим компонентом у численних програмних додатках і системах, призначених для організацій будь-якого розміру.

При розробці інформаційної системи для організації та моніторингу процесів в управлінні проєктами рішення про вибір MySQL як системи управління базами даних було обумовлено декількома ключовими факторами, які відповідають вимогам і цілям проєкту.

MySQL відомий своєю надійністю та продуктивністю, що робить його надійним вибором для програм, які потребують ефективної обробки даних. Враховуючи природу систем управління проєктами, які часто включають обробку численних завдань, користувачів і проєктів, здатність MySQL ефективно керувати великими обсягами даних гарантує, що система працюватиме добре, навіть якщо кількість користувачів і складність даних зростають.

Іншою значною перевагою MySQL є його природа з відкритим кодом, що забезпечує економічну ефективність проєкту. Як база даних з відкритим вихідним кодом, MySQL є безкоштовним для використання, що дозволяє розробляти бюджет без шкоди для якості та функціональності. Цей аспект особливо корисний для невеликих організацій або проєктів з обмеженими ресурсами, що робить його доступним варіантом для широкого кола користувачів.

MySQL також підтримує надійний набір функцій, необхідних для розробки системи управління проєктами. Ці функції включають підтримку складних запитів, керування транзакціями та цілісність даних через відповідність ACID (атомарність, послідовність, ізоляція, довговічність). Ця

можливість має вирішальне значення для того, щоб дані залишалися точними та надійними, особливо в системі, де декілька користувачів можуть отримувати доступ до даних і змінювати їх одночасно [16].

Широка популярність MySQL означає, що існує величезна спільнота користувачів і розробників, а також доступна обширна документація та ресурси. Ця мережа підтримки полегшує пошук рішень потенційних проблем і сприяє співпраці з іншими розробниками. Крім того, наявність численних інструментів і фреймворків, які легко інтегруються з MySQL, спрощує процес розробки та підвищує загальну продуктивність.

Гнучкість і масштабованість MySQL додатково сприяють його придатності для проєкту. Оскільки інформаційна система розвивається та вимоги змінюються, MySQL може легко пристосуватися до зростання, чи то шляхом вертикального масштабування за допомогою більш потужного обладнання, чи горизонтального шляхом реплікації та методів кластеризації. Ця адаптивність гарантує, що база даних може рости разом із проєктом, задовольняючи майбутні вимоги без необхідності повного перепроєктування.

Сумісність MySQL з різними мовами програмування та фреймворками, зокрема PHP і Symfony, які використовуються в розробці системи, забезпечує плавний процес інтеграції. Ця синергія між базою даних і середовищем розробки оптимізує робочий процес і допомагає скоротити час розробки.

Підсумовуючи, вибір MySQL як бази даних для інформаційної системи для організації та моніторингу процесів в управлінні проєктами ґрунтується на її надійності, економічній ефективності, наборі функцій, підтримці спільноти, гнучкості, масштабованості та сумісності з вибраним стеком технологій. Ці атрибути роблять MySQL ідеальною основою для створення надійної та ефективної системи управління проєктами.

Під час проєктування таблиць у фізичній моделі основою слугували сутності з логічної моделі. Атрибути цих сутностей були використані для розробки структури таблиці. Отриману схему бази даних зображено на рис. 3.1.

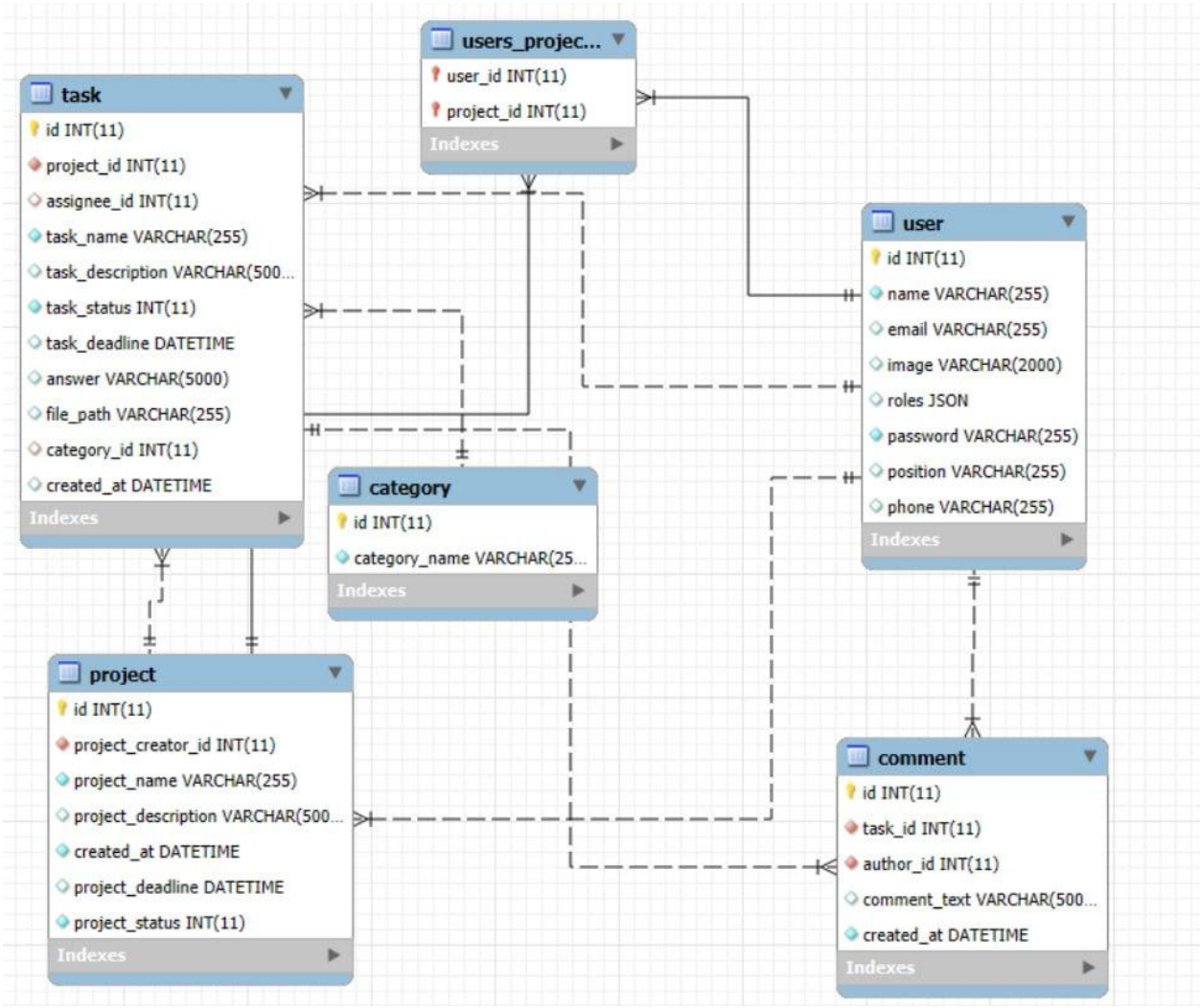

Рис. 3.1 База даних системи

3.2 Архітектура програмного забезпечення

Архітектура програмного забезпечення — це високорівнева структура й організація програмної системи, що охоплює її компоненти, їхні взаємозв'язки та керівні принципи її проєктування й розвитку. Він виступає як схема як для системи, так і для проєкту розробки, описуючи, як різні компоненти взаємодіють та інтегруються для задоволення конкретних функціональних і нефункціональних вимог.

В основі архітектури програмного забезпечення лежать компоненти, які представляють окремі будівельні блоки системи. Вони можуть включати модулі, класи, служби або функції, кожна з яких розроблена з чіткою метою та

інкапсулює певну функціональність. Відносини між цими компонентами визначають, як вони спілкуються та взаємодіють, включаючи потік даних, потік керування, залежності та інтерфейси, які сприяють співпраці [19].

Архітектурні стилі та шаблони забезпечують усталені рішення для поширених проблем проєктування. Приклади цих стилів включають багаторівневу архітектуру, яка поділяє систему на окремі рівні, відповідальні за різні функціональні можливості, такі як презентація, бізнес-логіка та доступ до даних. Архітектура мікросервісів організовує систему як набір слабко пов’язаних служб, які можна розробляти, розгортати та масштабувати незалежно. Архітектура, керована подіями, використовує події для ініціювання дій і забезпечення зв’язку між компонентами, підвищуючи швидкість реакції та масштабованість. Навпаки, клієнт-серверна архітектура розділяє систему на клієнтські та серверні компоненти, причому клієнти запитують послуги від серверів.

Атрибути якості є ще одним важливим аспектом архітектури програмного забезпечення, оскільки архітектурні рішення значно впливають на такі фактори, як продуктивність, масштабованість, зручність обслуговування, безпека та зручність використання. Ефективна архітектура прагне збалансувати ці атрибути на основі конкретних потреб програми та її користувачів.

Чітка документація архітектури програмного забезпечення є важливою для ефективного спілкування між зацікавленими сторонами, включаючи розробників, керівників проєктів і клієнтів. Ця документація, як правило, містить архітектурні діаграми, рішення щодо дизайну та обґрунтування цих виборів, що служить довідкою протягом усього процесу розробки.

Крім того, архітектура програмного забезпечення має бути гнучкою та адаптованою, враховуючи зміни, які виникають з часом через зміну вимог, технологічний прогрес або зміни в бізнес-цілях. Добре розроблена архітектура дозволяє легко модифікувати та покращувати, не викликаючи значних збоїв у існуючій системі.

Неможливо переоцінити значення архітектури програмного забезпечення в процесі розробки. Він надає вказівки та вказівки для команди розробників, гарантуючи, що всі члени розуміють структуру та цілі системи. Виявляючи потенційні проблеми на ранній стадії проєктування, архітектура допомагає зменшити ризики, пов’язані з масштабованістю, продуктивністю та зручністю обслуговування. Крім того, чітко визначена архітектура сприяє співпраці між членами команди, полегшуючи розподіл завдань та інтеграцію внесків [20].

