МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки та програмної інженерії
Кафедра інженерії програмного забезпечення

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувача кафедри
Катерина НЕСТЕРЕНКО
“____”_____2024 р.

КВАЛІФІКАЦІЙНА РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ
“БАКАЛАВР”

Тема: Програмний застосунок керування задачами ІТ - проекту

Виконавець: Грищенко Дмитро Валентинович

Керівник: Гололобов Дмитро Олександрович

Нормоконтролер: Варнавський В’ячеслав Володимирович

Київ 2024

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки та програмної інженерії

Кафедра інженерії програмного забезпечення

Освітній ступінь бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ

Завідувача кафедри

Катерина НЕСТЕРЕНКО

“___” _____ 2024 р

ЗАВДАННЯ
на виконання кваліфікаційної роботи студента
Грищенко Дмитро Валентинович

1. Тема дипломної роботи: «Програмний застосунок керування задачами IT – проекту»

затверджена наказом ректора від 8.12.2023 р. № 2483/ст.

2. Термін виконання проекту: з 03.01.2023 р. до 28.02.2024 р
3. Вихідні данні до проекту: програмний продукт розробити у вигляді інформаційної системи.
4. Зміст пояснювальної записки:
    1. Аналіз проблеми взаємодії з інформаційними системами керування задач.
    2. Вимоги до програмного застосунку керування задачами IT – проекту.
    3. Прототип програмного застосунку керування задачами IT – проекту.
5. Перелік обов'язкових слайдів презентації:
    1. Аналіз аналогів застосунку
    2. Вимоги до застосунку
    3. Структура застосунку
    4. Реалізація застосунку

6. Календарний план-графік

№ пор.	Завдання	Термін виконання	Відмітка про виконання
1.	Ознайомлення з постановкою задачі та вивчення літератури. Складання графіку роботи	03.01.24 – 09.01.24	
2.	Написання 1 розділу, представлення керівнику	10.01.22 – 17.01.22	
3.	Написання 2 розділу, представлення керівнику	18.01.22 – 22.01.22	
4.	Написання 3 розділу, представлення керівнику	23.01.22 – 26.01.22	
5.	Загальне редагування та друк пояснювальної записки, графічного матеріалу	27.01.22 – 31.01.22	
6.	Проходження нормо-контролю, перепліт пояснювальної записки. Отримання відгуку керівника	01.02.22 – 09.02.22	
7.	Розробка тексту доповіді. Оформлення графічного матеріалу для презентації	01.02.22 – 05.02.22	
8.	Отримання рецензії.	05.02.22 – 09.02.22	
9.	Підготовка матеріалів для передачі секретарю ДЕК (ПЗ, ГМ, CD-R з електронними копіями ПЗ, ГМ, презентації, відгук керівника, рецензія, довідка про успішність, 2 папки, 2 конверта)	23.02.22 – 28.02.22	

7. Дата видачі завдання 03.01.2024

Керівник: Дмитро ГОЛОЛОБОВ

Завдання прийняв до виконання: Дмитро ГРИЩЕНКО

Дата

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи «Програмний застосунок керування задачами IT – проекту »: 52 с., 30 рис., 2 лістингу., 15 літературних джерел.

ЗБЕРЕЖЕННЯ, БАЗА ДАННИХ, ПЛАНУВАННЯ, ПРОЕКТ, ФОРМУЛЮВАННЯ ЗАДАЧ, ТЕСТУВАННЯ, МОНІТОРИНГ ЗАВДАНЬ

Об’єкт розробки – застосунок керування задачами IT – проекту

Мета кваліфікаційної роботи – розробити програмний застосунок керування задачами IT – проекту.

ABSTRACT

Explanatory note to the qualification work "Software application for managing IT tasks - project": 52 pages, 30 figures, 2 listings, 15 literary sources.

STORAGE, DATABASE, PLANNING, PROJECT, FORMULATION OF TASKS, TESTING, MONITORING OF TASKS

The object of the study – is an IT project task management application

The purpose of the qualification work – is to develop a software application for managing IT project tasks.

ЗМІСТ

ВСТУП

В сучасному інформаційному суспільстві, де висока конкуренція та стрімкий технологічний прогрес стають невід'ємною частиною підприємницького середовища, ефективне управління ІТ – проектами набуває визначального значення. Розвиток програмних застосунків, спрямованих на керування завданнями в масштабі проектів інформаційних технологій, визначається потребою в надійних інструментах для планування, виконання та моніторингу завдань.

Даний дипломний проект спрямований на дослідження та розробку програмного застосунку, спеціалізованого у керуванні задачами ІТ – проекту. Обґрунтування актуальності цієї теми полягає в тому, що швидкі та ефективні рішення в галузі управління завданнями стають стратегічною перевагою для організацій, що здійснюють розробку та впровадження інформаційних технологій.

Аналіз сучасного стану справ у сфері управління ІТ – проектами виявляє необхідність розробки інтелектуальних інструментів, спроможних оптимізувати процеси планування, контролю та аналізу завдань, що стоїть перед високотехнологічними підприємствами. Цей диплом докладатиме зусиль для створення програмного застосунку, який враховуватиме специфіку ІТ – проектів та сприятиме підвищенню продуктивності розробки, зменшенню ризиків та поліпшенню загальної якості управління завданнями в цьому контексті.

Мета кваліфікаційної роботи полягає у вивченні, розробці та реалізації програмного застосунку для ефективного керування завданнями в контексті ІТ –

проектів. Головною метою є створення інтелектуального інструменту, який дозволить підвищити продуктивність управління завданнями, спростити планування та моніторинг робочих процесів, а також забезпечити зручну взаємодію між учасниками проекту.

РОЗДІЛ 1
АНАЛІЗ ПРОБЛЕМИ ВЗАЄМОДІЇ З ЗАСТОСУНКАМИ ДЛЯ КЕРУВАННЯ ПРОЕКТАМИ

1.1. Аналіз існуючих застосунків для керування проектами.

У сучасному динамічному середовищі розробки програмного забезпечення, великі та складні ІТ – проекти нерідко стикаються із завданням ефективного розподілу завдань, управління ресурсами та вчасного виявлення та вирішення ризиків. Застосування наукових підходів до розробки програмного застосунку для керування завданнями ІТ – проектів дозволить не лише оптимізувати робочі процеси, але і підвищити загальну якість продукту.

Обґрунтування важливості відстеження завдань в контексті ІТ-проектів. Упровадження систем відстеження завдань в сфері ІТ – проектів сприяє підвищенню продуктивності та швидкості виконання завдань, забезпечуючи негайний огляд прогресу та можливість прийняття швидких рішень на основі актуальної інформації. Крім того, ефективне розподілення ресурсів та визначення відповідальності для кожного завдання сприяє оптимізації робочих процесів та уникненню перевантажень. Також, відстеження завдань дозволяє точно контролювати терміни виконання та реагувати на зміни в проекті, що допомагає уникнути затримок та дотримуватися дедлайнів.

Типові Проблеми Відстеження Завдань. В сучасному контексті управління ІТ – проектами виникає низка складних завдань, пов'язаних з відстеженням та контролем за завданнями. Традиційні методи, такі як використання електронної пошти або паперових заміток, демонструють свою обмеженість у забезпеченні ефективного контролю та прозорості. Однією з типових проблем є недостатня прозорість в стані виконання завдань.

Застосування традиційних засобів не завжди дозволяє забезпечити чітке визначення та доступ до інформації про прогрес, що може призводити до непорозумінь та невпевненості серед учасників команди. У випадку великих та розподілених команд виникають додаткові труднощі. Кількість завдань і обсяг комунікації можуть робити важким збереження єдності та взаєморозуміння.

Сучасні Рішення для Відстеження Завдань. У зусиллях подолати типові проблеми відстеження завдань, сучасні ІТ – проекти використовують різноманітні інноваційні рішення. Однією з ключових характеристик таких систем є їх здатність забезпечувати реальний час для користувачів. Забезпечення негайного огляду прогресу завдань та актуальної інформації дозволяє командам приймати оперативні рішення та швидко реагувати на зміни у завданнях. Пріоритизація завдань стає більш ефективною, що дозволяє ефективніше розподіляти завдання та визначати їх терміни виконання. Це сприяє уникненню перевантажень та забезпеченню балансу між завданнями та членами команди. З врахуванням часових відмінностей та особливостей роботи великих та розподілених команд, сучасні системи відстеження завдань враховують та пропонують рішення для полегшення комунікації та координації в таких умовах. Системи автоматично оновлюють та переглядають графік завдань при будь-яких змінах, забезпечуючи адаптацію до нових умов і забезпечуючи стабільність проекту.

Виклики та Переваги Використання Сучасних Систем. Не зважаючи на беззаперечні переваги, сучасні системи відстеження завдань також стикаються зі своїми викликами. Одним з основних аспектів, який може вплинути на їх ефективність, є необхідність адаптації до конкретних потреб та характеристик кожного ІТ – проекту. Інтеграція з Іншими Інструментами. Спрощення робочого процесу та підвищення ефективності часто залежать від здатності системи взаємодіяти з іншими інструментами, що використовуються в проекті. Невдала інтеграція може призвести до розбиття робочого потоку та втрати ефективності. Вартість та Виправданість. З високою технологічною

складністю та різнообразністю ринку систем відстеження завдань, виникає питання виправданості витрат на їх впровадження та обслуговування, особливо для менших проектів. Підвищення Ефективності Команди. Використання сучасних систем дозволяє збільшити ефективність команди, полегшуючи взаємодію, покращуючи комунікацію та дозволяючи швидше реагувати на зміни.

Рекомендації. У заснуванні висновків щодо використання сучасних систем відстеження завдань в IT – проектах, важливо враховувати не лише їхні переваги, але й виклики, які можуть виникнути в процесі їхнього впровадження та експлуатації. Рекомендації щодо вибору та впровадження систем відстеження завдань: Адаптація до Потреб Проекту. Вибір системи повинен враховувати конкретні потреби та особливості кожного IT – проекту. Ретельний аналіз та адаптація можуть забезпечити максимальну ефективність використання.

Використання сучасних систем відстеження завдань у сфері IT – проектів принесло численні переваги, такі як підвищення ефективності команд, контроль прогресу та покращення комунікації. Однак важливо розуміти, що їхнє впровадження може супроводжуватися викликами, такими як необхідність інтеграції та виправданість витрат. З урахуванням цих аспектів, важливо обирати систему відстеження завдань, яка оптимально відповідає потребам конкретного проекту, та впроваджувати її з урахуванням всіх факторів для досягнення максимальної результативності та успіху в управлінні IT-проектами.

1.2 Аналіз програмних застосунків для керування задачами IT – проекту.

Jira – це інтегрована платформа управління проектами, розроблена компанією *Atlassian*, яка використовується для планування, відстеження та керування завданнями та проектами в галузі розробки програмного забезпечення та інших сферах. Основна мета *Jira* – забезпечити командам

ефективний інструмент для керування робочими процесами, полегшити спільну роботу та підвищити продуктивність. Система *Jira* пропонує величезний обсяг конфігураційних можливостей, що дозволяє визначати типи завдань для кожної програми, встановлювати окремий робочий процес з унікальним потоком операцій та набором статусів. Із використанням так званих "схем", можна налаштовувати права доступу, поведінку та видимість полів для кожного окремого проекту в системі. *Jira*, завдяки своєму універсальному підходу, може бути адаптована для різних задач, включаючи управління вимогами, управління ризиками та імплементацію невеликих систем, таких як системи бронювання чи автоматизації процесів рекрутингу. Для розробників надаються інструменти для створення розширень – плагінів. Розробники розширень можуть розміщувати свої плагіни для продажу на спеціальному розділі сайту *Atlassian*.

– Управління Завданнями: *Jira* дозволяє створювати, призначати та відстежувати завдання. Кожне завдання може мати призначених учасників, строк виконання та інші атрибути.

– Планування та Відстеження Проектів: Платформа надає інструменти для планування та відстеження проектів, де можна визначити етапи роботи, завдання та ресурси.

– *Agile* – Методології: *Jira* підтримує *Agile* – методології, такі як *Scrum* та *Kanban*, що дозволяє командам ефективно використовувати ці підходи у розробці.

– Звітність та Аналітика: Є вбудовані інструменти для створення звітів та аналізу продуктивності проектів, що допомагає приймати обгрунтовані рішення.

– Інтеграція з Іншими Інструментами: Можливість інтеграції з різними інструментами розробки, такими як *Bitbucket, Confluence, GitHub* та інші, для покращення комунікації та обміну інформацією.

– Можливості Розширення: Є велика кількість плагінів та додаткових розширень, які дозволяють налаштувати та розширити функціональність *Jira* відповідно до конкретних потреб користувача.

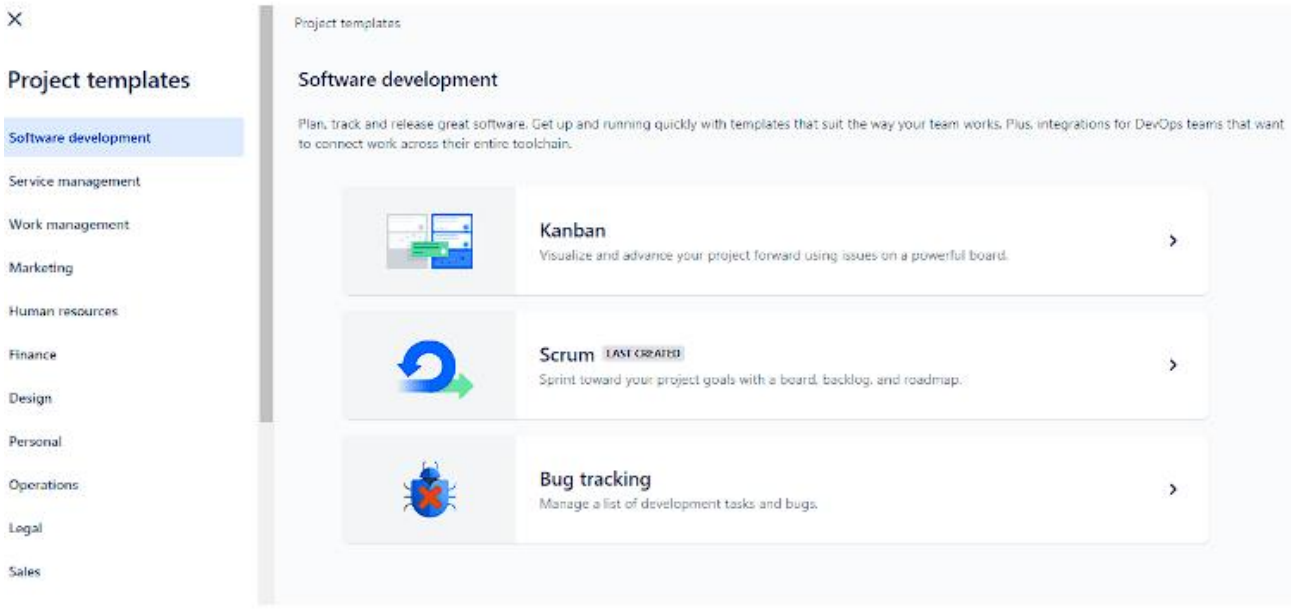

Рис. 1.1. Типи проектів у *Jira*

Початковим кроком у роботі з *Jira* є створення нового проекту в адміністративній частині системи. Платформа пропонує широкий вибір шаблонів, дозволяючи обрати методологію роботи, що найкраще відповідає специфіці кожного проекту.