Зрештою, архітектура програмного забезпечення є фундаментальним аспектом розробки програмного забезпечення, який визначає структуру, компоненти та зв’язки всередині системи. Він служить комплексним проєктом, який керує розробкою, впровадженням і еволюцією програмного забезпечення, гарантуючи, що кінцевий продукт ефективно відповідає своїм функціональним і нефункціональним вимогам, залишаючись адаптованим до майбутніх змін.

Рішення запровадити клієнт-серверну архітектуру для інформаційної системи, призначеної для організації та моніторингу процесів в управлінні проєктами, ґрунтувалося на кількох ключових факторах, які відповідають вимогам проєкту та очікуваним моделям використання.

Однією з головних переваг клієнт-серверної архітектури є її здатність відокремити інтерфейс користувача (клієнт) від обробки та зберігання даних (сервер). Такий поділ дозволяє створити більш організовану структуру, де клієнти обробляють взаємодію користувачів, а сервер керує даними та бізнес-логікою. Така конструкція підвищує зручність обслуговування, оскільки оновлення або зміни логіки на стороні сервера можна робити незалежно від клієнтської програми. Цей модульний підхід спрощує розробку та полегшує пошук і налагодження несправностей.

Масштабованість є ще одним критичним фактором, який вплинув на вибір клієнт-серверної архітектури. Оскільки кількість користувачів і проєктів зростає, система повинна бути в змозі задовольнити підвищений попит без втрати продуктивності. У налаштуваннях клієнт-сервер кілька клієнтів можуть

підключатися до централізованого сервера, що забезпечує ефективний розподіл ресурсів і керування ними. Ця архітектура спрощує масштабування системи шляхом оновлення ресурсів сервера або додавання додаткових серверів для обробки збільшених навантажень, забезпечуючи оперативне реагування користувачів, навіть якщо база користувачів розширюється.

Крім того, клієнт-серверна архітектура підвищує безпеку та цілісність даних. Завдяки централізованому зберіганню даних на сервері система може запровадити надійні заходи безпеки, такі як контроль доступу та шифрування даних. Таке налаштування зменшує ризик витоку даних і гарантує захист конфіденційної інформації. Крім того, централізоване керування даними допомагає підтримувати послідовність і цілісність даних, оскільки всі користувачі взаємодіють з одним сховищем даних, а не мають окремі копії на своїх локальних машинах.

Архітектура також підтримує одночасний доступ, що дозволяє кільком користувачам одночасно взаємодіяти з системою. Ця функція особливо корисна в контексті управління проєктами, коли членам команди може знадобитися співпрацювати в реальному часі. Модель клієнт-сервер гарантує, що зміни, внесені одним користувачем, негайно відображаються для інших, полегшуючи спілкування та співпрацю між членами команди [21].

Іншим фактором, який сприяв цьому рішенню, є гнучкість, яку пропонує клієнт-серверна архітектура з точки зору вибору технологій. Сервер можна побудувати за допомогою різних технологій, що дозволяє реалізувати потужні серверні інфраструктури та бази даних, які можуть ефективно справлятися зі складними процесами управління проєктами. У той же час клієнти можуть бути розроблені для кількох платформ, включаючи веб, настільні та мобільні пристрої, забезпечуючи доступ користувачів до системи з різних пристроїв.

Клієнт-серверна архітектура добре узгоджується з довгостроковими цілями проєкту. Оскільки інформаційна система розвивається та додається нові функції, модульний характер цієї архітектури дозволяє легше інтегрувати

додаткові функції та технології. Ця адаптивність гарантує, що система може рости та розвиватися разом із мінливими потребами організації.

Таким чином, рішення вибрати клієнт-серверну архітектуру для інформаційної системи для організації та моніторингу процесів управління проєктами ґрунтується на її організаційній структурі, масштабованості, безпеці, підтримці одночасного доступу, гнучкості та довгостроковій адаптивності. Цей архітектурний вибір забезпечує міцну основу для створення надійної, ефективної та безпечної системи, яка відповідає вимогам користувачів і підтримує ефективне управління проєктами.

Нижче буде продемонстровано 4-шарова архітектура даного програмного забезпечення (рис. 3.2)

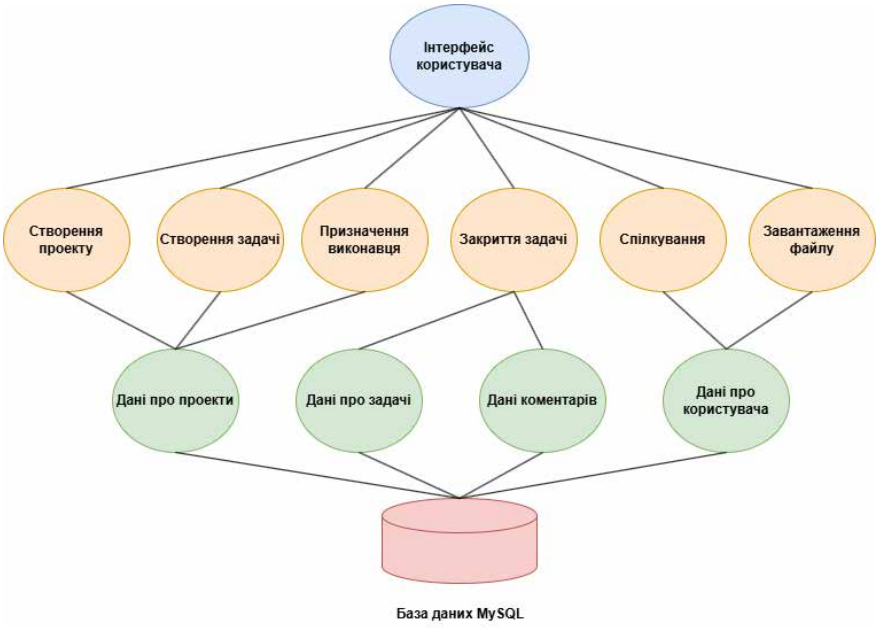

Рис. 3.2 Багатошарова архітектура веб-сайту

З рисунка вище ми бачимо, що наша система побудована на 4-рівневій багатошаровій архітектурі, що складається з презентаційного рівня, бізнес-рівня, рівня збереження інформації та рівня бази даних.

3.3 Організаційна структура програмного забезпечення

3.3.1 Діаграма пакетів. Діаграми пакетів — це тип структурної діаграми в уніфікованій мові моделювання (UML), яка зображує організацію та зв’язки пакетів у програмній системі. Пакет — це групування пов’язаних класів, інтерфейсів, компонентів або інших пакетів, які допомагають керувати складністю, організовуючи елементи в об’єднані одиниці. Діаграми пакетів надають високорівневе уявлення про архітектуру системи, ілюструючи, як різні пакети взаємодіють і залежать один від одного.

Однією з головних цілей пакетної діаграми є спрощення представлення великих систем, розбиваючи їх на менші, більш керовані частини. Цей модульний підхід дозволяє розробникам і зацікавленим сторонам зрозуміти структуру системи та організацію її компонентів, не втрачаючись у деталях окремих класів або елементів.

Діаграми пакетів служать кільком важливим цілям у розробці програмного забезпечення. Вони надають чітке та стисле представлення архітектури системи, полегшуючи розробникам, архітекторам та зацікавленим сторонам розуміння організації компонентів та їхніх взаємозв’язків. Візуалізуючи пакети та їхні залежності, команди можуть виявити потенційні проблеми зі зв’язком і згуртованістю, що дозволить їм покращити загальний дизайн і зручність обслуговування системи [24].

Крім того, пакетні діаграми полегшують спілкування між членами команди, надаючи спільне розуміння структури системи. Це особливо цінно у великих проєктах, де кілька команд можуть працювати над різними аспектами системи. Чітко визначаючи межі та взаємодію між пакетами, команди можуть ефективніше співпрацювати та гарантувати бездоганну інтеграцію їхніх компонентів.

Діаграма пакетів (рис. 3.3) для даного програмного забезпечення використовується для візуального представлення архітектури та організації системи. Діаграма пакетів зображує різні пакети, які складають програмне забезпечення, і ілюструє зв’язки та залежності між ними.

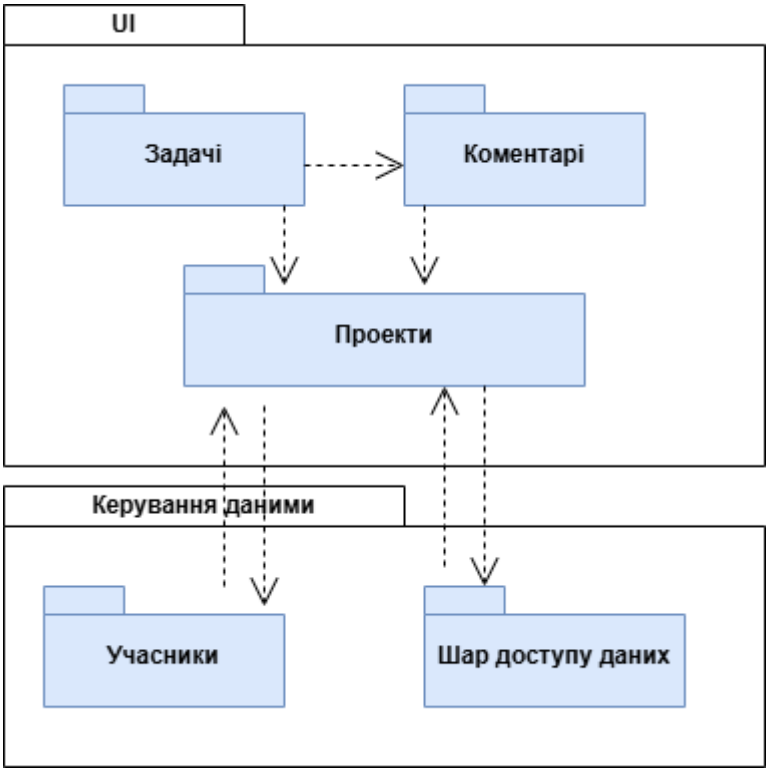

Рис. 3.3 Діаграма пакетів

3.4 Вимоги до апаратного та програмного забезпечення

Діаграми розгортання є основним типом структурної діаграми в уніфікованій мові моделювання (UML), призначеній для ілюстрації фізичного розгортання артефактів у програмній системі. Ці діаграми пропонують візуальне уявлення про те, як програмні компоненти розподілені між різними апаратними вузлами, ефективно демонструючи взаємозв’язки між програмним забезпеченням і фізичними пристроями або середовищами, в яких вони працюють. Зосереджуючись на фізичній архітектурі системи, діаграми розгортання відіграють вирішальну роль у розумінні складних програм із кількома компонентами та середовищами.

Ключові елементи діаграм розгортання включають вузли, артефакти,

підключення та специфікації розгортання. Вузли представляють фізичні апаратні пристрої або середовища, де розгортаються компоненти програмного забезпечення, такі як сервери, робочі станції, мобільні пристрої або хмарна інфраструктура. Ці вузли зазвичай зображуються як тривимірні прямокутники або куби на діаграмі.

З іншого боку, артефакти стосуються фізичних файлів або компонентів, розгорнутих на вузлах. Це може включати виконувані файли, бібліотеки, сценарії та файли конфігурації, які зображені у вигляді прямокутників, приєднаних до відповідних вузлів. З'єднання між вузлами та артефактами представлені суцільними лініями, що вказують на шляхи зв'язку, які існують між пристроями, наприклад мережеві з'єднання або протоколи зв'язку.

Специфікації розгортання надають додатковий контекст щодо того, як артефакти розгортаються на певних вузлах. Вони можуть містити відомості про конфігурацію, середовище виконання та будь-які залежності між компонентами.

Використання діаграм розгортання приносить кілька значних переваг процесу розробки програмного забезпечення. Вони сприяють чіткій візуалізації архітектури системи, дозволяючи зацікавленим сторонам зрозуміти, як різні компоненти розподілені та як вони взаємодіють у різних середовищах. Це розуміння є життєво важливим для ефективного планування та розподілу ресурсів, оскільки воно допомагає визначити необхідні апаратні та програмні ресурси, необхідні для розгортання [26].

Крім того, діаграми розгортання допомагають визначити залежності та потенційні вузькі місця в системі. Ілюструючи зв'язки між вузлами та артефактами, ці діаграми можуть інформувати про рішення, пов'язані з масштабуванням, балансуванням навантаження та відмовостійкістю, які є критичними для забезпечення надійності та продуктивності системи.

Крім того, діаграми розгортання покращують спілкування між членами команди, керівниками проєктів і зацікавленими сторонами, надаючи спільне розуміння фізичної архітектури. Ця чіткість особливо цінна в середовищах спільної роботи, де в процесі розгортання можуть бути залучені кілька команд.

Крім того, наявність чіткого зображення архітектури розгортання підтримує постійне технічне обслуговування та зусилля з усунення несправностей. Це дозволяє розробникам і системним адміністраторам швидко визначати розташування конкретних компонентів і розуміти, як вони взаємодіють один з одним, спрощуючи процес діагностики та вирішення проблем.

На рис. 3.4 зображено діаграму розгортання даної системи. Клієнт-серверна архітектура реалізована на двох окремих серверах.

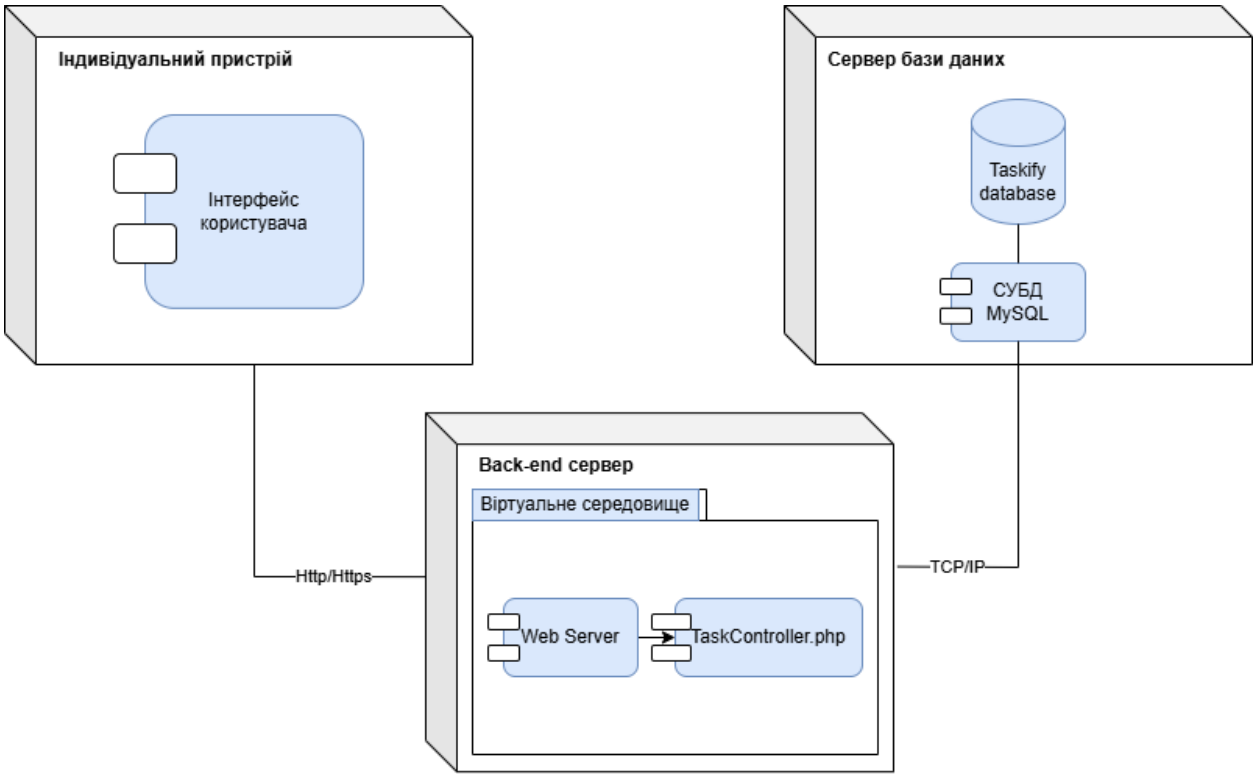

Рис. 3.4 Діаграма розгортання

Апаратні вимоги які необхідні бути у комп’ютера для функціонування програмного забезпечення:

- процесор із принаймні двома ядрами, наприклад Intel Core i3 або AMD Ryzen 3;
- оперативна пам’ять 4 ГБ;
- SSD 10 ГБ або більше.

3.5 Висновок до 3 розділу

У третьому розділі було здійснено проєктування програмної системи, що включає обґрунтований вибір системи управління базами даних, архітектури програмного забезпечення, структури додатку та інструментальних засобів для реалізації системи.

Для збереження та обробки даних обрано реляційну систему керування базами даних MySQL, що забезпечує надійність, масштабованість і сумісність із обраною серверною технологією — Symfony. Архітектура додатку базується на багаторівневому підході з чітким розділенням логіки, представлення та обробки даних. Це сприяє покращенню підтримуваності та розширюваності системи.

У розділі також визначено вимоги до апаратного та програмного забезпечення для стабільної роботи системи, а також описано процес тестування, результати якого підтверджують правильність реалованої функціональності та її відповідність технічним вимогам.

Загалом, реалізоване проєктування створює надійну основу для подальшої експлуатації та розвитку інформаційної системи.

4 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ

4.1 Вибір інструментарію для створення програмного забезпечення

При розробці інформаційної системи для організації та моніторингу процесів управління проєктами вибір відповідних інструментів і технологій має вирішальне значення для забезпечення успіху проєкту. Наступні інструменти були обрані на основі їх можливостей, сумісності та конкретних вимог проєкту.

PHP є основною мовою програмування програми. Він широко використовується для веб-розробки завдяки своїй гнучкості, простоті використання та сильній підтримці спільноти. PHP особливо добре підходить для сценаріїв на стороні сервера, що робить його ідеальним вибором для створення динамічних веб-додатків, які потребують взаємодії з базою даних і введення користувача.

Symfony було обрано як основу для проєкту через його надійні функції, модульність і акцент на найкращих практиках розробки програмного забезпечення. Як зрілий і добре задокументований фреймворк, Symfony забезпечує структуроване середовище, яке сприяє розробці зручних і масштабованих програм. Його вбудовані компоненти, такі як маршрутизація, створення шаблонів і безпека, сприяють швидкій розробці, забезпечуючи високу якість і продуктивність коду.

MySQL була обрана в якості системи управління базами даних для

проєкту через її надійність, продуктивність і простоту використання. MySQL здатний обробляти великі обсяги даних і підтримує складні запити, що робить його придатним для вимог до даних системи управління проєктами. Його природа з відкритим вихідним кодом також сприяє економічній ефективності, дозволяючи команді розробників зосереджувати ресурси на створенні функцій, а не на витратах на ліцензування.

Для розробки інтерфейсу HTML і CSS були обрані як основні технології для структурування та стилізації інтерфейсу користувача. HTML забезпечує необхідну розмітку для створення веб-сторінок, тоді як CSS використовується для покращення візуального представлення та макета. Разом ці технології забезпечують зручність і візуальну привабливість програми, покращуючи загальну взаємодію з користувачем.

Doctrine ORM (Object-Relational Mapping) було інтегровано в проєкт для оптимізації взаємодії з базою даних. Doctrine дозволяє розробникам працювати із записами бази даних як об’єктами PHP, спрощуючи процес запитів і маніпулювання даними. Цей рівень абстракції підвищує продуктивність і допомагає підтримувати чистий і організований код, оскільки розробники можуть зосередитися на бізнес-логіці, а не на запитах SQL.

Composer, інструмент керування залежностями для PHP, був включений для ефективного керування бібліотеками та залежностями проєкту. Composer спрощує процес включення пакетів сторонніх розробників і гарантує, що проєкт залишається в актуальному стані з останніми версіями його залежностей. Цей інструмент пропонує найкращі методи керування кодом і зменшує ризик проблем із сумісністю, полегшуючи підтримку програми з часом.