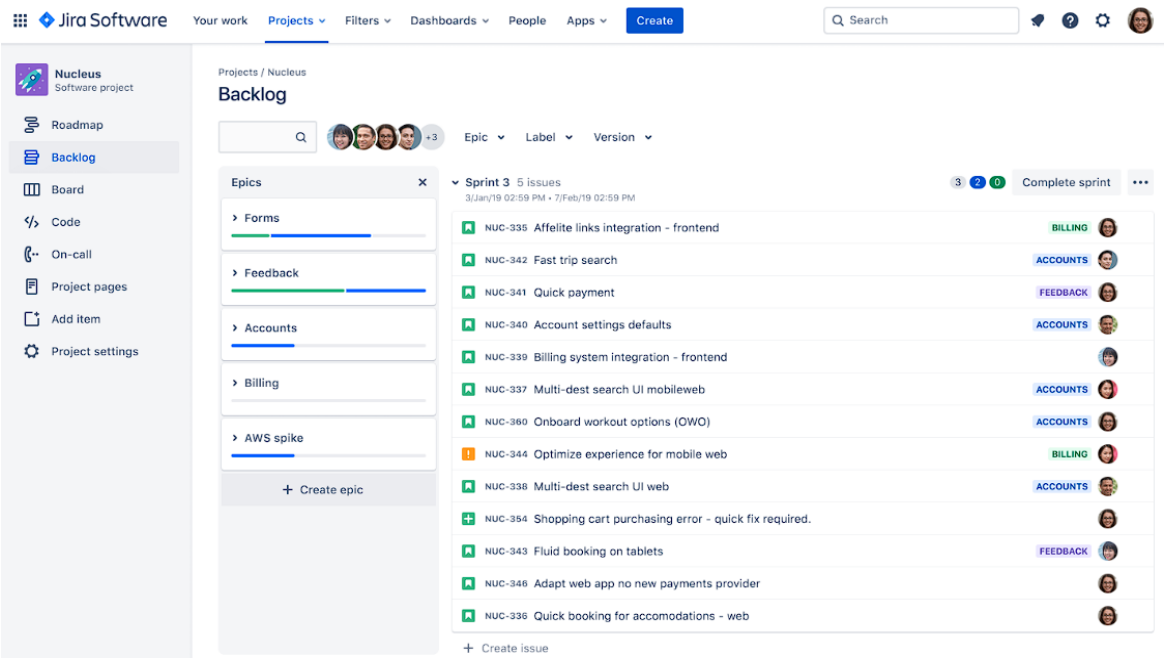

Рис. 1.2. Беклог *Scrum*

У беклогу *Scrum* можна організовувати спринти, наповнювати їх завданнями різних типів, а потім оцінювати ці завдання та виконувати їх сортування за пріоритетом.

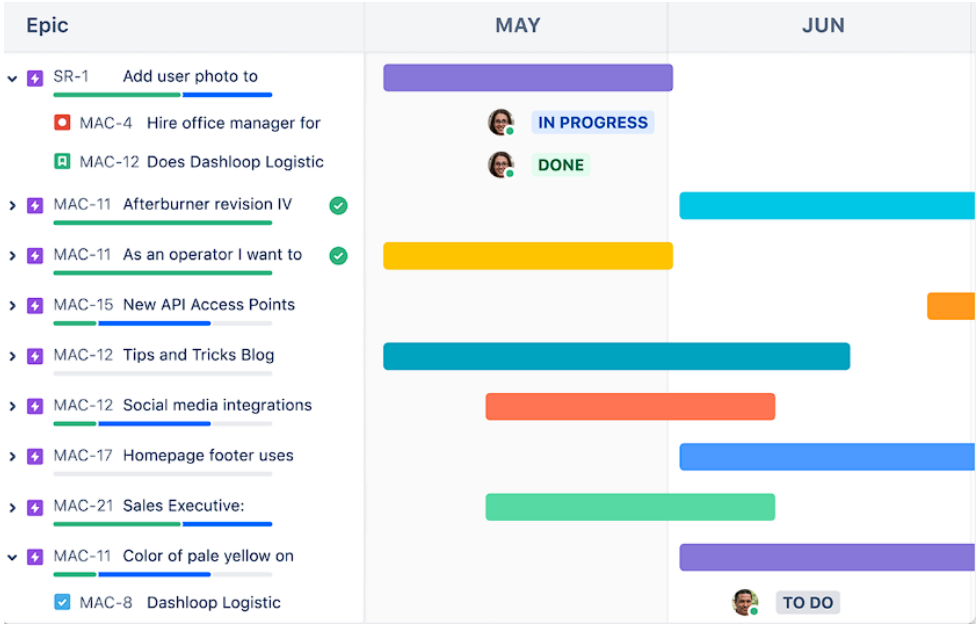

Рис. 1.3. Графік виконання завдань.

Можна створити інтерактивний графік, додавши епопеї та відповідні робочі завдання, визначте залежності та релізи.

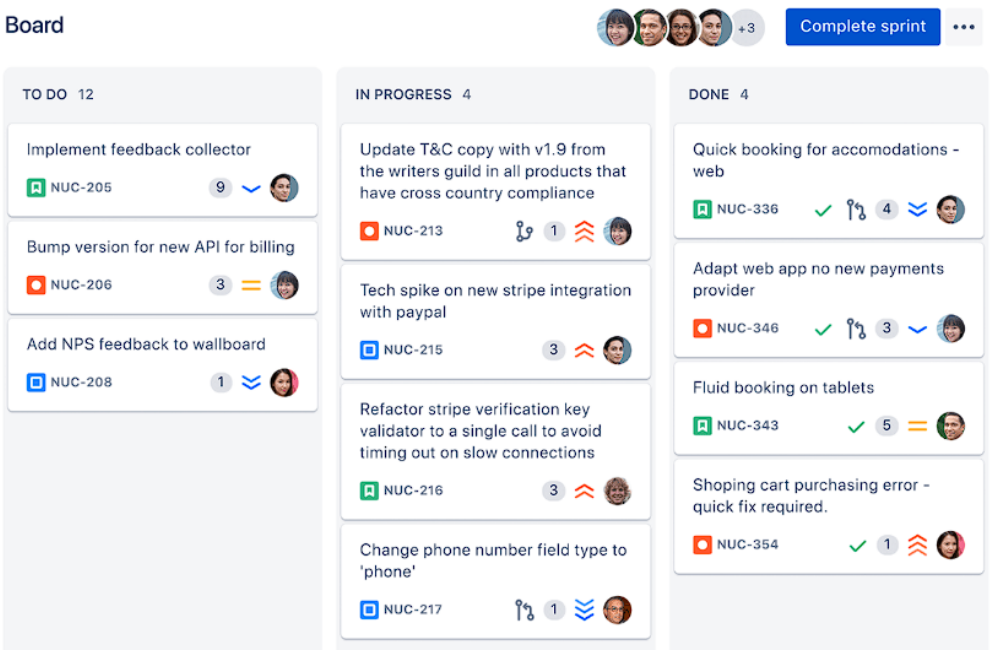

Рис. 1.4. Дошка, щоб розбивати задачі між командами.

Дошки *Scrum* допомагають командам, які відповідають принципам *Agile*, розбивати великі та складні проекти на зручні блоки, щоб вони могли фокусуватися на суттєвому та швидше впроваджувати нові версії протягом спринтів.

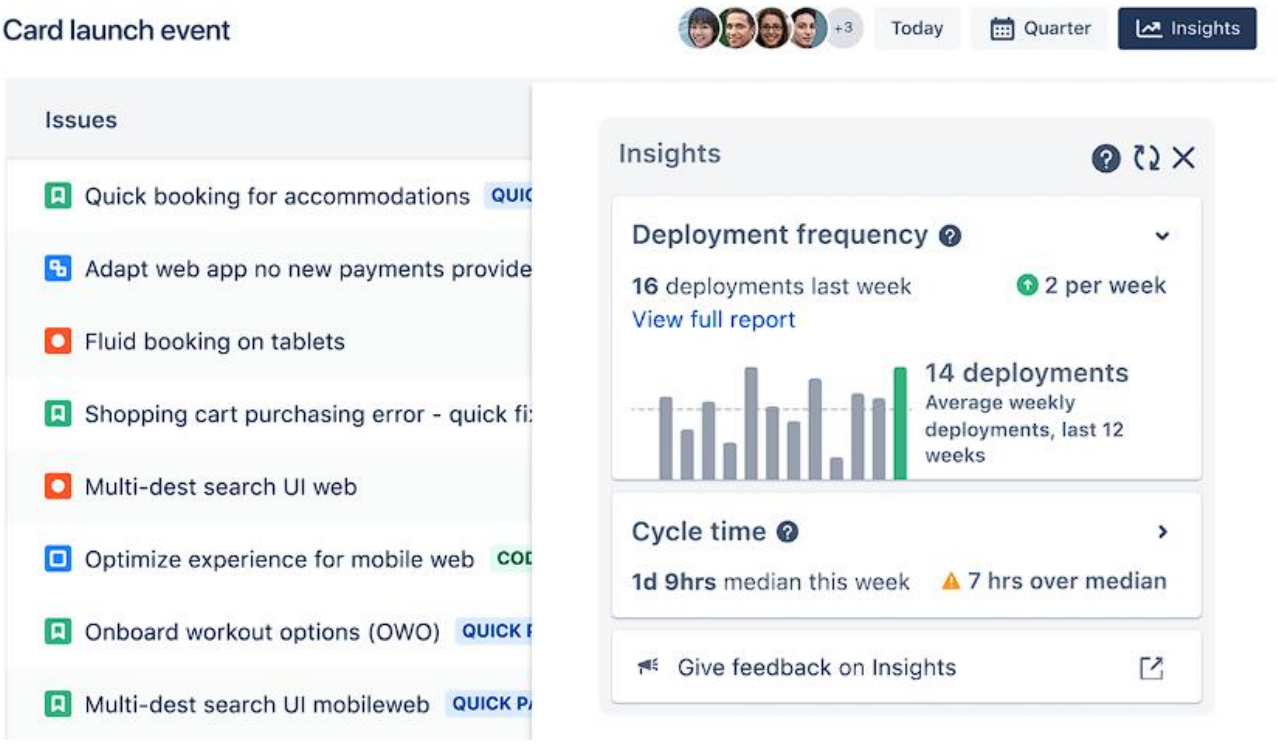

Рис. 1.5. Звіт, щоб перевіряти виконання завдань

Готові звіти та інформаційні панелі в *Jira Software* надають критично важливу інформацію у контексті вашої роботи, дозволяючи командам завжди бути в курсі подій та зберігати налаштованість на успіх.

Незважаючи на широке використання та популярність *Jira*, існують деякі недоліки, які користувачі вказують:

– Складність для Новачків: Інтерфейс *Jira* може виглядати складним для новачків, і вивчення всіх функцій може зайняти час.

– Вартість Ліцензії: Вартість ліцензій для *Jira* може бути високою, особливо для великих команд та підприємств, що може бути фактором обмеження для деяких організацій.

– Потреба у Додаткових Розширеннях: Для задоволення певних потреб чи вимог команди може виникати необхідність у встановленні додаткових плагінів чи розширень, що може збільшувати складність системи.

– Обмежена Гнучкість для Нестандартних Процесів: Деякі команди можуть відзначити, що *Jira*, спрямована на роботу за *Agile* – методологіями, може бути менш гнучкою для вирішення нестандартних процесів розробки.

– Збільшена Системна Вимога: Зростаюча кількість функцій та можливостей *Jira* може призводити до збільшення системних вимог, особливо для великих проектів.

– Неефективне Використання для Малих Команд: Деякі користувачі відзначають, що *Jira* може бути надто потужною для малих команд та проектів, і може вимагати додаткового часу на налаштування та адаптацію.

Несприятливі відгуки можуть варіювати в залежності від конкретних потреб та контексту використання. Багато з цих недоліків можуть бути подолані за допомогою правильної конфігурації та вибору відповідних плагінів чи доповнень.

Asana – це інтегрована веб–платформа для управління проектами та завданнями, призначена для полегшення співпраці в командах. Завдяки *Asana*,

користувачі можуть створювати проекти, додавати завдання, встановлювати строки виконання та призначати відповідальних. Платформа сприяє комунікації шляхом коментування завдань та обміну внутрішніми повідомленнями. *Asana* надає можливість створення та відстеження календарів та термінів, дозволяючи ефективно планувати та виконувати завдання вчасно. Кожен користувач може мати доступ до огляду усіх своїх завдань та проектів через особистий кабінет. Крім того, *Asana* інтегрується з іншими популярними інструментами, що дозволяє командам максимально використовувати їхні поточні робочі процеси. Платформа також підтримує мобільні додатки, що дозволяє користувачам взаємодіяти з *Asana* з різних пристроїв.

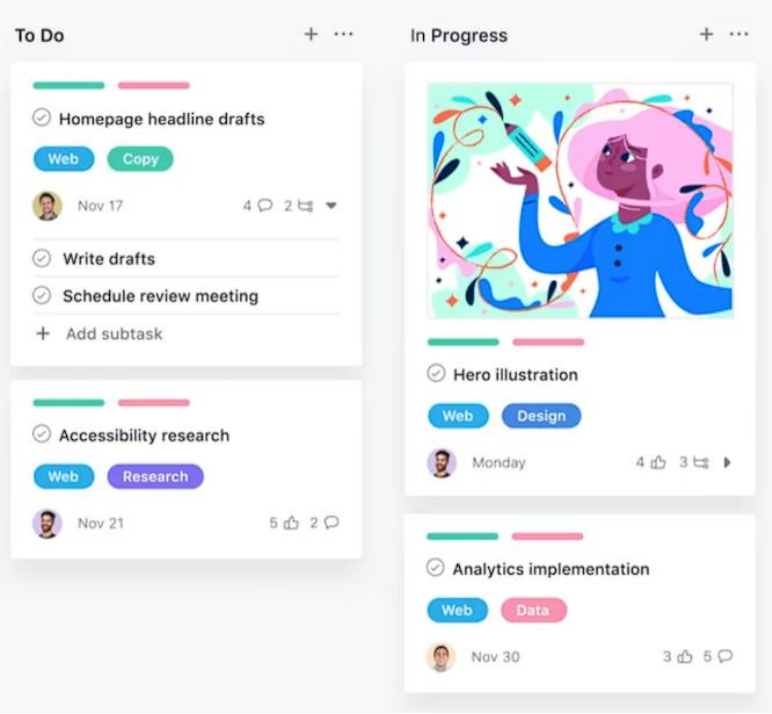

Рис. 1.6. Створення завдання для команди.