Таким чином, вибір інструментів для інформаційної системи стратегічно узгоджується з цілями та вимогами проєкту. PHP і Symfony забезпечують міцну основу для створення програми, тоді як MySQL забезпечує надійне керування даними. HTML і CSS покращують інтерфейс користувача, а Doctrine ORM спрощує взаємодію з базою даних. Нарешті, Composer спрощує керування залежностями, сприяючи більш ефективному процесу розробки. Разом ці

інструменти утворюють цілісну технологічну систему, яка підтримує створення надійної та ефективної системи управління проєктами.

4.2 Розробка системи

Розробка інформаційної системи здійснювалася з використанням сучасного стеку веб-технологій, а саме: PHP, фреймворку Symfony, СУБД MySQL, HTML, CSS, JavaScript, а також інтегрованого середовища розробки PHPStorm. Вибір даного інструментарію був зумовлений високою продуктивністю, гнучкістю та активною підтримкою спільноти.

Backend-частина

Серверна частина системи реалізована за допомогою PHP 8.2 з використанням фреймворку Symfony, що дозволило дотримуватися принципів MVC-архітектури, розділяючи логіку, дані та представлення. Усі маршрути, контролери, сервіси та шаблони були структуровані відповідно до стандартів Symfony.

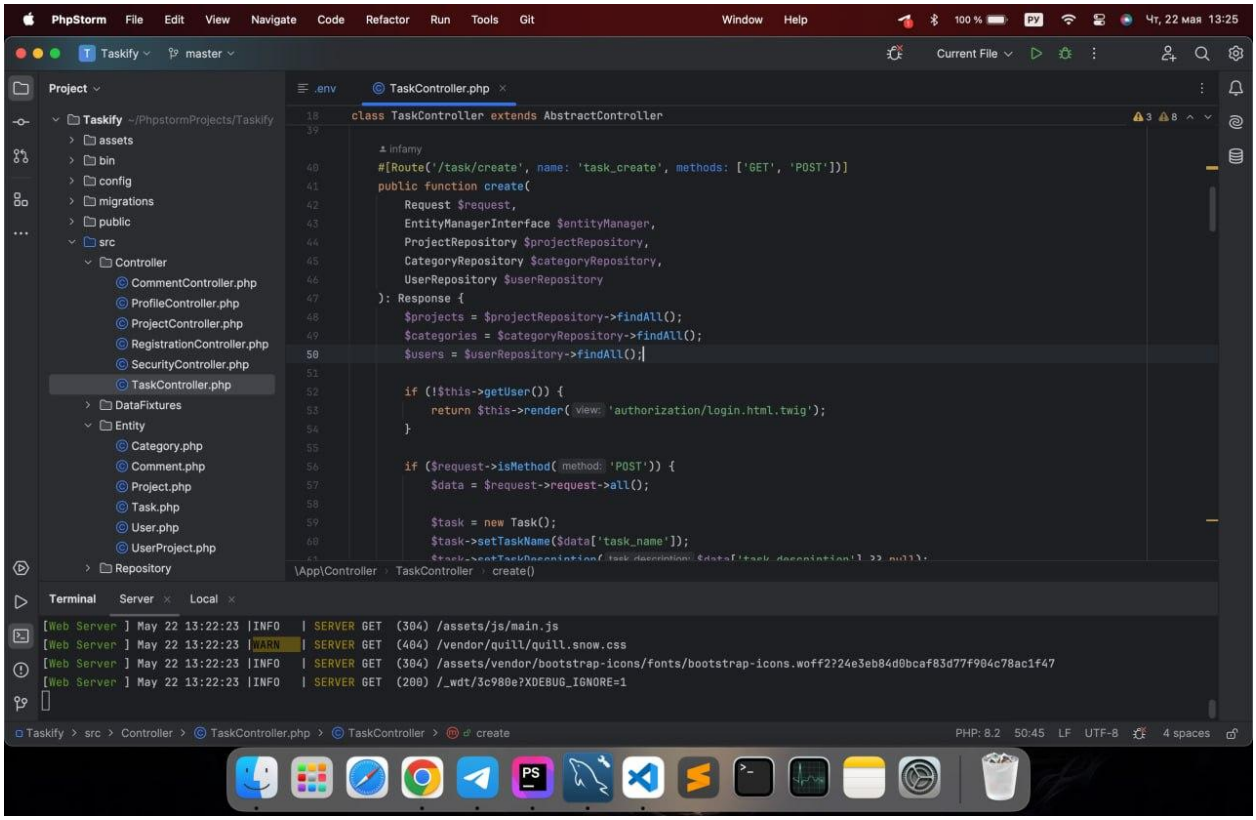

Рис. 4.1 Розробка backend-частини

База даних проєкту спроєктована та реалізована на MySQL, з використанням ORM-бібліотеки Doctrine, яка забезпечує зручну взаємодію між об’єктами PHP та таблицями бази даних. Міграції застосовувалися для керування схемою БД у процесі розробки.

Основні функціональні можливості, які були реалізовані на backend-рівні:

- реєстрація та авторизація користувачів з розмежуванням ролей (наприклад, адміністратор, менеджер, учасник проєкту);
- CRUD-операції для проєктів, завдань, користувачів та ролей;
- пошук, фільтрація та сортування елементів;
- збереження файлів, оновлення статусів, ведення журналу дій користувачів;
- обробка форм з валідацією введених даних.

Frontend-частина

Користувацький інтерфейс реалізований із застосуванням HTML5, CSS3 та JavaScript, з фокусом на зручність, адаптивність і швидкість взаємодії. Дизайн побудований за принципами responsive design, що гарантує коректне відображення інтерфейсу на різних пристроях — від мобільних телефонів до настільних ПК.

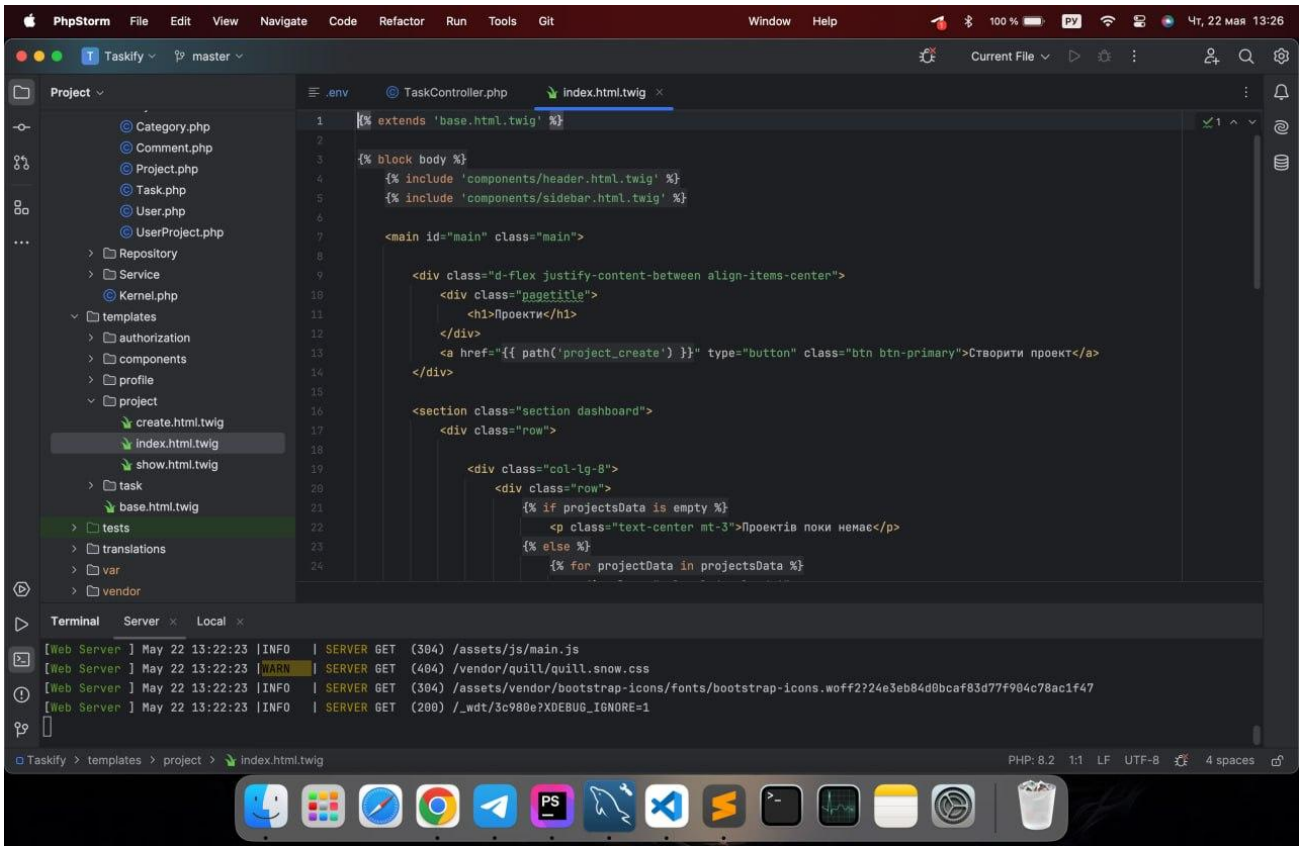

Рис. 4.2 Розробка frontend-частини

Використовувалися такі підходи:

- Компонентна верстка з повторно використовуваними елементами (наприклад, модальні вікна, таблиці, картки);
- Клієнтська валідація форм;
- Інтерактивність через JavaScript, зокрема динамічне оновлення списків, підказки, сповіщення;
- Флеш-повідомлення для зворотного зв'язку з користувачем.
- Інструменти розробки

Для написання та налагодження коду використовувалося середовище PHPStorm, що забезпечувало високу продуктивність за рахунок інтелектуального автозаповнення, інтеграції з Git, підтримки тестування та вбудованого відладчика. Усі версії коду контролювалися за допомогою системи контролю версій Git.

Тестування та налагодження

У процесі розробки проводилось поетапне тестування функціональності, включно з модульним тестуванням окремих функцій, ручним тестуванням через браузер для перевірки UI, перевіркою валідності форм, доступності даних та коректності маршрутизації, перевіркою на стійкість до помилок, обробки виключень і недопущення SQL-ін’єкцій.

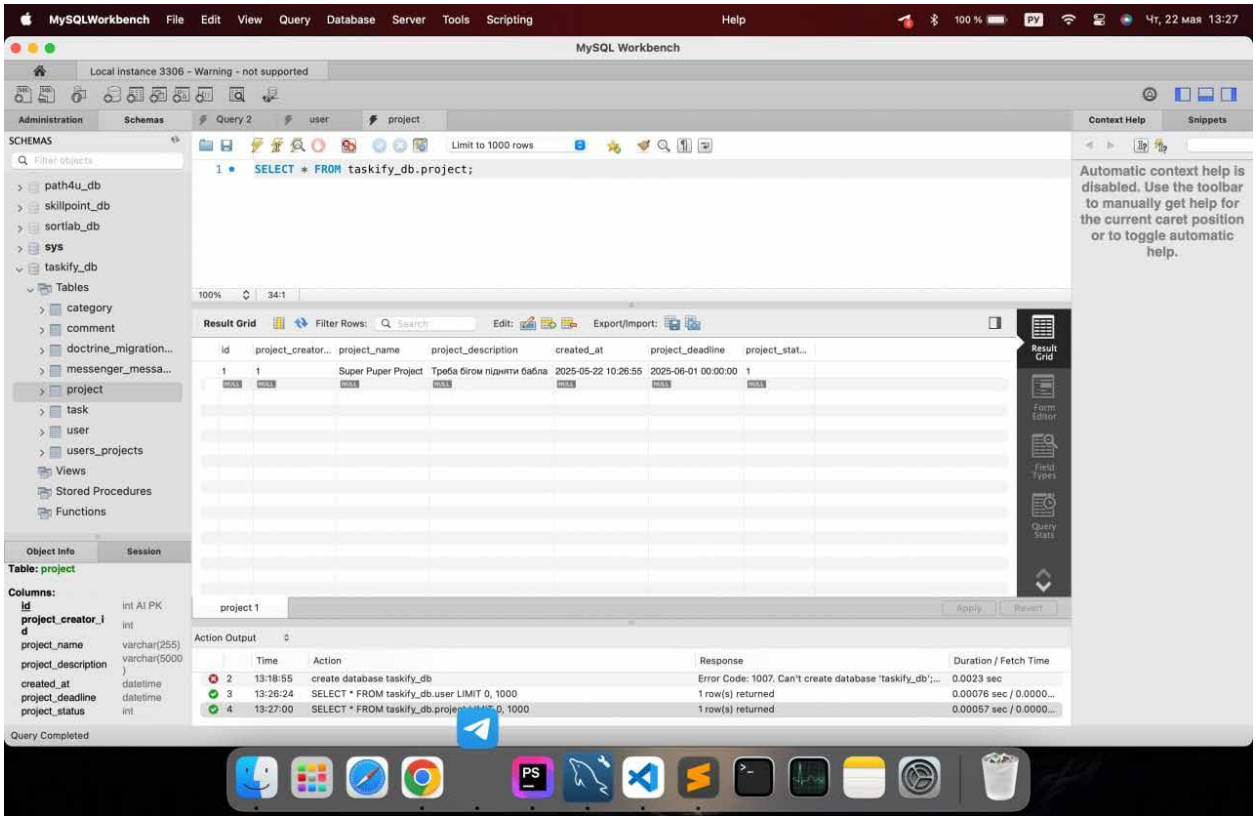

Рис. 4.3 Підключення до СУБД

У результаті проведеної розробки було створено повнофункціональний прототип системи, який відповідає поставленим вимогам, забезпечує зручну взаємодію з користувачем, підтримує гнучке управління проєктами та завданнями.

4.3 Тестування системи

Запустивши систему, бачимо головну сторінку програмного забезпечення (рис. 4.4). На ній ми бачимо список завдань, які нам надані. Ми можемо переглянути активні завдання та виконані завдання. Кожне завдання буде клікабельним і відкривати свою сторінку по натисканню. Так само праворуч

у нас є недавня активність де показуються останні дії пов'язані з авторизованим користувачем.

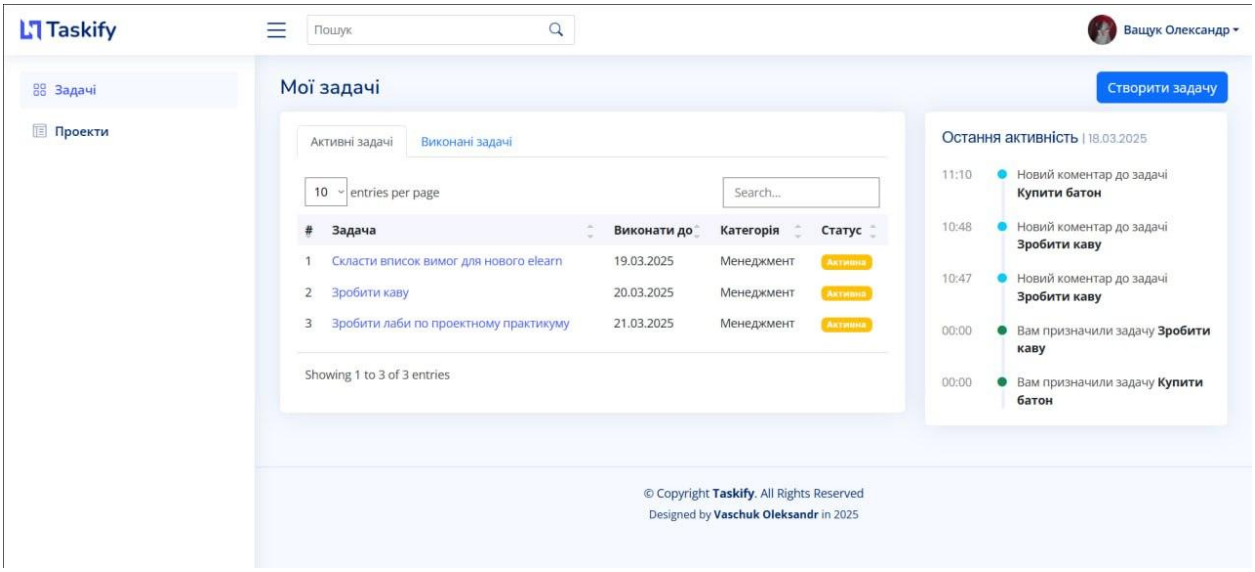

Рис. 4.4 Головна сторінка

Наступним кроком ми переходимо на сторінку проєктів (Рис. 4.5). На ній у нас показані всі проєкти, у яких ми беремо участь, тобто або створили самі, або були додані іншими учасниками. Також є кнопка створення проєкту.

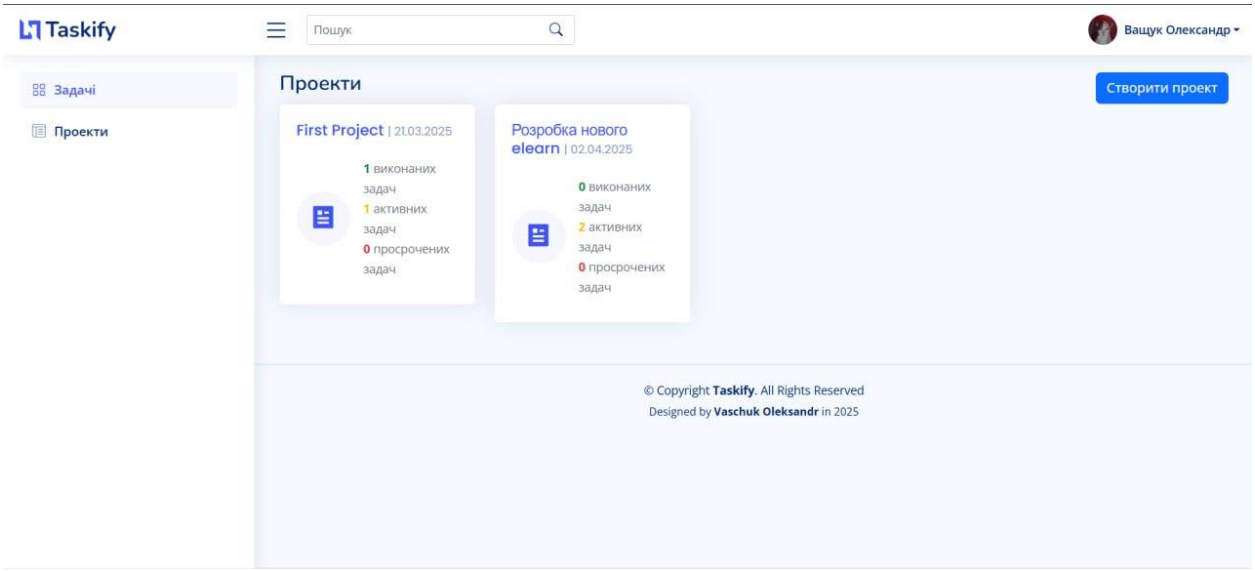

Рис. 4.5 Сторінка “Проєкти”

Спробувавши створити проєкт, ми оформимо форму для створення з усіма необхідними полями для введення (Рис. 4.6).