Для початку виконання завдання потрібно сформувати технічне завдання та призначити відповідального. Використовуйте функцію "*Boards*" для розділення великого проекту на завдання. Створіть завдання у стовпчику "*To do*" та перемістіть його у стовпчик "*Done*", коли завдання виконано.

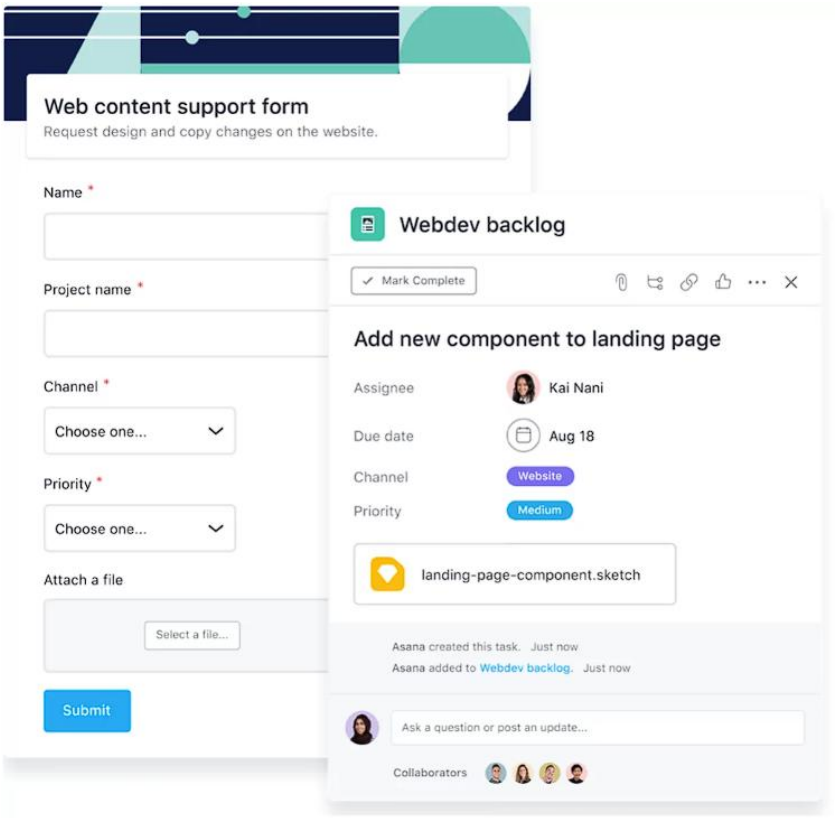

Рис. 1.7. Формування завдань для різних команд.

Для ефективності управління проектами встановіть налаштування форми, яку легко заповнювати для кожного завдання. За допомогою *Asana* ви можете створити таку форму, яка чітко вказує, до якого відділу віднести завдання та визначає його пріоритет та інші деталі, щоб уникнути повторних витрат часу на створення завдань.

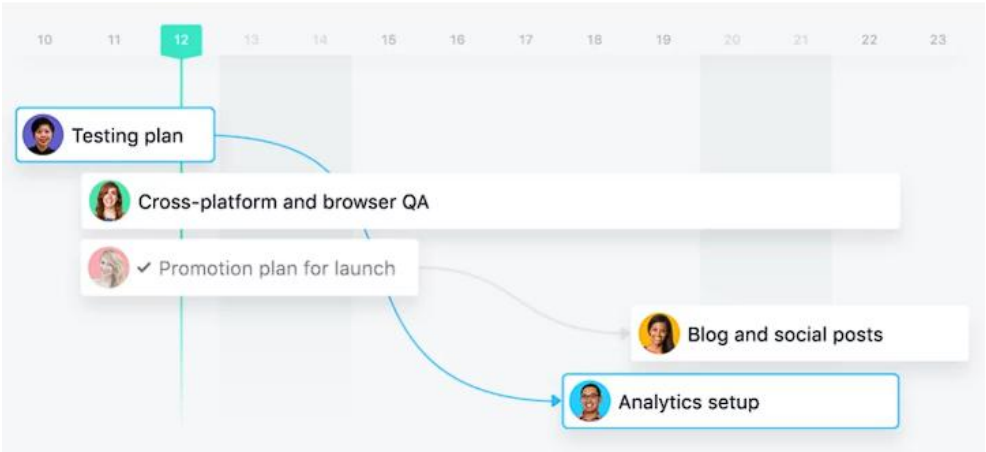

Рис. 1.8. Встановлення дедлайнів для виконання завдання.

Після створення пулу завдань та їх розподілу за дашбордами, третім кроком є встановлення дедлайнів. Використовуйте інструмент *Timeline*, де можна вказати дату завершення та визначити дату початку наступного завдання. Якщо вам потрібно чекати на завершення роботи колеги, *Timeline* відобразить це на графіку, допомагаючи вам установити чіткі дедлайни.

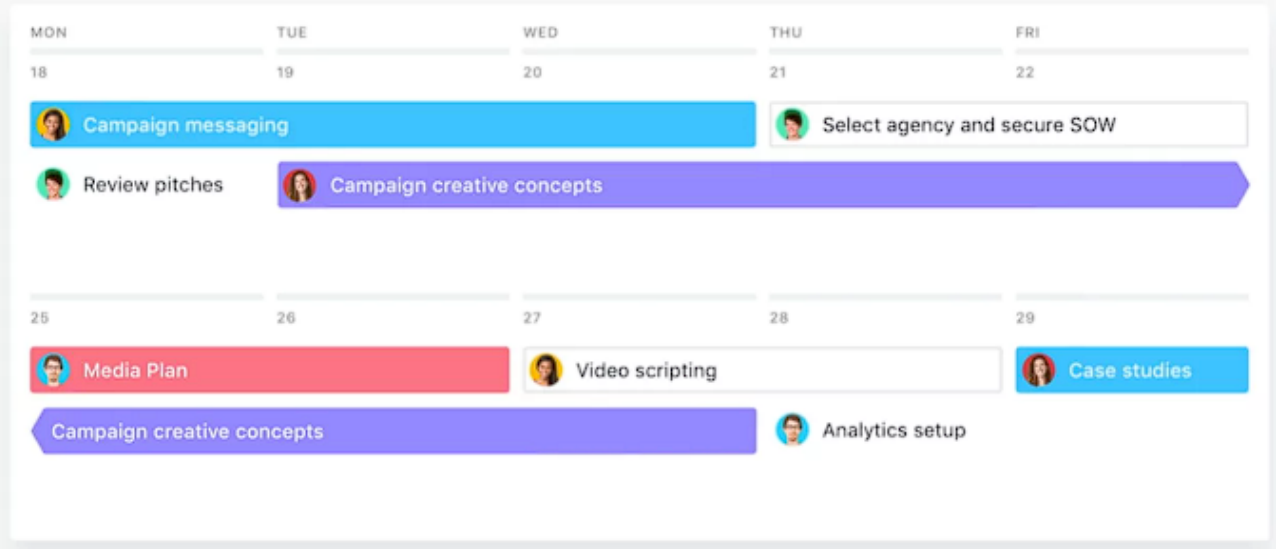

Рис. 1.9. Використання *Calendar*, щоб визначати загруженість команд.

Використовуйте *Calendar*, щоб глобально оцінити події за днями. Це також дозволить вам розрахувати потрібний час для досягнення результату та визначити завантаженість команди.

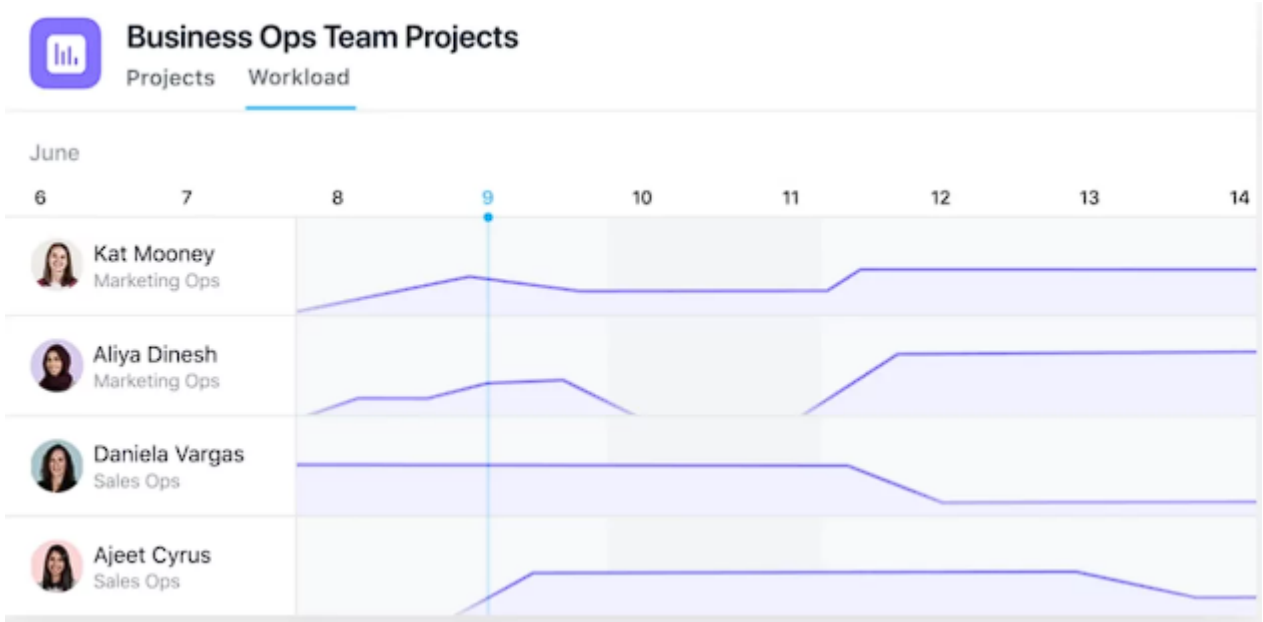

Рис. 1.10. Інструмент *Workload,* для визначення завантаженості кожного члена команди.

Використовуйте інструмент *Workload*, щоб визначити рівень завантаження кожного співробітника. Це дозволить вам ефективніше розподіляти навантаження, контролювати робочий потік та уникати перевантаження колег. Крім того, ви зможете визначити вільних експертів і чітко призначити завдання на проект.

Незважаючи на популярність та багато позитивних аспектів *Asana,* користувачі також можуть відзначати деякі недоліки:

– Специфічний Дизайн Інтерфейсу: Деякі користувачі відзначають, що дизайн інтерфейсу *Asana* може бути специфічним та вимагати певного часу для зручності.

– Відсутність Розширених Функцій У Вільній Версії: Деякі розширені функції доступні лише у платних версіях *Asana*, що може бути обмеженням для користувачів, які використовують безкоштовну версію.

– Спрощені Можливості Звітності: У порівнянні з іншими інструментами управління проектами, *Asana* може мати обмежені можливості звітності та аналітики.

– Неінтегровані Інструменти Комунікації: Інструменти комунікації в *Asana* можуть бути менш розвиненими порівняно з іншими аналогами, що може впливати на продуктивність команди.

– Обмежені Функції Версії Для Офлайн Роботи: Деякі функції можуть бути обмеженими або недоступними для роботи без підключення до Інтернету, що може бути невигідним для користувачів, які часто працюють в офлайн – режимі.

– Можливість Перенасичення Інформацією: Для новачків може виникати проблема перенасичення інформацією та складнощів у розумінні всіх можливостей *Asana*.

1.3. Аналіз вимог на основі готових рішень.

– Функціональність: Здатність створювати та видаляти завдання, призначати їх відповідальним, встановлювати терміни виконання. Можливість розподілу завдань за категоріями або етапами проекту. Функції для визначення пріоритетів та залежностей між завданнями. Можливість вносити зміни та оновлювати статус завдань.

– Керування Користувачами: Система аутентифікації та авторизації користувачів з різними ролями (адміністратор, користувач). Можливість створювати та керувати користувачами та їхніми правами доступу.

– Звітність та Аналітика: Можливість генерації звітів та статистики про виконання завдань та прогрес проекту. Аналітичні інструменти для визначення ефективності та завантаження команди.

– Інтерфейс та Взаємодія: Інтуїтивний та зручний інтерфейс для користувачів. Можливість взаємодії та обміну інформацією між учасниками проекту.

– Безпека: Заходи безпеки для захисту конфіденційної інформації та даних користувачів.

– Масштабованість та Продуктивність: Здатність працювати ефективно при великій кількості завдань та користувачів.

– Оптимізована продуктивність для забезпечення швидкого реагування на дії користувачів.

Висновки

Створення програмного застосунку для керування задачами в IT –проекті виявляється завданням, яке вимагає комплексного підходу та докладного проектування. Зазначені вище вимоги формують фундамент для розробки

ефективного та зручного інструменту, що сприяє ефективному управлінню завданнями та оптимізації робочих процесів в команді.

Застосунок повинен відповідати потребам команди, надаючи засоби для ефективного створення, розподілу та відстеження завдань. Функціональність, яка включає в себе розподіл завдань за категоріями, встановлення пріоритетів та залежностей, а також звітність та аналітику, гарантує повноцінне використання інструменту для керування проектами. Засоби керування користувачами та їхніми правами дозволяють ефективно організувати робочий процес та гарантують конфіденційність інформації. Інтерфейс повинен бути інтуїтивно зрозумілим та зручним для користувачів, щоб забезпечити зручність взаємодії та навігації. Загальною метою є створення інструменту, який сприяє ефективному та організованому виконанню завдань, забезпечуючи команді необхідні інструменти для досягнення успіху в управлінні IT – проектами.

РОЗДІЛ 2
ВИМОГИ ДО ЗАСТОСУНКУ КЕРУВАННЯ ЗАДАЧАМИ ІТ – ПРОЕКТУ

2.1. Функціональні та нефункціональні вимоги

Застосунок для керування задачами в ІТ – проектах спрямований на надання зручного та інноваційного інструменту для оптимізації взаємодії між учасниками проекту. Зокрема, цей інструмент сприяє створенню та відстеженню завдань, плануванню ресурсів, комунікації у команді та доступу до важливої інформації. Мета – покращення ефективності та результативності управління проектами в області ІТ.

Функціональні вимоги для мого дипломного проекту у сфері керування задачами в ІТ – проектах описують широкий набір можливостей, спрямованих на ефективне управління завданнями та проектами. Серед них – можливість створення та редагування завдань, відстеження прогресу виконання, ресурсне планування, спілкування між командами та доступ до даних проекту. Ці функціональності сприятимуть поліпшенню продуктивності та ефективності управління проектами в області ІТ.

Нефункціональні вимоги визначають стандарти безпеки, ефективності та інших аспектів. Ці вимоги включають аспекти захисту конфіденційності даних, оперативної реакції системи на користувацькі запити, інтуїтивного та зрозумілого інтерфейсу для зручної взаємодії з платформою. Також враховуються можливості використання додатку на різних пристроях та забезпечення доступності для всіх користувачів, незалежно від їхніх фізичних особливостей.