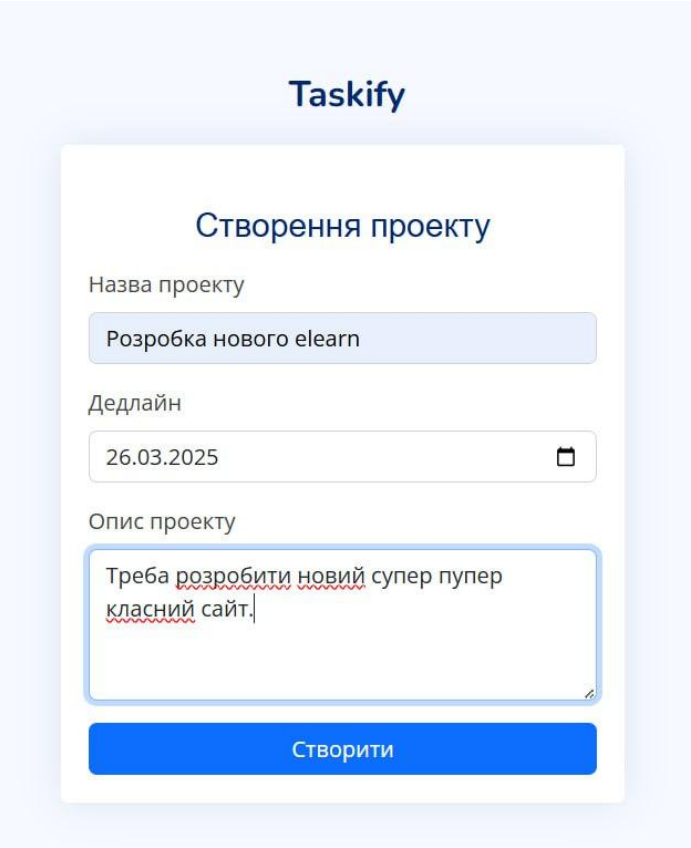

Рис. 4.6 Форма створення нового проєкту

Після створення проєкту можемо відкрити його та перейти на його сторінку. Тут ми маємо списки завдань, створені в цьому проєкті. Завдання фільтруються за активними, виконаними та простроченими (Рис. 4.7). Також у нас є кнопка створення завдання.

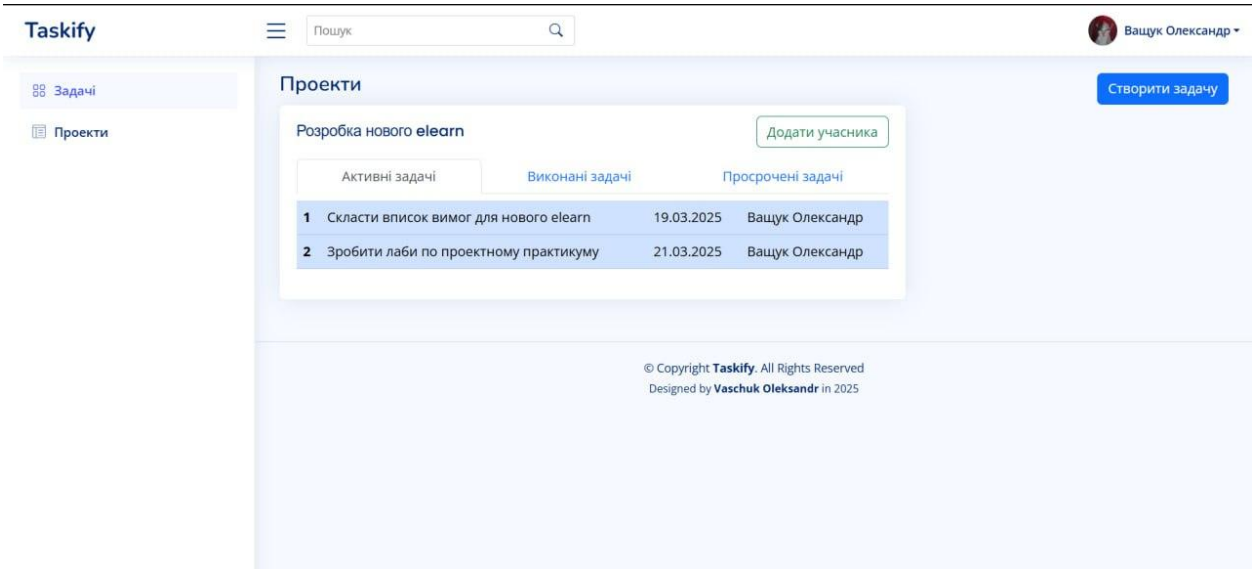

Рис. 4.7 Сторінка з обраним проєктом

Натиснувши кнопку додавання учасника, у нас відкривається модальне вікно з усіма учасниками. Ми можемо скористатися пошуком вгорі і власне додати учасника натиснувши кнопку (Рис. 4.8).

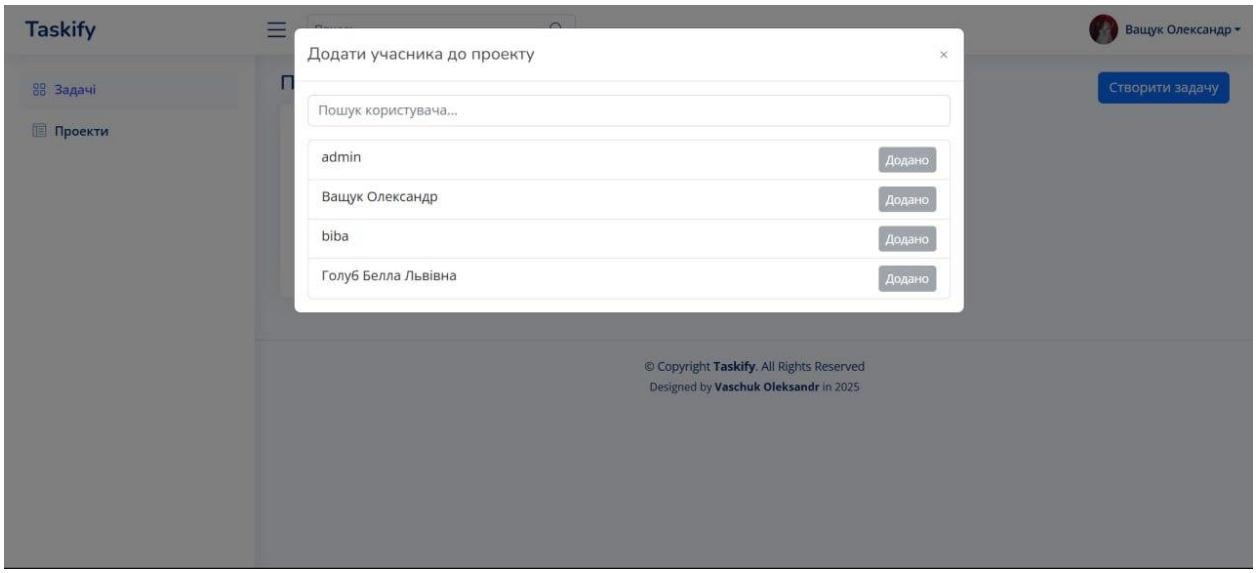

Рис. 4.8 Модальна форма додання учасника до проєкту

Далі перейдемо до створення завдання. Тут у нас є форма для створення завдання з усіма необхідними полями для заповнення. Тут можна вибрати проєкт, вибрати виконавця, категорію завдання тощо (Рис. 4.9).

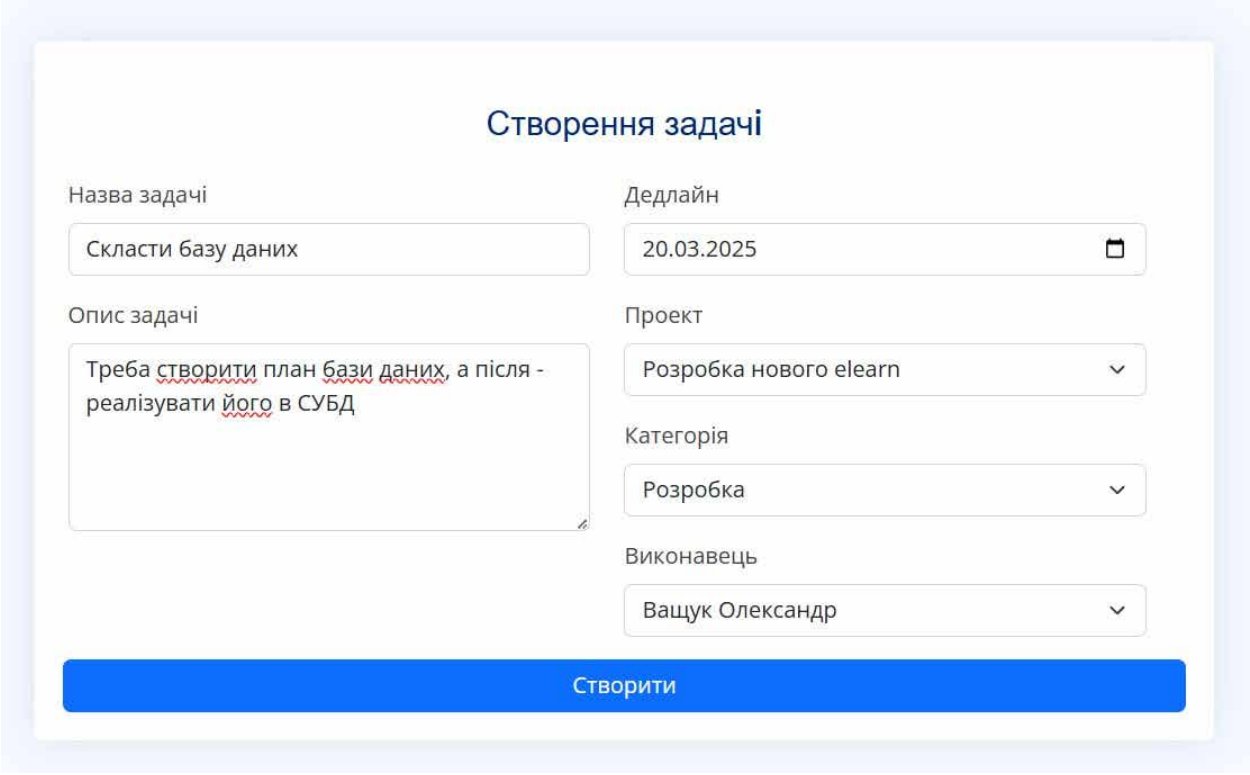

Рис. 4.9 Форма створення задачі

Отже, ми створили завдання, можемо одразу відкрити його та побачимо перед собою сторінку завдання (Рис. 4.10). Тут ми бачимо всі дані про завдання, можемо поміняти дедлайн, залишити коментар, завантажити файл або надіслати відповідь на завдання.