Функціональні вимоги:

– Створення та Редагування Завдань: Користувач повинен мати можливість додавати нові завдання, вказуючи опис, дату виконання, відповідальних учасників та інші необхідні атрибути. Система повинна дозволяти редагувати інформацію щодо завдань.

– Відстеження Прогресу та Статусів: Забезпечити функціонал для відстеження прогресу виконання завдань та визначення їхнього поточного статусу (в роботі, завершено, відкладено і т. д.). Надати можливість визначення пріоритетів для кожного завдання.

– Керування Ресурсами та Завданнями: Реалізувати функції для призначення ресурсів (людських, матеріальних) для виконання конкретних завдань. Забезпечити можливість планування та розподілу завдань між учасниками проекту.

– Система Комунікацій: Розробити засіб комунікації в межах застосунку для обговорення завдань, обміну ідеями та співпраці між учасниками проекту. Забезпечити можливість надсилання сповіщень та повідомлень.

Нефункціональні вимоги:

– Швидкодія: Система повинна надавати швидкий та безперебійний доступ до інформації та функціоналу, щоб користувачі могли ефективно взаємодіяти з застосунком.

– Безпека та Захист Даних: Забезпечити високий рівень безпеки для захисту конфіденційності та цілісності проектної інформації. Реалізувати систему аутентифікації та авторизації користувачів.

– Масштабованість: Застосунок повинен бути масштабованим, здатним обробляти збільшення обсягу даних та користувачів з ростом проектів.

– Сумісність: Забезпечити сумісність з різними веб-браузерами та операційними системами для максимальної доступності.

– Документація та Підтримка: Забезпечити докладну документацію з використання та налаштування застосунку. Надати канали підтримки для користувачів та вирішення можливих проблем.

Ці функціональні та нефункціональні вимоги є визначальними для розробки веб-додатку для ефективного керування завданнями в області IT – проектів. Вони чітко формулюють основні функції, необхідні для ефективного керування завданнями, такі як створення, відстеження та планування ресурсів для проектів. Крім того, вони встановлюють високі стандарти щодо безпеки, ефективності та зручності використання, забезпечуючи надійні умови для ефективного управління завданнями в сфері IT – проектів.

2.2. Вимоги до апаратної частини та інтерфейсу

Апаратні вимоги:

– Процесор: Мінімальний процесор: *Intel Celeron* або еквівалент. Рекомендований процесор: *Intel Core i3* 6100 або вище для оптимальної продуктивності.

– Оперативна пам'ять (*RAM*): Мінімальна кількість *RAM*: 2 ГБ. Рекомендована кількість *RAM*: 4 ГБ або більше для обробки великої кількості даних та одночасної роботи користувачів.

– Простір на Жорсткому Диску: Мінімальний обсяг вільного простору: 5 ГБ для встановлення та роботи застосунку. Рекомендований обсяг вільного простору: 20 ГБ для забезпечення достатнього простору для даних проектів та резервних копій.

– Графічна Карта: Мінімальні вимоги графічної карти для виведення інтерфейсу користувача. Рекомендується графічна карта із підтримкою високого розширення для зручності роботи з графікою та зображеннями.

– Мережевий Адаптер: Мережевий адаптер для забезпечення підключення до Інтернету та обміну даними між користувачами та сервером.

– Операційна Система: Сумісність із операційними системами: Windows 10, macOS, Linux.

– Інші Пристрої: Миша та клавіатура для зручного введення та навігації.

Вимоги до інтерфейсу:

– Інтуїтивний та Простий Дизайн: Максимально спростити інтерфейс для забезпечення легкості використання навіть для новачків. Забезпечити логічну та структуровану організацію елементів інтерфейсу.

– Легкість Навігації: Розробити чітку та доступну систему навігації, яка дозволяє користувачам швидко переміщатися між різними частинами застосунку. Додати інтуїтивні елементи керування та кнопки для швидкого доступу до основних функцій.

– Зручність для Великих Обсягів Даних: Забезпечити зручний перегляд та редагування великої кількості завдань та даних проекту без втрати продуктивності. Включити ефективні фільтри та сортування для полегшення роботи з обсягами інформації.

– Гнучкість та Персоналізація: Дозволити користувачам налаштовувати інтерфейс згідно з їхніми потребами. Надати можливість персоналізації кольорів, мови, розмірів шрифтів та інших параметрів.

– Графічні Елементи: Використовувати чіткі та естетично приємні графічні елементи. Забезпечити підтримку високої якості зображень для зручного відображення графічного контенту.

– Мобільна Доступність: Якщо можливо, розробити адаптивний дизайн для забезпечення ефективної роботи застосунку на різних пристроях, включаючи мобільні та планшетні.

– Сповіщення та Система Підказок: Додати систему сповіщень та підказок для повідомлення користувачів про важливі події та можливості застосунку.

– Тестування Інтерфейсу: Провести тестування інтерфейсу з реальними користувачами для отримання фідбеку та вдосконалення зручності використання.

2.3. Вимоги до безпеки

– Аутентифікація та Авторизація: Реалізувати надійну систему аутентифікації, щоб переконатися, що тільки вповноважені користувачі мають доступ до застосунку та його функціоналу. Встановити чіткі правила авторизації, обмежуючи доступ до конфіденційної інформації відповідно до ролей користувачів.

– Шифрування Даних: Забезпечити шифрування для збереження конфіденційної інформації, особливо при зберіганні та передачі даних через мережу. Використовувати сучасні алгоритми шифрування для захисту від несанкціонованого доступу.

– Захист від Атак: Вбудовувати заходи захисту від різних типів атак, таких як *SQL*-ін'єкції, атаки переповнення буфера та інші. Здійснювати регулярні аудити та сканування безпеки для виявлення потенційних вразливостей.

– Резервне Копіювання та Відновлення: Забезпечити систему регулярного резервного копіювання даних для запобігання втраті інформації у випадку аварій, атак або інших подій. Здійснювати тестування процедур відновлення для забезпечення ефективності в разі потреби.

– Моніторинг та Виявлення Інцидентів: Розробити систему моніторингу захисту, яка виявлятиме ненормальну або підозрілу активність. Визначити процедури реагування на інциденти та швидкого реагування на потенційні загрози.

– Оновлення та Патчі: Забезпечити регулярні оновлення та патчі для застосунку для виправлення виявлених вразливостей та підтримки актуальності захисту.

– Фізична Безпека: Здійснювати заходи забезпечення фізичної безпеки серверів та зберігання даних.

Ці вимоги допоможуть створити стійкий до загроз середовище та забезпечити високий рівень захисту для важливої інформації.

Висновки

В ході розробки програмного застосунку для керування задачами IT – проекту було визначено широкий спектр функціональних та нефункціональних вимог для забезпечення високої ефективності, зручності використання та безпеки. Функціональні вимоги визначили основні можливості застосунку, що дозволяють користувачам ефективно керувати завданнями та проектами. Зокрема, створення та редагування завдань, відстеження прогресу, керування ресурсами та інші функції забезпечують повноту та зручність використання.

Нефункціональні вимоги надають основу для надійності та ефективності застосунку. Спрямованість на швидкодію, безпеку даних, масштабованість та інші аспекти забезпечують стабільність та високий рівень взаємодії. Гнучкість та можливість персоналізації інтерфейсу дозволяють користувачам адаптувати середовище роботи до своїх потреб.

У розділі апаратних вимог визначено необхідність використання сучасних обчислювальних ресурсів та забезпечення мережевого підключення. Це важливо для забезпечення ефективності та зручності використання застосунку на різних пристроях та платформах.

Вимоги до інтерфейсу визначають ключові аспекти зручності та ергономіки, забезпечуючи легкість навігації, гнучкість та привабливий дизайн. Це створює комфортне робоче середовище для користувачів та сприяє підвищенню продуктивності.

Вимоги до безпеки є важливим компонентом усього процесу розробки. Вони гарантують захист конфіденційної інформації, надійність та стійкість системи до потенційних загроз.

Загальною метою цих вимог є створення високоефективного, зручного та безпечного інструменту для керування завданнями IT – проектів, який відповідає найвищим стандартам та вимогам сучасної індустрії.

РОЗДІЛ 3
СТРУКТУРА ПРОГРАМНОГО ПРОДУКТУ

3.1. Структура програмного застосунку

Для реалізації даного програмного продукту було обрано, *MVT (Model-View-Template)* – це архітектурний шаблон, який використовується у фреймворку *Django* для розробки веб-додатків. Він взятий із шаблону *Model-View-Controller* (*MVC*), проте відрізняється своєю реалізацією та внутрішнім устроєм. Основні компоненти *MVT* включають:

Модель (*Model*): Модель представляє базу даних та бізнес – логіку додатку. Вона визначає структуру даних, з якими працює програма. В *Django* модель – це клас *Python*, який визначає атрибути та методи для роботи з базою даних.

Вид (*View*): Вид відповідає за обробку та відображення інформації. В *Django* відображення – це функції або класи *Python*, які приймають запит, обробляють його та повертають відповідь. Вони визначають, як дані відображаються та які дії виконуються при взаємодії з користувачем.

Шаблон (*Template*): Шаблон відповідає за представлення даних. В *Django* це файл, що містить *HTML*-код та вбудовані теги, які дозволяють вставляти дані в сторінку. Шаблон використовується для створення зовнішнього вигляду веб-сторінок.

Основна ідея *MVT* полягає в тому, що модель взаємодіє з базою даних та зберігає дані, вид обробляє запити та відображає інформацію, а шаблон відповідає за структуру та вигляд виведених даних. Цей підхід дозволяє розробникам легко розділити логіку додатку та його представлення, спрощуючи процес розробки та підтримки.

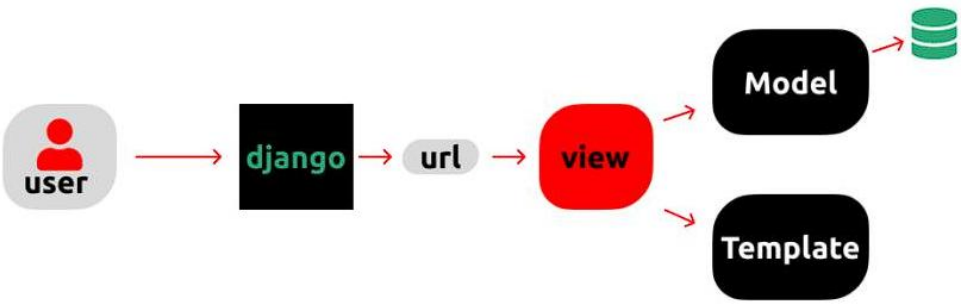

Рис. 3.1. *MVT (Model-View-Template)* – архітектурний шаблон

Інтерфейс користувача додатку орієнтований на максимальну зручність та інтуїтивність взаємодії. Головна сторінка включає перегляд завдань у вигляді списку з ключовими деталями, сортування та фільтрацію за різними параметрами, а також відображення загального прогресу та статусів завдань. Форма для створення/редагування завдань дозволяє вводити необхідні дані та встановлювати зв'язки між ними. Детальна інформація про кожне завдання доступна на окремій сторінці разом із історією змін, призначеннями та коментарями. Зручне меню та навігаційна система полегшують доступ до різних розділів додатку. Форми авторизації та реєстрації забезпечують безпеку облікових даних. Модуль аналітики та звітності надає візуалізацію ключових метрик для аналізу продуктивності. Інтерактивні функції включають можливість перетягування завдань та інтерактивні кнопки для швидкої дії. Ця концепція інтерфейсу створена для полегшення взаємодії та надання повного контролю над управлінням завданнями в IT – проектах.

Серверна частина системи спроектована з урахуванням високої продуктивності, надійності та масштабованості. Вона базується на архітектурі клієнт-сервер, де сервер відповідає за обробку та зберігання даних, а також надання їхнього доступу клієнтам.

Модуль Обробки Завдань: Відповідає за логіку збереження, редагування та видалення завдань. Його функції також включають обробку запитів, які надходять від клієнта, щоб ефективно управляти завданнями в базі даних.

Модуль Авторизації та Аутентифікації: Гарантує безпеку системи шляхом перевірки та ідентифікації користувачів. Він відповідає за віртуальний вхід та перевірку прав доступу, забезпечуючи, що тільки авторизовані користувачі мають доступ до системи.

Модуль Керування Користувачами: Займається обробкою реєстрації нових користувачів, а також управлінням їхніми правами та привілеями. Цей модуль визначає процес реєстрації, перевірку ідентичності та налаштування прав доступу для кожного користувача.

Модуль Звітності та Аналітики: Відповідає за створення звітів та аналіз даних для поліпшення продуктивності. Його завдання включає генерацію звітів, які надають інформацію про виконання завдань.

База Даних: Таблиця Завдань: Зберігає інформацію про кожне завдання, включаючи його статус, призначення та інші атрибути. Ця таблиця дозволяє ефективно управляти та відстежувати прогрес виконання завдань в системі.

Таблиця Користувачів: Вміщує дані про користувачів, такі як їхні ідентифікаційні дані та рівні доступу. Ця таблиця необхідна для ефективного управління користувачами, обліку їхніх даних та забезпечення вірного використання системи.

Таблиця Сесій: Використовується для збереження унікальних ідентифікаторів сесій та забезпечення безпеки процесу авторизації. Ця таблиця гарантує, що лише авторизовані користувачі мають доступ до системи та її функціоналу.

Розберемо функціонал нашого застосунку на (рис. 2.2.).

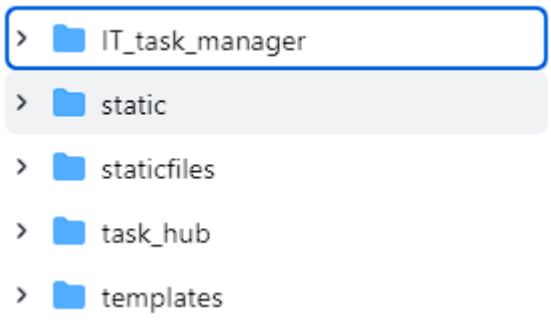

Рис. 3.2. Як можна побачити, тут розміщені усі компоненти з яких і складається функціонал нашого застосунку.

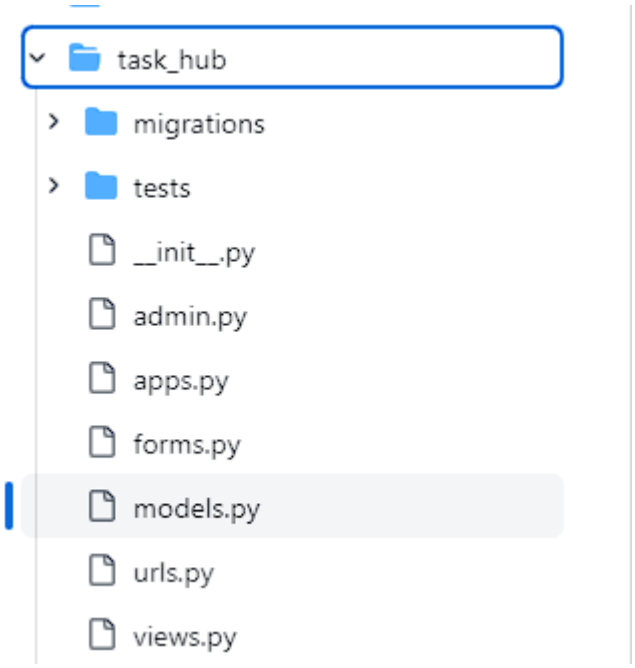

Рис. 3.3. У папці task_hub, Даний код містить опис класів для моделей даних у фреймворку *Django*.

Розглянемо компонент *models.py* в якому кожен клас відповідає за конкретний аспект системи.

Position: Клас *Position* визначає модель для представлення посад у системі. Включає поле *name*, що визначає унікальне ім'я посади. Має метод *str*, який повертає рядок, що представляє посаду.

Worker: Клас *Worker*, що розширює стандартну модель користувача *Django* (*AbstractUser*). Включає поле *position*, яке є зовнішнім ключем до моделі *Position*, вказуючи на посаду користувача. Мета-клас містить налаштування для

адміністративної панелі, а метод *str* повертає рядок, представляючи користувача.

TaskType: Клас *TaskType* визначає модель для типів завдань. Включає поле *name* для унікального імені типу завдання. Метод *str* повертає рядок, представляючи тип завдання.

Task: Клас *Task* визначає модель для представлення завдань у системі керування завданнями. Включає різні поля, такі як ім'я, опис, термін виконання та інші. Зовнішні зв'язки з моделями *TaskType* та *Worker* вказують на тип завдання та користувачів, яким призначено завдання відповідно. Мета-клас має налаштування сортування, а метод *str* повертає рядок з іменем завдання та його пріоритетом.

Приклад того як компоненти взаємодіють (див. лістинг 3.1.)

Лістинг 3.1.

Код рендерингу компонента *models.py*

```
from django.contrib.auth.models import AbstractUser
from django.db import models
class Position(models.Model):
    name = models.CharField(max_length=255, unique=True)
    def __str__(self):
        return self.name
class Worker(AbstractUser):
    position = models.ForeignKey(
        Position,
        on_delete=models.CASCADE,
        related_name="workers"
    )
    class Meta:
        verbose_name = "worker"
        verbose_name_plural = "workers"
        ordering = ["position", ]
```

```
    def __str__(self):
        return f"{self.position} ({self.first_name} {self.last_name})"
class TaskType(models.Model):
    name = models.CharField(max_length=255, unique=True)
    def __str__(self):
        return self.name
class Task(models.Model):
    class TaskPriority:
        low = "4"
        medium = "3"
        high = "2"
        urgent = "1"
    PRIORITY_CHOICES = (
        (TaskPriority.low, "Low"),
        (TaskPriority.medium, "Medium"),
        (TaskPriority.high, "High"),
        (TaskPriority.urgent, "Urgent"),
    )
    name = models.CharField(max_length=255)
    description = models.TextField()
    deadline = models.DateField()
    is_completed = models.BooleanField(default=False)
    priority = models.CharField(
        max_length=6,
        choices=PRIORITY_CHOICES,
        default=TaskPriority.medium,
    )
    task_type = models.ForeignKey(
        TaskType,
```

```
        on_delete=models.CASCADE,
        related_name="tasks"
    )
    assignees = models.ManyToManyField(Worker, related_name="tasks")
    class Meta:
        ordering = ["is_completed", "priority", "deadline"]
    def __str__(self):
        return f"{self.name} - {self.get_priority_display()}"
```

Наступним розглянемо компонент *views.py*, код якого визначає класи та функціонал для кількох сторінок веб-додатку, який базується на *Django*.

Клас *TaskListView* визначає перегляд списку завдань у системі. Основні аспекти цього класу:

– Використовує *LoginRequiredMixin* для перевірки, чи користувач увійшов у систему перед доступом до цього перегляду.

– Використовує *generic.ListView* для відображення списку завдань.

– Встановлює модель для цього перегляду, якою є *Task*.

– Визначає параметр *paginate_by*, що вказує кількість елементів на одній сторінці при пагінації списку.

Метод *get_context_data* викликається для отримання додаткових даних для передачі в шаблон. У цьому випадку він додає до контексту форму пошуку *TaskSearchForm* зі значенням, яке користувач ввів у поле пошуку.

Метод *get_queryset* визначає запит до бази даних для отримання списку завдань. Він використовує *TaskSearchForm* для фільтрації результатів за назвою завдання, враховуючи значення, введене користувачем у форму пошуку.

Клас *TaskDetailView* визначає перегляд деталей конкретного завдання. Основні особливості:

– Також використовує *LoginRequiredMixin* для перевірки аутентифікації користувача.

– Використовує *generic.DetailView* для відображення деталей конкретного завдання.

– Встановлює модель для цього перегляду, якою є *Task*.

– Метод *get_context_data* додає до контексту змінну *num_assignee*, яка містить кількість працівників, призначених на це завдання.

Лістинг 3.2.

Код рендерингу компонета *views.py*

```
class TaskListView(LoginRequiredMixin, generic.ListView):
    model = Task
    paginate_by = 6
    def get_context_data(self, *, object_list=None, **kwargs):
        context = super(TaskListView, self).get_context_data(**kwargs)
        name = self.request.GET.get("name", "")
        context["search_form"] = TaskSearchForm(
            initial={"name": name}
        )
        return context
    def get_queryset(self):
        form = TaskSearchForm(self.request.GET)
        queryset = Task.objects.all().select_related("task_type")
        if form.is_valid():
            return queryset.filter(name__icontains=form.cleaned_data["name"])
        return queryset
class TaskDetailView(LoginRequiredMixin, generic.DetailView):
    model = Task
    def get_context_data(self, *, object_list=None, **kwargs):
        context = super(TaskDetailView, self).get_context_data(**kwargs)
        num_assignee = self.object.assignees.count()
        context["num_assignee"] = num_assignee
        return context
```

3.2. Діаграми розгортання, класів та логіки взаємоді додатку.

Діаграми відіграють суттєву роль у процесі розробки програмних додатків, оскільки вони виявляються ефективним засобом візуалізації та передачі великої кількості інформації, пов'язаної з архітектурою, структурою, процесами та взаємодією компонентів програмного забезпечення. Діаграми служать потужним інструментом для забезпечення ефективного спілкування між розробниками, дизайнерами, керівництвом проекту та зацікавленими сторонами.

Використання діаграм дозволяє відобразити взаємозв'язки між компонентами додатку, описати потоки даних, виявити залежності та інші суттєві аспекти проекту. Ці інструменти сприяють розумінню повного життєвого циклу додатку, починаючи від архітектурного проектування і закінчуючи реалізацією та тестуванням.

Діаграми можуть приймати різні форми та типи, такі як блок-схеми, *UML*-діаграми, потокові діаграми, сіткові діаграми та інші. Вони сприяють чіткому розумінню структури та функціональності додатку, а також виявленню потенційних проблем і недоліків у проектуванні.

Загалом, використання діаграм є важливим засобом візуалізації та забезпечення ефективної комунікації під час розробки програмних додатків. Це сприяє забезпеченню зрозумілості, узгодженості та ефективності у роботі над проектом, сприяючи успішному впровадженню та функціональності додатку. Використання діаграм допомагає зробити процес розробки додатку більш структурованим, організованим і ефективним, що в свою чергу сприяє досягненню бажаних результатів.

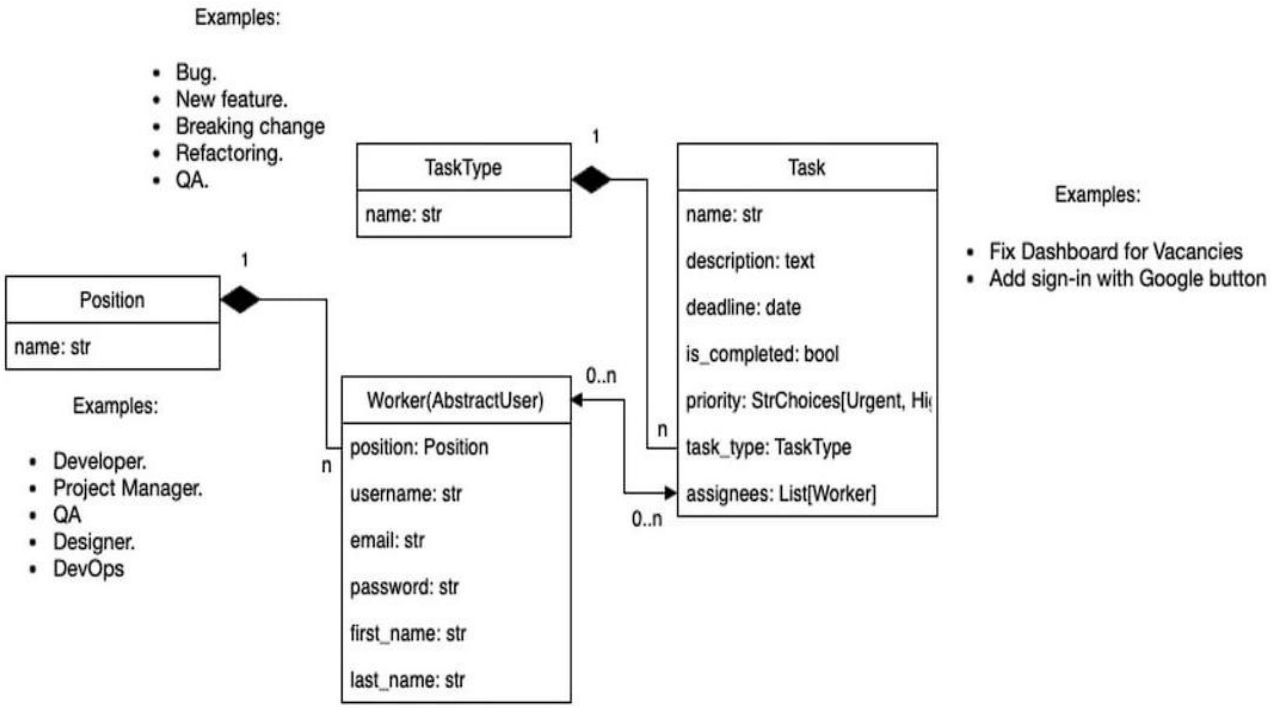

Рис. 3.4. Структура Бази Даних.

Position (Посада): Зберігає назву посади.

Worker (Працівник), що успадковує *AbstractUser*: Зберігає дані про працівника та посилання на його посаду.

TaskType (Тип завдання): Зберігає різновиди завдань.

Task (Завдання): Зберігає дані про завдання, включаючи його назву, опис, термін виконання, ознаку завершення, пріоритет, тип та виконавців.

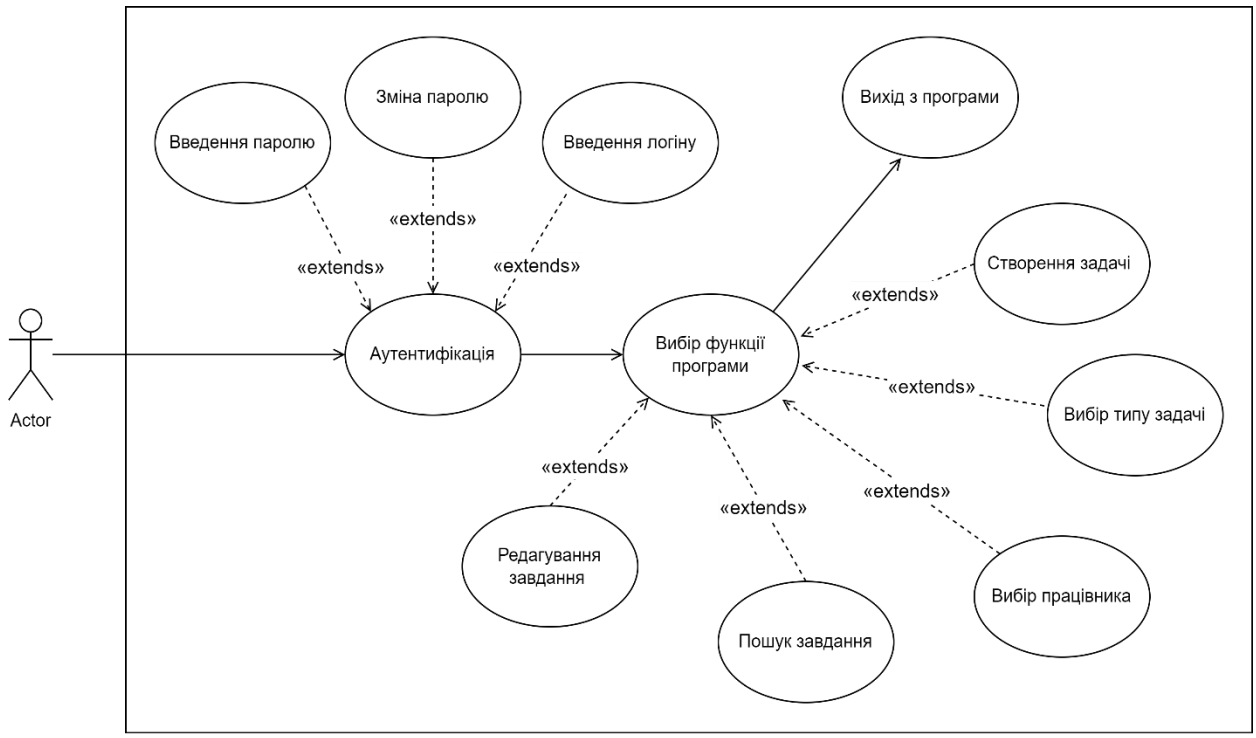

Рис. 3.5. Use – case діаграма.

На даній діаграмі видно, що та чітко зрозуміло як взаємодіє користувач безпосередньою з додатком та які є «маршрути» роботи.

Структурна схема є ключовим елементом для візуалізації організації системи чи програми. Цей вид діаграми дозволяє представити компоненти системи та їх взаємодії у вигляді блоків та стрілок. Кожен блок представляє окремий компонент, такий як модуль, клас чи бібліотека, а стрілки вказують на взаємодію чи залежність між ними.

Структурна схема спрощує розуміння того, як компоненти взаємодіють та як вони організовані всередині системи. Вона може включати в себе різні рівні деталей, такі як модулі, класи, підсистеми, і відображає їхні взаємозв'язки за допомогою стрілок.

Цей тип діаграми полегшує аналіз загальної архітектури системи та допомагає виявляти зв'язки між компонентами. Важливою є можливість використання структурної схеми для виявлення можливих проблем чи недоліків у структурі системи, що дозволяє вчасно вносити необхідні зміни та удосконалення.