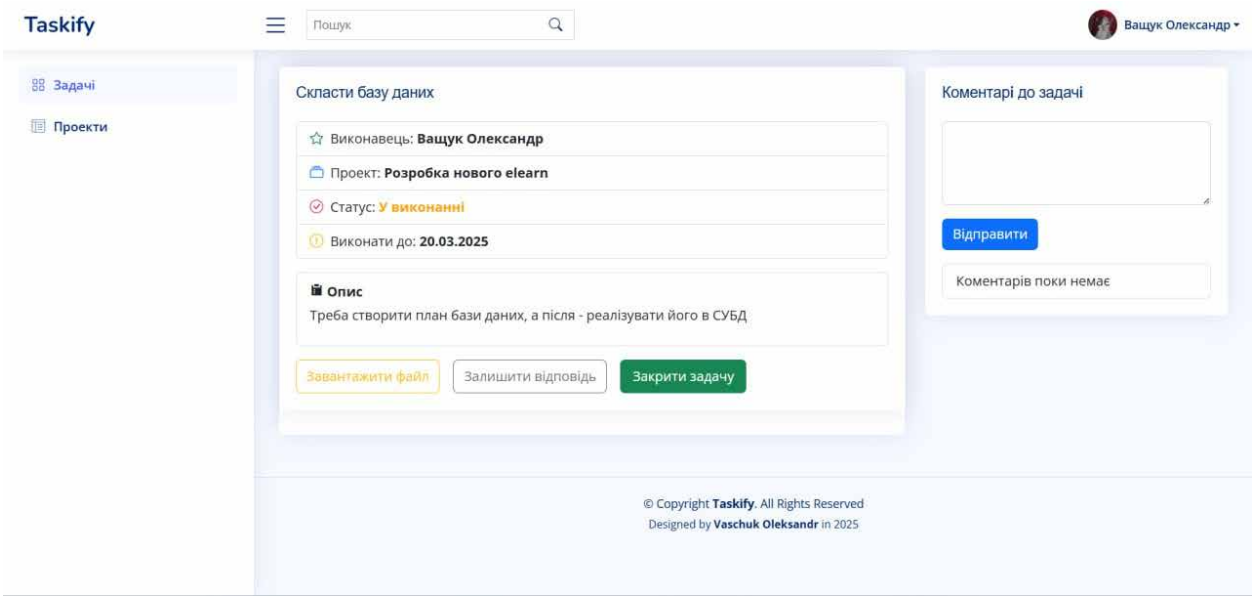

Рис. 4.10 Сторінка задачі

Ми завантажили файл, внесли відповідь до завдання, залишили кілька коментарів та закрили завдання (Рис. 4.11)

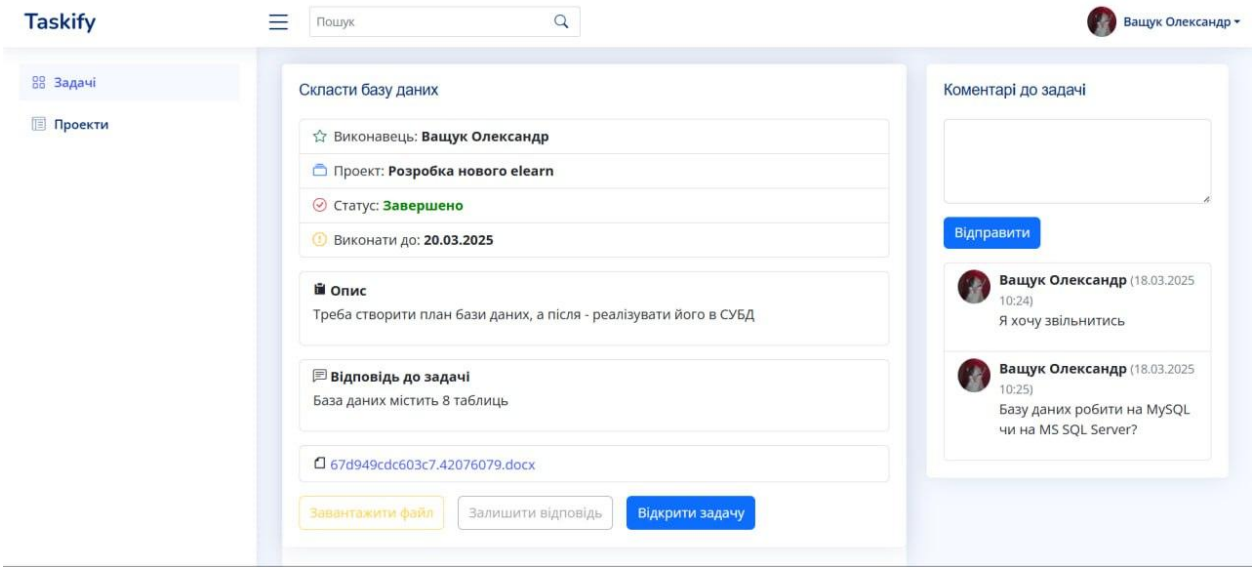

Рис. 4.11 Закрита задача

Також ми можемо редагувати свій профіль. Тут можемо змінити всі особисті дані, змінити фотографію та зберегти нові зміни (Рис. 4.12).

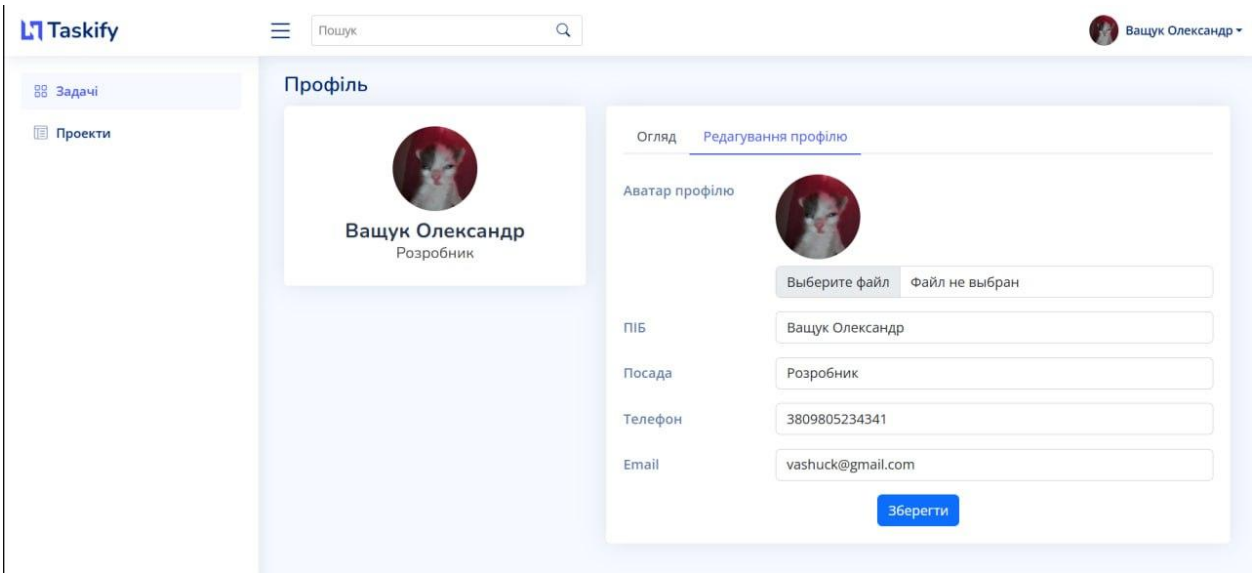

Рис. 4.12 Сторінка редагування профілю

Система має функціонал реєстрації/авторизації, тому у нас представлена форма авторизації в систему (Рис. 4.13).

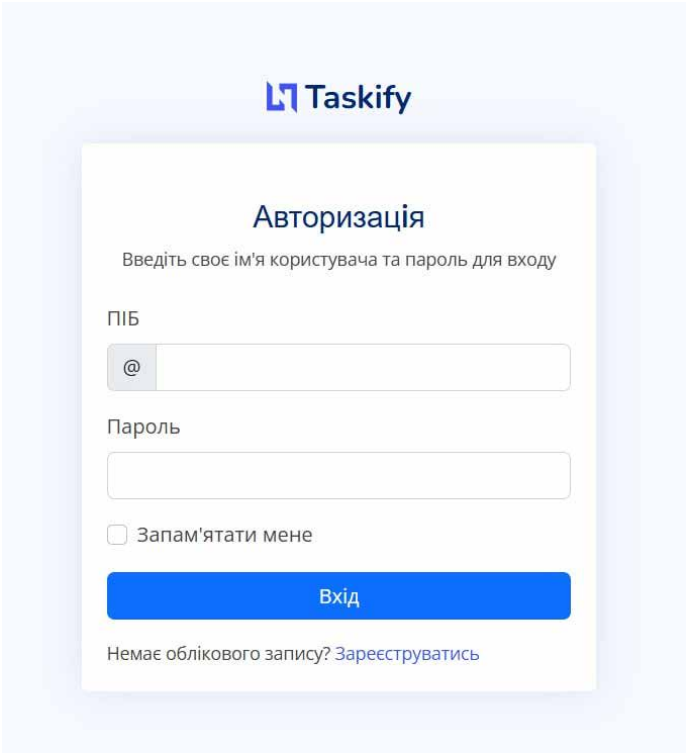

Рис. 4.13 Сторінка авторизації у систему

Ще у цьому проєкті є функціонал пошуку завдань. Хедер сайту має форму для введення тексту. Якщо ввести якийсь текст, то спрацює пошук за назвою задачі (Рис. 4.14).