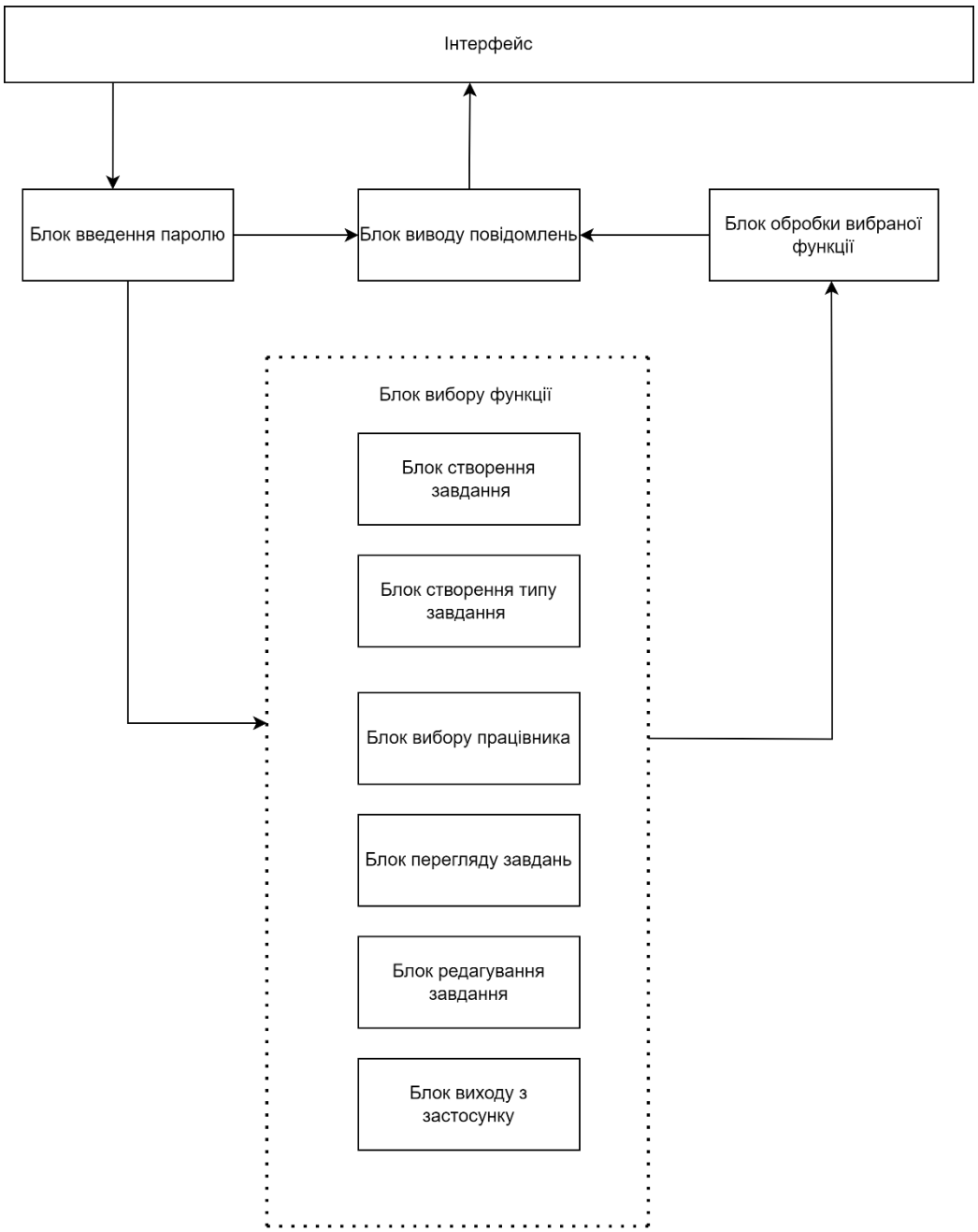

Рис. 3.6. Структурна схема.

На діаграмі присутні такі блоки:

– Блок виводу форми пароля: Цей блок відповідає за відображення форми на екрані, де користувач може вказати пароль, створити або змінити його. У формі може бути текстове поле для введення пароля та кнопка для підтвердження.

– Блок виводу повідомлень: Відповідає за відображення повідомлень на екрані користувача, таких як інформаційні повідомлення, помилки або

підтвердження дій. Повідомлення можуть мати текстовий або графічний вигляд.

– Блок вибору функції: Дозволяє користувачу обрати бажану функцію або опцію з доступних варіантів. Цей блок може містити меню, де користувач обирає необхідну дію або налаштування.

– Блок обробки вибраної функції: Відповідає за обробку вибраної користувачем функції. Включає логіку та код, які виконуються після вибору функції. Тут можуть бути операції, що залежать від вибору користувача, такі як виконання певних дій або взаємодія з іншими компонентами системи.

Схема алгоритму – це графічний інструмент для моделювання послідовності дій та прийняття рішень у виконанні алгоритму. Вона використовує геометричні фігури та зв'язки між ними для структурованого візуального подання алгоритмічного процесу.

Кожен блок на схемі алгоритму відображає окрему дію або операцію, пов'язану з обробкою даних чи змінними. Графічні символи, такі як прямокутники, ромби, кола чи паралелограми, використовуються для представлення цих блоків в залежності від призначення та характеру дії. Зв'язки між блоками вказують на послідовність виконання дій та відображають потік керування алгоритму.

Схема алгоритму надає чітке та структуроване представлення алгоритмічних процесів, спрощуючи розуміння та аналіз програмного коду. Вона допомагає виявляти помилки та оптимізувати ефективність виконання алгоритмів. Цей інструмент є важливим при розробці програмних продуктів, проєктуванні систем, а також у навчанні програмуванню та вирішенні складних завдань.

Схема алгоритму забезпечує стандартизований підхід до моделювання алгоритмів, що полегшує спілкування, аналіз та вдосконалення алгоритмів для розробників, програмістів та інших зацікавлених сторін. Її використання сприяє уніфікованому та зрозумілому представленню програмних рішень.

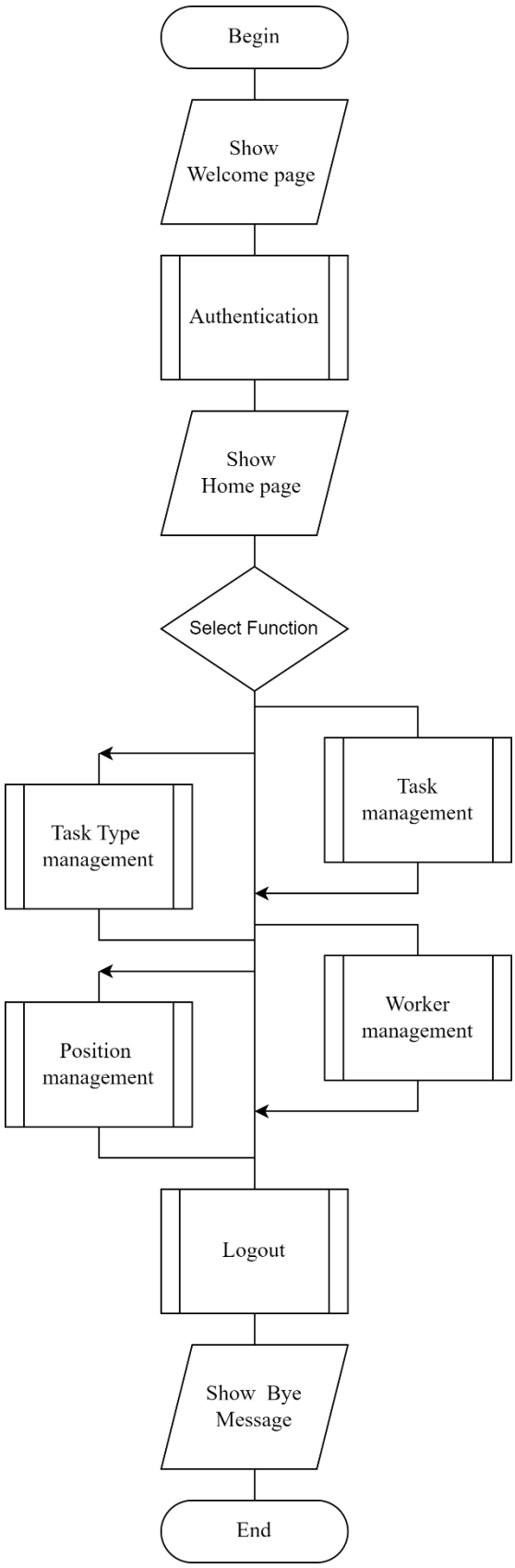

Рис. 3.7. Схема алгоритму

На схемі алгоритму присутні наступні елементи:

– початковий блок «*Begin*» та кінцевий блок «*End*». Вони вказують на початок та кінець виконання алгоритму;

– блоки виводу «*Show Welcome page*», «*Show Home page*», «*Show Bye Message*». Ці блоки відповідають за вивід на екран вказаної форми;

– блоки виклику функцій «*Task management*», «*Task Type management*», «*Worker management*», «*Position management* », «*Authentification* ». Вказані блоки відповідають за виклик функцій;

– умови «*Select Function*. Ці елемент відображають обробку умови. В залежності від обраної дії, функції або відповіді користувача, система переходить до наступних елементів.

Висновки

Розроблено та визначено діаграми модуля, що становлять невід'ємну складову уніфікованого процесу розробки програмного забезпечення. Ці діаграми використовуються на всіх етапах життєвого циклу аналізу систем та розробки додатків, надаючи засіб для візуального відображення системи таким чином, що сприяє легкості перетворення її у програмний код.

РОЗДІЛ 4
ПРОТОТИП ПРОГРАМНОГО ЗАСТОСУНКУ КЕРУВАННЯ ЗАДАЧАМИ В IT – ПРОЕКТІ

4.1. Модуль авторизації-реєстрації

Реєстрація дозволяє користувачам створювати облікові записи у додатку, введенням необхідних особистих даних та обираючи унікальні облікові дані, такі як ім'я, електронна пошта та пароль. Цей процес служить для ідентифікації користувача та забезпечує збереження його особистих налаштувань та даних.

З іншого боку, авторизація дозволяє зареєстрованим користувачам увійти до системи, використовуючи свої облікові дані. Після успішної авторизації користувач отримує доступ до особистого кабінету та функціоналу, який може бути недоступним для незареєстрованих або неавторизованих користувачів.

Модуль авторизації-реєстрації забезпечує безпеку та контроль доступу до інформації, яка зберігається в додатку. Таким чином, наш додаток не виключається з цих функцій і має в своєму арсеналі подібний функціонал. Для здійснення одного з вищезазначених варіантів використання користувач бачить у шапці сайту кнопки "*Sign in*" or "*Sign Up*".

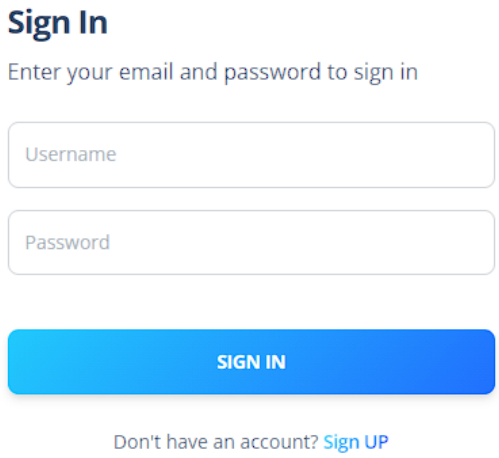

Рис. 4.1. Форма містить поля, за допомогою яких користувач може здійснити авторизацію у додатку.

Після ведення даних користувач може увійти у вже створений обліковий запис, або натиснути на "*Sign Up*", та зареєструвати новий, для цього використовується інша форма (рис. 4.2.).

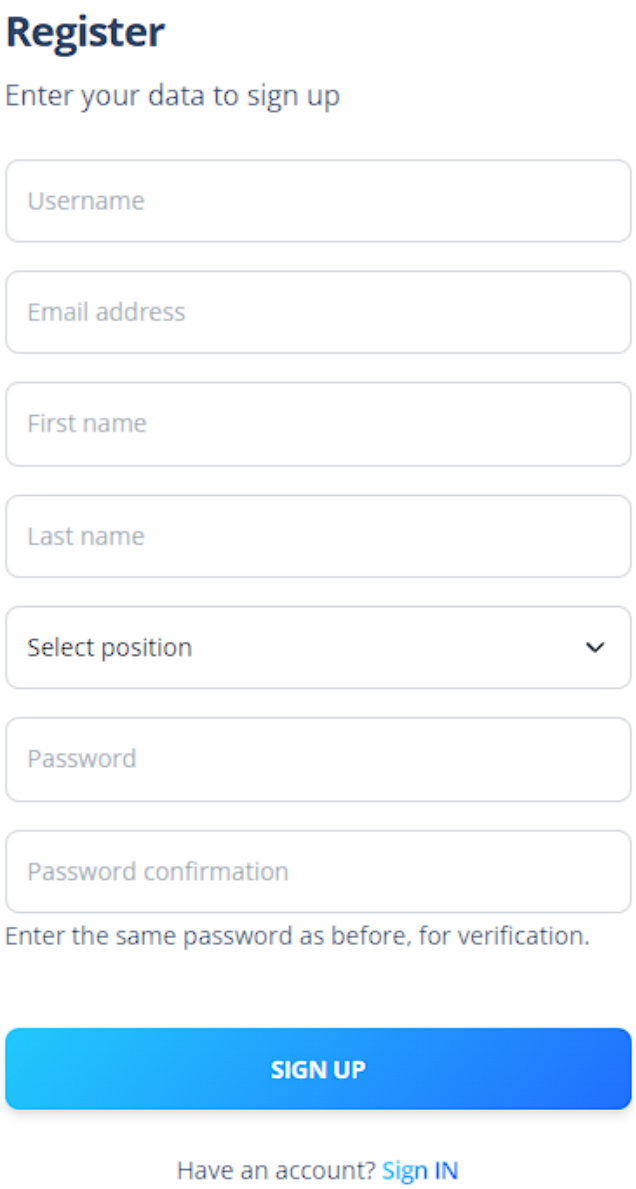

Рис. 4.2. Форма реєстрації користувача.

Форма включає всі необхідні поля для заповнення, причому кожне поле є обов'язковим для введення. Таким чином, якщо дані не будуть внесені або будуть внесені з помилками, реєстрація не відбудеться, і користувач отримає повідомлення про помилку.

Після успішної реєстрації користувач вважається авторизованим і отримує повний доступ до функціоналу інформаційної системи. Це означає, що користувач може здійснювати всі необхідні операції.

4.2. Модуль створення завдання, перегляду та виконання

Після успішної реєстрації або авторизації, меню змінюється відповідно до зображеного на (рис. 4.3.). Користувачеві стають доступні нові функції.

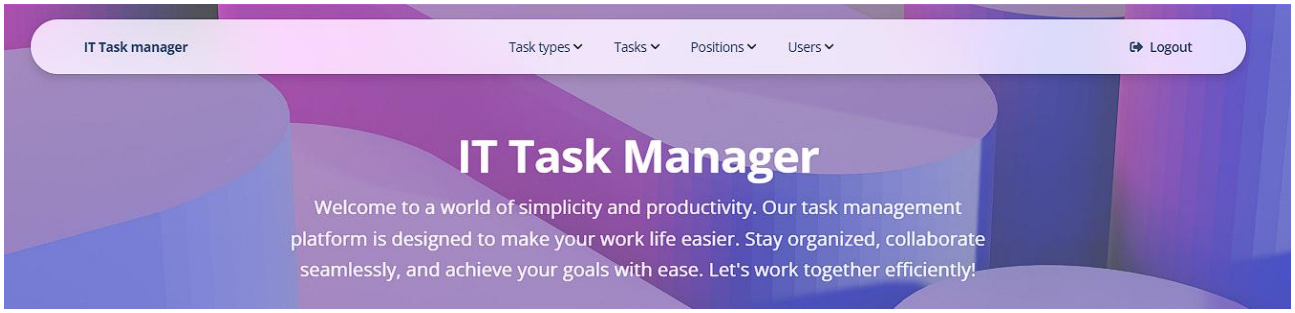

Рис. 4.3. Головне меню застосунку.

В вкладці "*Task Types*" користувач може створити задачу або вивести існуючі задачі (рис. 4.4.)

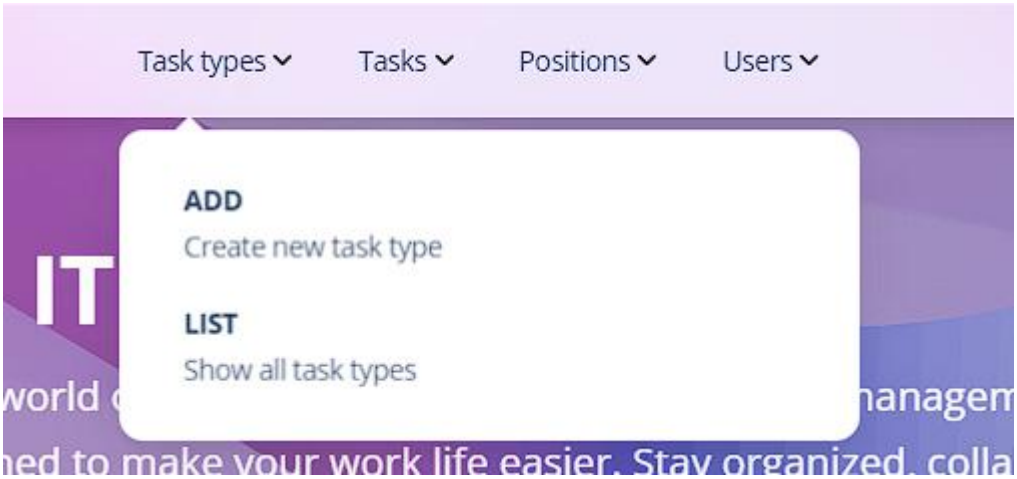

Рис. 4.4. Меню вибору дії, створити або переглянути завдання.

Натискаючи на клавішу « *ADD* » користувач отримує форму, для створення нового типу задач.

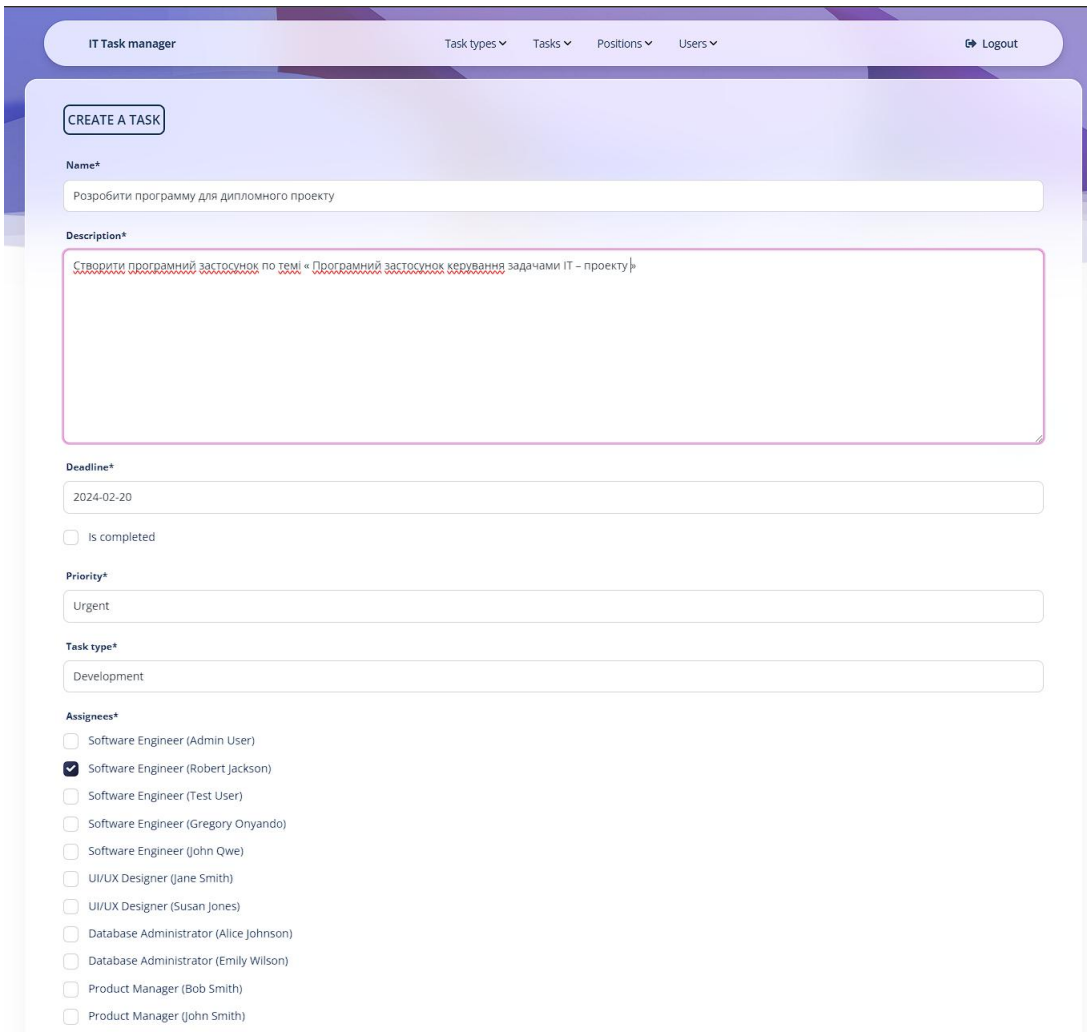

Рис. 4.5. Створення нового завдання.

На новій формі необхідно ввести дані, що потрібно зробити для виконня цієї задачі, вибираємо дату до якої повинна бути готова програма та закріпляємо її на вибір за одним з розробників

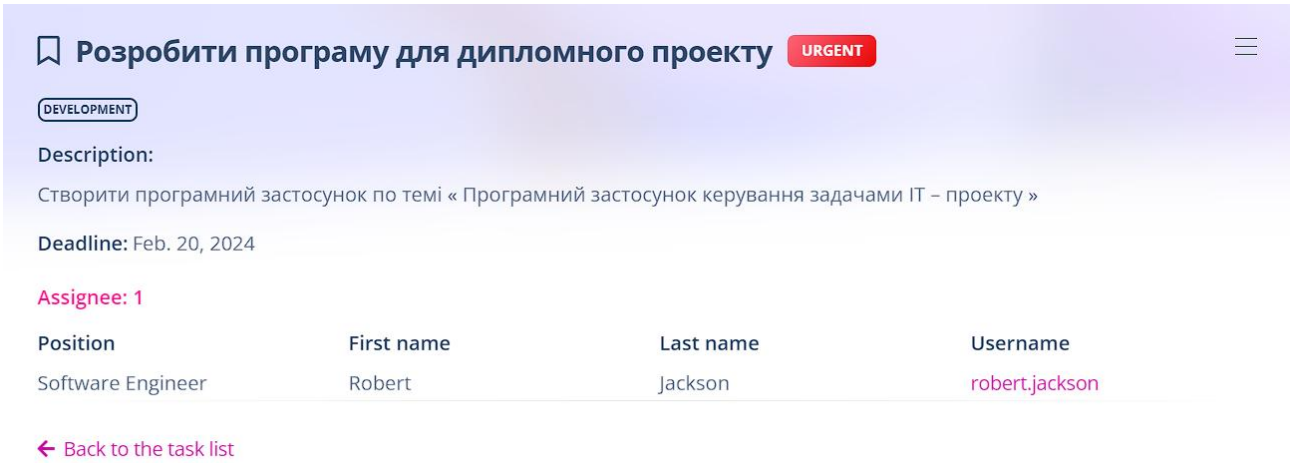

Рис. 4.6. Представлення детальної інформації про створене завдання.

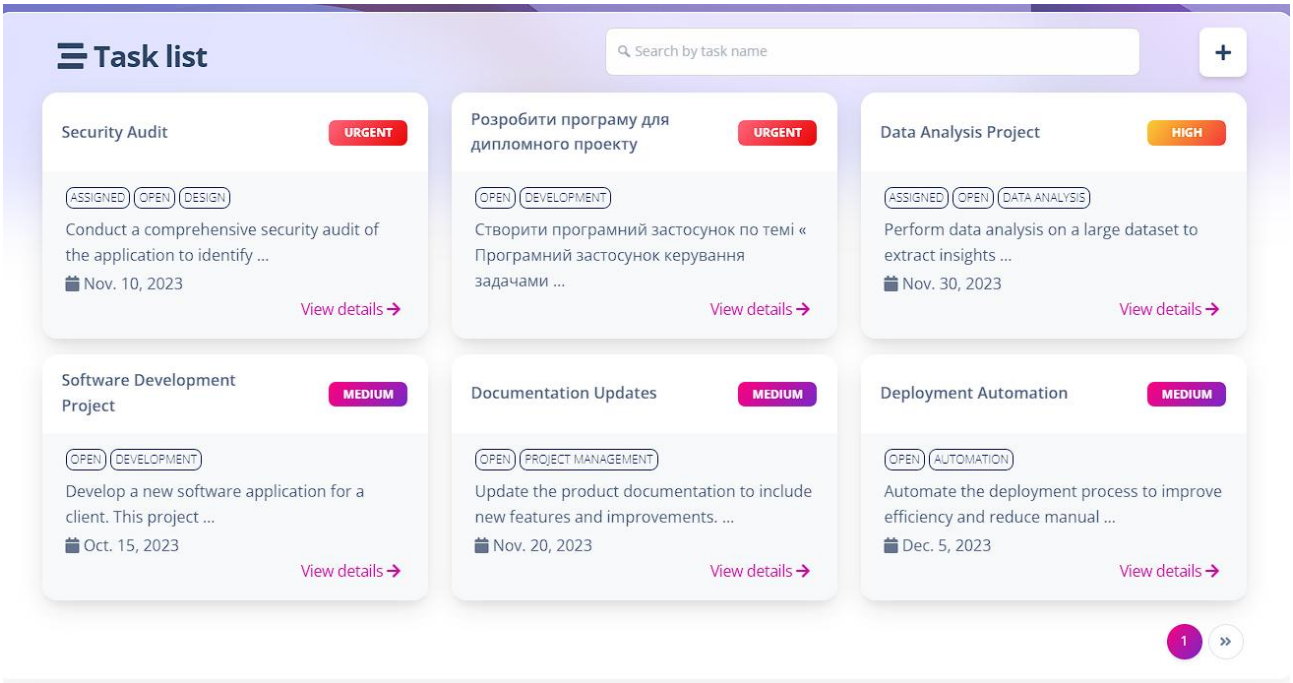

Рис. 4.7. Після створення завдання, воно з’являється у переліку « *Task List* ».

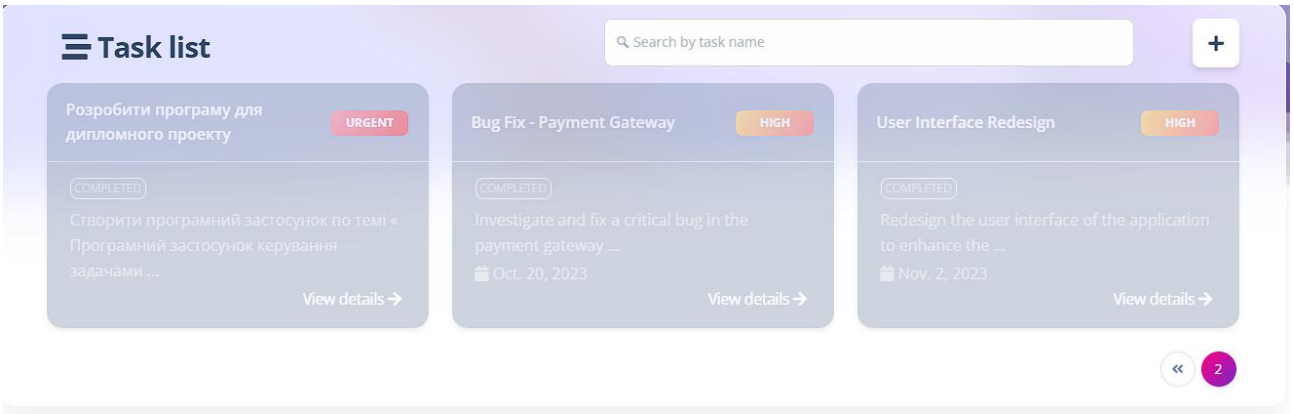

Рис. 4.8. Після виконання завдання, воно буде відображатися сірим кольором.

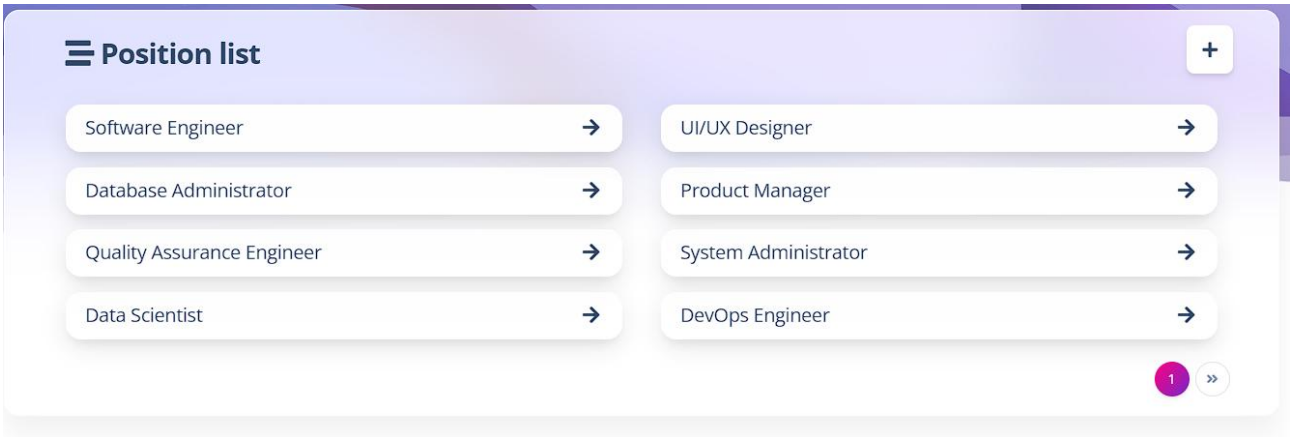

Рис. 4.9. Сервіс надає можливість взаємодії з посадами членів команди.