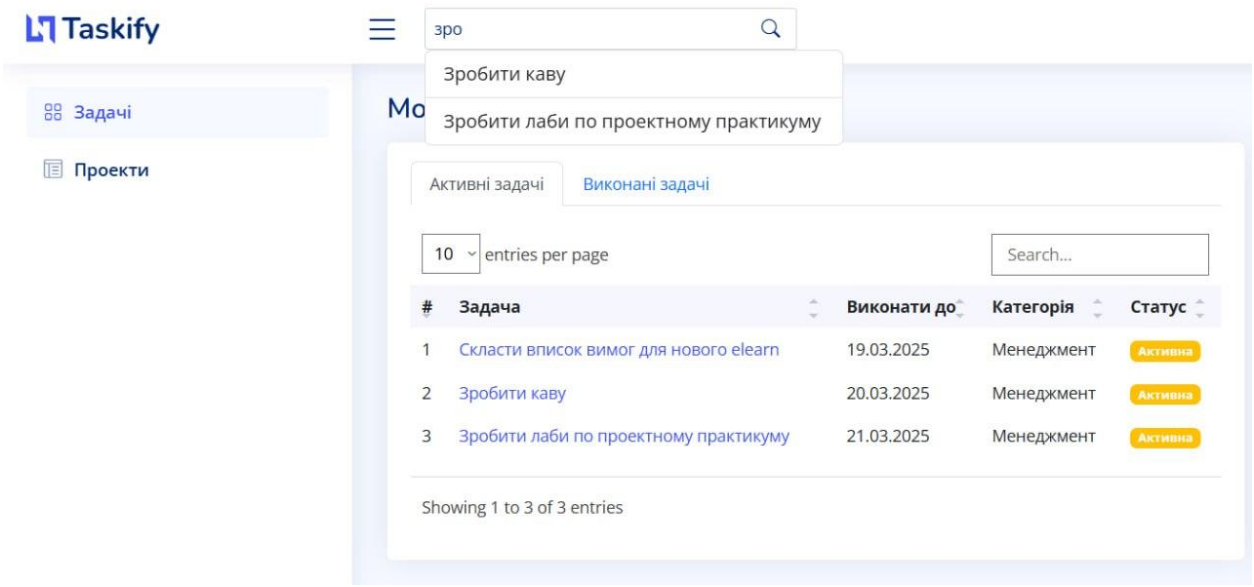

Рис. 4.14 Функціонал пошуку задач

4.3 Висновок до 4 розділу

У цьому розділі було здійснено безпосередню реалізацію інформаційної системи для організації та моніторингу процесів у проєктному менеджменті. На першому етапі було обґрунтовано вибір інструментів для розробки програмного забезпечення. Основу технологічного стеку склали мова PHP із фреймворком Symfony, система управління базами даних MySQL, ORM-бібліотека Doctrine, а також технології HTML, CSS і JavaScript для створення інтерфейсу користувача.

У процесі розробки було створено повноцінну веб-систему з можливістю реєстрації користувачів, створення проєктів і завдань, призначення виконавців, контролю дедлайнів та моніторингу статусу виконання. Забезпечено підтримку ролей користувачів і гнучку систему прав доступу.

Було перевірено коректність авторизації, створення і редагування проєктів та завдань, роботу фільтрації й сортування, а також точність збереження даних у базі. Результати тестування підтвердили стабільність і працездатність системи відповідно до поставлених вимог

ВИСНОВКИ

Підсумовуючи, розробка інформаційної системи для організації та моніторингу процесів управління проєктами є значним прогресом у тому, як команди керують проєктами та ефективно співпрацюють. Використовуючи сучасні технології та добре структуровану архітектуру, система розроблена для підвищення продуктивності, оптимізації робочих процесів і покращення спілкування між членами команди.

Під час проєкту були прийняті ключові рішення щодо вибору інструментів і технологій, включаючи PHP, Symfony, MySQL, HTML, CSS, Doctrine ORM і Composer. Ці інструменти були обрані через їхні можливості, надійність і сумісність, що гарантує надійність системи, її масштабованість і легкість в обслуговуванні. Використання клієнт-серверної архітектури додатково підтримує продуктивність і безпеку системи, забезпечуючи ефективне керування даними та одночасний доступ кільком користувачам.

Конструкція системи включає основні функції, такі як керування завданнями, відстеження проєктів, керування користувачами та інструменти для співпраці, які є життєво важливими для ефективного виконання проєкту. Забезпечуючи зручний інтерфейс і інтуїтивно зрозумілу навігацію, інформаційна система дозволяє користувачам легко керувати своїми проєктами, відстежувати прогрес і ефективно спілкуватися з членами команди.

Крім того, впровадження цілісності даних і заходів безпеки гарантує, що конфіденційна інформація захищена, а система залишається надійною та надійною. Використання системи управління реляційною базою даних, зокрема MySQL, сприяє ефективній обробці даних і підтримує складні запити, необхідні для управління проєктом.

Оскільки організації все більше покладаються на цифрові рішення для вдосконалення своєї діяльності, ця інформаційна система виділяється як цінний інструмент для вдосконалення процесів управління проєктами. Він не тільки

вирішує поточні виклики, з якими стикаються команди, але й позиціонує себе для майбутніх удосконалень і масштабованості в міру розвитку потреб організації.

Підводячи підсумок, можна сказати, що ця інформаційна система для організації та моніторингу процесів у управлінні проєктами готова принести значні переваги користувачам шляхом підвищення ефективності, співпраці та загальних результатів проєкту. Ретельний вибір інструментів, продуманий дизайн і увага до взаємодії з користувачем сприятимуть успіху та зручності використання, зрештою допомагаючи командам ефективніше досягати цілей проєкту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Jira – [Електронний ресурс] – Режим доступу: https://www.atlassian.com/?gad_source=1&gclid=Cj0KCQjws-S-BhD2ARIsALssG0YG2D8DWEII2WLQsL6vVpYqiD6ocLMoWUqwnxx9fwLaqauMZrQ0u8EaAkPPEALw_wcB

2. Asana – [Електронний ресурс] – Режим доступу: https://asana.com

3. Багаторівнева архітектура – [Електронний ресурс] – Режим доступу: https://simpleone.ru/glossary/mnogourovnevaya-arhitektura/

4. Типи архітектури програмного забезпечення – [Електронний ресурс] – Режим доступу: https://medium.com/nuances-of-programming/4-типа-архитектуры-программного-обеспечения-917133174724

5. What is Unified Modeling Language (UML)? – [Електронний ресурс] – Режим доступу: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/

6. Object Diagram in UML: Definition & Examples – [Електронний ресурс] – Режим доступу: https://study.com/academy/lesson/object-diagram-in-uml-definition-examples.html

7. Logical Data Model. Dataedo – [Електронний ресурс] – Режим доступу: https://dataedo.com/kb/data-glossary/logical-data-model

8. Logical Data Model. Techopedia – [Електронний ресурс] – Режим доступу: https://www.techopedia.com/definition/23969/logical-data-model

9. Microsoft Docs. (2021). Layered architecture pattern – [Електронний ресурс] – Режим доступу: https://docs.microsoft.com/en-us/azure/architecture/patterns/layered

10. What is Component Diagram? – [Електронний ресурс] – Режим доступу: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-component-diagram/

11. Deployment Diagram in UML: Definition, Examples & Components – [Електронний ресурс] – Режим доступу: https://study.com/academy/lesson/deployment-diagram-in-uml-definition-examples-components.html

12. What is a data flow diagram? – [Електронний ресурс] – Режим доступу: https://www.lucidchart.com/pages/data-flow-diagram

13. Діаграма варіантів використання (UseCase diagram) – [Електронний ресурс] – Режим доступу: https://flexberry.github.io/ru/fd_use-case-diagram.html

14. UML-діаграми класів – [Електронний ресурс] – Режим доступу: https://prog-cpp.ru/uml-classes/

15. ПРОЄКТУВАННЯ ER-ДІАГРАМИ – [Електронний ресурс] – Режим доступу: https://nationalteam.worldskills.ru/skills/proektirovanie-er-diagrammy/

16. Моделювання діаграми пакетів – [Електронний ресурс] – Режим доступу: https://www.wikiwand.com/uk/Діаграма_пакетів

17. Діаграма розгортання – [Електронний ресурс] – Режим доступу: https://studfile.net/preview/5010027/page:6/

18. СУБД MS SQL Server – [Електронний ресурс] – Режим доступу: https://learn.microsoft.com/ru-ru/sql/relational-databases/lesson-1-connecting-to-the-database-engine?view=sql-server-ver16

19. Амблер, С. В. (2012). «Гнучке моделювання: ефективні методи екстремального програмування та уніфікованого процесу». Wiley.

20. Бек К. та Андрес К. (2004). «Пояснення екстремального програмування: сприймайте зміни». Аддісон-Уеслі.

21. Бок, Г.-В., Кім, Дж. (2012). «Порушення мовчання: роль участі користувачів у розробці системи». Журнал інформаційних технологій, 27 (1), 42-54.

22. Кон, М. (2004). «Застосування історій користувачів: для гнучкої розробки програмного забезпечення». Аддісон-Уеслі.

23. Dingsøyr, T., & Sjøberg, D.I.K. (2013). "Гнучка розробка програмного забезпечення: вичерпний посібник". Спрингер.

24. Друкер, П. Ф. (2006). «Ефективний керівник: повний посібник із виконання потрібних справ». HarperBusiness.

25. Гессе, С., і Мьоллер, Д. (2018). «Управління проєктами та гнучкі методи». Journal of Business Economics, 88(4), 589-620.

26. Хайсміт, Дж. (2009). «Гнучке управління проєктами: створення інноваційних продуктів». Аддісон-Уеслі.

27. Ліч, Л. П. (2005). «Управління проєктами критичного ланцюга». Дім Артех.

28. McKinsey & Company. (2015). «Майбутнє управління проєктами». McKinsey & Company Insights. Отримано з https://www.mckinsey.com/industries/engineering-construction/our-insights/the-future-of-project-management

29. Швальбе, К. (2019). «Управління проєктами інформаційних технологій». Cengage Learning.

30. Соммервіль, І. (2016). «Інженерія програмного забезпечення». Пірсон.

31. Перша версія. (2020). «Звіт про стан Agile 2020». Отримано з https://www.versionone.com/state-of-agile/

32. Висоцький Р. К. (2014). «Ефективне управління проєктами: традиційне, спритне, екстремальне». Wiley.

33. Чжан, Х., і Ван, Х. (2015). «Проєктування інформаційної системи управління проєктами: кейс». Міжнародний журнал управління проєктами, 33 (4), 856-865.