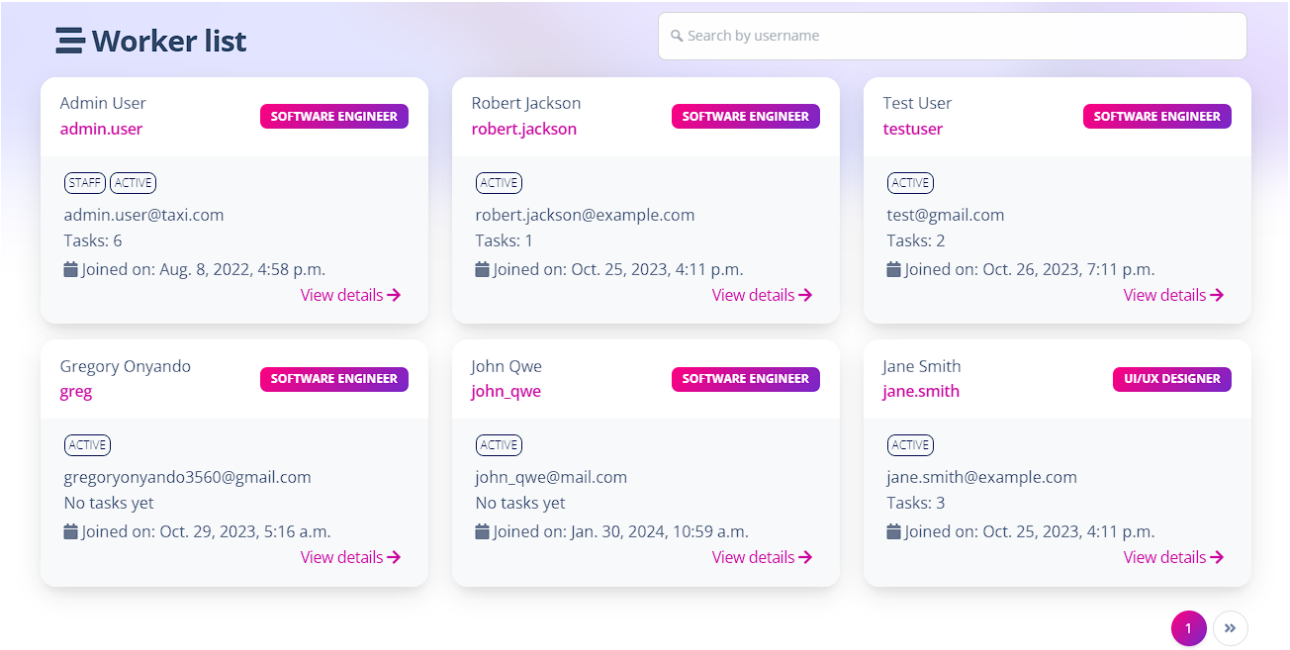

Рис. 4.10. В списку працівників виводиться базова інформація по кожному члену команди.

З вікна списку можна перейти до перегляду детальної інформації людини показано на (рис. 4.11.).

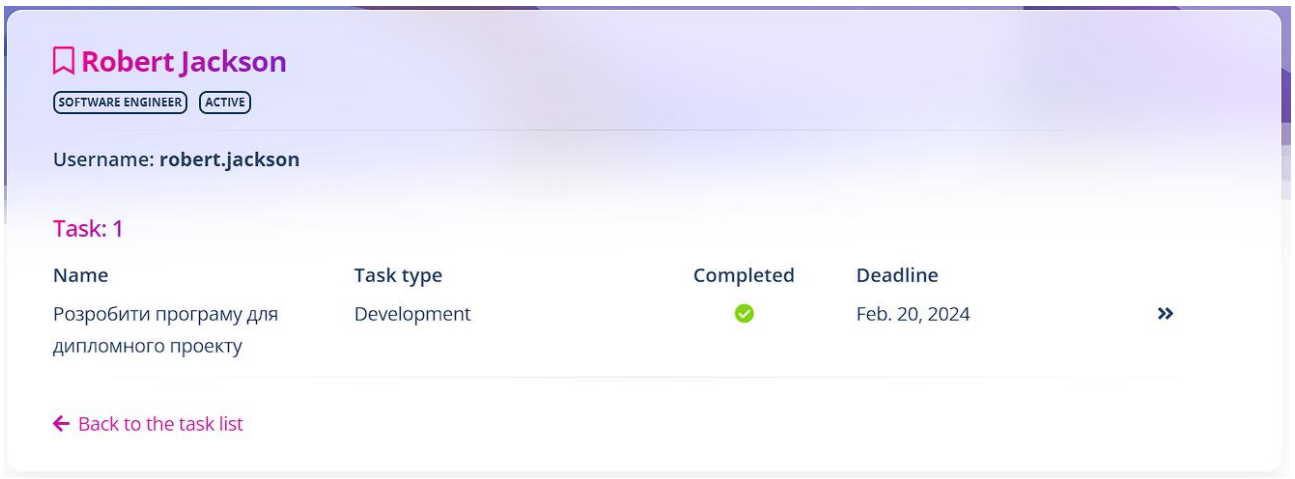

Рис. 4.11. Детальна інформація містить дані про посаду, його повне ім'я, ім'я користувача, активність та перелік завдань, що йому присвоєні.

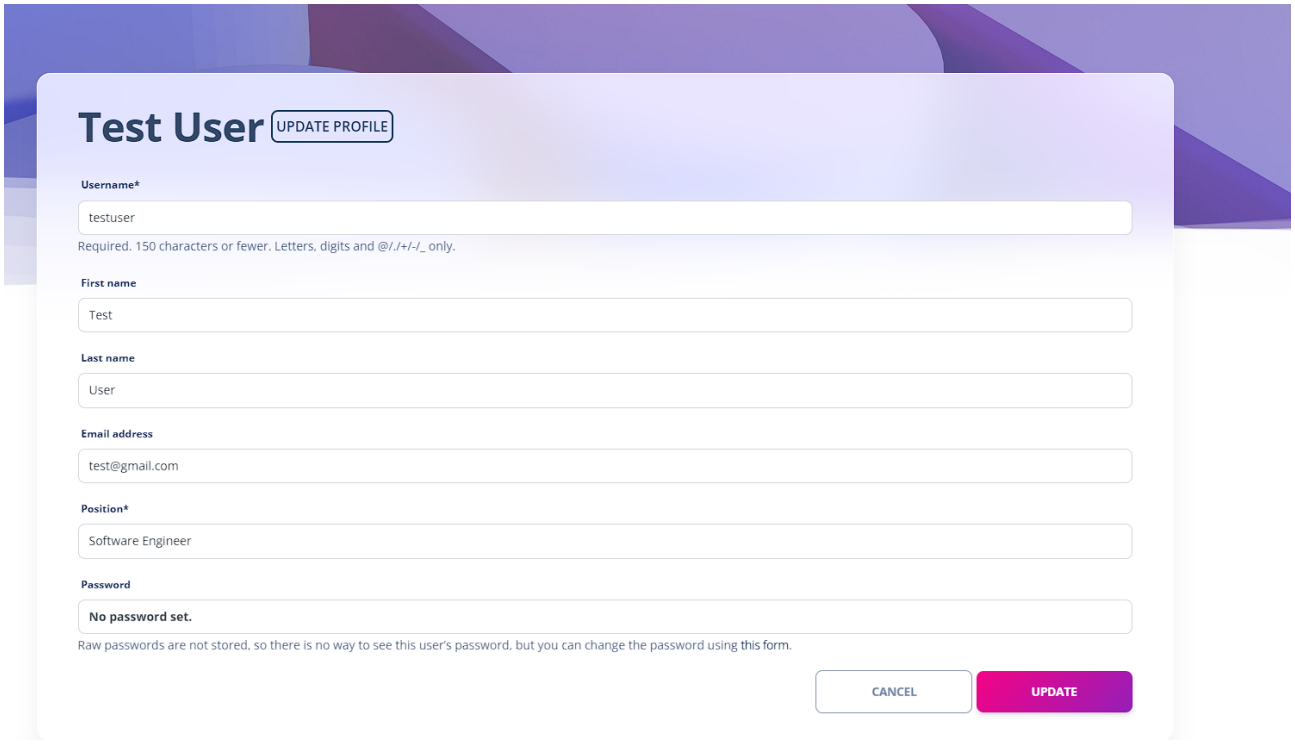

Рис. 4.12. Зареєстрований користувач може відредагувати свої дані.

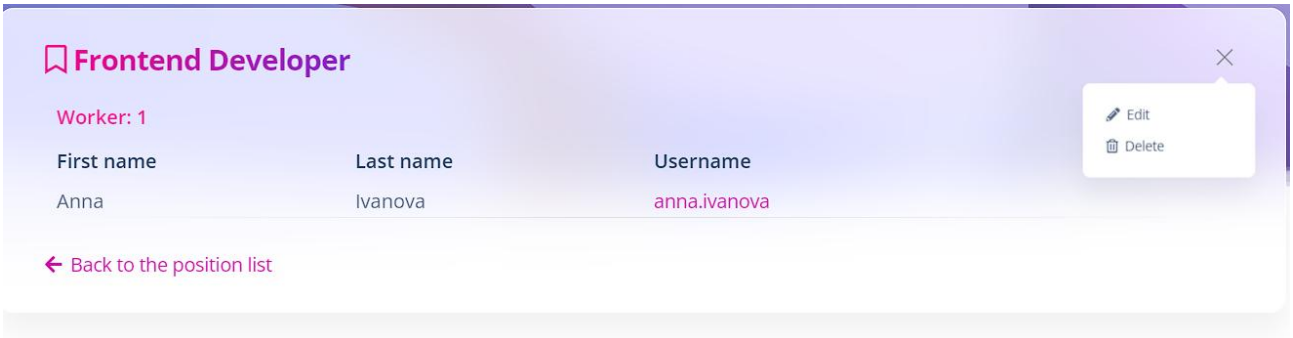

Рис. 4.13. Також в детальній інформації про робітника, можна видалити його зі списку працівників.

Висновки

Реєстрація та авторизація в додатку є важливим етапом, що відкриває користувачам повний функціонал та можливості взаємодії. Коректна форма реєстрації, обов'язкові поля та висока ступінь безпеки забезпечують надійність та захищеність особистої інформації.

Після успішної авторизації користувачеві стають доступні нові функції, зокрема можливість створення та виконання завдань. Цей функціонал розширює можливості користувача в ефективному плануванні та контролі за власними завданнями.

Забезпечуючи легкий доступ до інструментів управління завданнями, додаток стимулює користувачів бути більш продуктивними та організованими. Використання цих функцій додає значущої цінності до досвіду використання додатку та сприяє його ефективному використанню для досягнення поставлених цілей.

Таким чином, правильна реєстрація, авторизація та функції управління завданнями роблять додаток більш привабливим, забезпечуючи користувачам інструменти для успішного досягнення їхніх завдань та планів.

ВИСНОВКИ

Підсумовуючи виконану роботу, ми успішно розробили та впровадили продукт – програмний застосунок для керування завданнями в IT – проектах. Виважено враховані функціональні та нефункціональні вимоги гарантують високу ефективність, зручність використання та безпеку системи.

Розроблений застосунок відповідає потребам команди, надаючи інструменти для ефективного створення, розподілу та відстеження завдань. Його функціональність, така як розподіл завдань, встановлення пріоритетів, залежностей, звітність та аналітика, забезпечують повноту та зручність використання для керування проектами.

Нефункціональні вимоги, спрямовані на надійність та ефективність застосунку, враховують швидкодію, безпеку даних, масштабованість та інші аспекти, забезпечуючи стабільність та високий рівень взаємодії. Застосунок також відповідає апаратним вимогам, що гарантує його ефективність на різних пристроях та платформах.

Інтерфейс відзначається легкістю навігації, гнучкістю та привабливим дизайном, створюючи комфортне робоче середовище для користувачів та сприяючи підвищенню продуктивності. Вимоги до безпеки гарантують захист конфіденційної інформації та стійкість системи до потенційних загроз.

В цілому, розроблений продукт відповідає найвищим стандартам та вимогам сучасної індустрії, надаючи команді інструмент для успішного досягнення їхніх завдань та планів в управлінні IT – проектами.

Розглянуті можливості збільшення масштабів та розширення функціоналу системи. Враховано можливі потреби у збільшенні кількості користувачів та розширенні можливостей системи, що забезпечить її ефективну роботу у майбутньому.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. СМЯ НАУ П 03.01(10) – 02 – 2017. Положення про дипломні роботи (проєкти) випускників національного авіаційного університету.

2. ДСТУ 3008-95. Документація. Звіти у сфері науки і техніки; чинний від 1996-01-01. – К. : Держстандарт України, 1995. – 88 с.

3. *Python* 3.10.5 *documentation* [Електронний ресурс] / *Python Software Foundation*. – 2022. – Режим доступу до ресурсу: *https://docs.python.org/release/3.10.5/*(дата звернення: 28.05.2023). – Назва з екрана.

4. *Fowler M. UML Distilled: A Brief Guide to the Standard Object Modeling Language* / *Martin Fowler*. – Бостон: *Addison-Wesley Professional*, 2003. – 208 с.

5. *IEEE* 829-1998. *Standard for Software Test Documentation*. – чинний від 1998-12-16. – Вид. офіц. *Institute of Electrical and Electronic Engineers*, 1998. – 59 с.

6. *Jira Software Data Center* [Електронний ресурс] *Atlassian*. – 2023. – Режим доступу до ресурсу: *https://www.atlassian.com/software/jira/download/data-center* (дата звернення: 30.05.2023). – Назва з екрана.

7. Python documentation [Електронний ресурс] – Режим доступу до ресурсу: https://www.python.org/

8. Asana documentation [Електронний ресурс] – Режим доступу до ресурсу: https://asana.com/

9. Asana [Електронний ресурс] – Режим доступу до ресурсу: https://cloudfresh.com/ru/cloud-blog/chto-takoe-asana-project-management-kak-uprostyt-vash-rabochyj-protsess/

10. Asana [Електронний ресурс] – Режим доступу до ресурсу:https://euprostir.org.ua/practices/134869

11. Jira [Електронний ресурс] – Режим доступу до ресурсу: https://www.atlassian.com/ru/software/jira

12. PyCharm [Електронний ресурс] – Режим доступу до ресурсу: https://foxminded.ua/ru/pycharm-eto/

13. CSS [Електронний ресурс] – Режим доступу до ресурсу:https://developer.mozilla.org/ru/docs/Learn/Getting_started_with_the_web/CSS_basics

14. HTML [Електронний ресурс] – Режим доступу до ресурсу:https://developer.mozilla.org/ru/docs/Learn/Getting_started_with_the_web/HTML_basics

15. Draw UML [Електронний ресурс] – Режим доступу до ресурсу:https://app.diagrams.net/