Міністерство освіти і науки України

Ніжинський державний університет імені Миколи Гоголя

Навчально-науковий інститут природничо-математичних, медико-біологічних наук та інформаційних технологій

Кафедра інформаційних технологій, фізико-математичних та  
економічних наук

Освітня програма: Комп’ютерні науки

Спеціальність: 122 Комп’ютерні науки

**КВАЛІФІКАЦІЙНА РОБОТА**

на здобуття освітнього ступеня *магістр*

**Програмна реалізація системи управління  
задачами для керування IT проєктами**

студента **Мандрики Юрія Валерійовича**

**Науковий керівник:**  
Лисенко Ірина Миколаївна,

кандидат фізико-математичних наук, доцент

**Рецензенти:**

Пузікова А.В.,

кандидат фізико-математичних наук, доцент,

Фетісов В.С.,

кандидат економічних наук, доцент

**Допущено до захисту:** \_\_\_\_ \_\_\_\_\_\_ 2024р.  
Завідувач кафедри

**\_\_\_\_\_\_\_\_\_\_\_\_\_**\_\_\_\_\_\_\_проф.Казачков І.В.

Ніжин – 2024

АНОТАЦІЯ

Магістерська робота "Програмна реалізація системи управління задачами для керування IT проєктами" присвячена розробці та аналізу ефективної системи управління задачами, адаптованої до потреб сучасних IT-проєктів. Об'єктом дослідження є процеси управління задачами в сфері інформаційних технологій, а предметом роботи – програмна реалізація системи, яка оптимізує ці процеси.

Робота складається зі вступу, основної частини, поділеної на чотири розділи, висновків та списку використаних джерел. Перший розділ містить теоретичний огляд існуючих підходів і рішень у сфері систем управління задачами для IT-проєктів, зокрема методологій Waterfall, Agile, Scrum, Kanban та Extreme Programming (XP). У другому розділі проведено детальний аналіз існуючих систем та методологій, включаючи функціональні та нефункціональні аспекти.

Третій розділ зосереджений на формуванні основних вимог до системи та розробці архітектури системи управління задачами. Обґрунтовано вибір методології Kanban як оптимальної для вирішення поставлених завдань.

Четвертий розділ описує процес програмної реалізації системи та її інтеграцію з вибраними технологіями. Детально розглянуто вибір операційної системи, налаштування середовища розробки, встановлення необхідних компонентів та створення основних фронтенд- та бекенд-компонентів системи.

Отримані результати дозволили сформулювати висновки та рекомендації щодо впровадження ефективної системи управління задачами на основі Kanban, яка сприяє оптимізації робочих процесів, підвищенню продуктивності команди та успішному керуванню IT-проєктами.

Ключові слова: управління задачами, IT-проєкти, Kanban, Agile, Scrum, система управління проєктами, оптимізація робочих процесів, продуктивність команди, методології розробки ПЗ, візуалізація робочого процесу, програмна реалізація.

ANNOTATION

Master's Thesis "Software Implementation of a Task Management System for IT Project Management" is dedicated to the developing and analyzing of an efficient task management system tailored to the needs of modern IT projects. The object of study is task management processes in the field of information technology, while the subject is the software implementation of a system that optimizes these processes.

The thesis consists of an introduction, a main part divided into four chapters, conclusions, and a list of references. The first chapter provides a theoretical overview of existing approaches and solutions in task management systems for IT projects, including methodologies such as Waterfall, Agile, Scrum, Kanban, and Extreme Programming (XP). The second chapter presents a detailed analysis of existing systems and methodologies, covering both functional and non-functional aspects.

The third chapter focuses on defining the primary requirements for the system and developing its architecture. The choice of the Kanban methodology is justified as the most suitable for solving the identified tasks.

The fourth chapter describes the process of software implementation and integration with selected technologies. It thoroughly examines the choice of operating system, development environment setup, installation of necessary components, and creation of core frontend and backend components of the system.

The results allowed the formulation of conclusions and recommendations for implementing an effective Kanban-based task management system that promotes workflow optimization, team productivity improvement, and successful IT project management.

Keywords: task management, IT projects, Kanban, Agile, Scrum, project management system, workflow optimization, team productivity, software development methodologies, workflow visualization, software implementation.

ЗМІСТ

ВСТУП 7

РОЗДІЛ 1 ТЕОРЕТИЧНИЙ ОГЛЯД ІСНУЮЧИХ МЕТОДОЛОГІЙ ТА ПІДХОДІВ У СФЕРІ СИСТЕМ УПРАВЛІННЯ ЗАДАЧАМИ ДЛЯ IT-ПРОЄКТІВ 11

1.1. Вступ до методологій управління проєктами 11

1.2. Основний огляд підходу "Водоспад" (Waterfall) 11

1.3. Основний огляд підходу Agile 13

1.3.1. Scrum як фреймворк Agile 15

1.3.2. Kanban як метод управління задачами 17

1.3.3. Extreme Programming (XP) 20

1.3.4. Порівняльний аналіз гнучких методологій 21

1.4. Висновки по розділу 1 23

РОЗДІЛ 2 ОГЛЯД ДЕЯКИХ СУЧАСНИХ ПРОГРАМНИХ РЕАЛІЗАЦІЙ У СФЕРІ СИСТЕМ УПРАВЛІННЯ ЗАДАЧАМИ ДЛЯ IT-ПРОЕКТІВ 26

2.1. Вступ до систем управління задачами 26

2.2. Огляд основних програмних рішень 26

2.2.1. JIRA від Atlassian 26

2.2.2. Trello 28

2.2.3. Asana 30

2.2.4. Microsoft Project 32

2.3. Порівняння програмних рішень 35

2.3.1. Підтримка методологій програмними рішеннями 35

2.4. Висновки до розділу 2 38

РОЗДІЛ 3 ФОРМУВАННЯ ВИМОГ ТА АРХІТЕКТУРИ ДО СИСТЕМИ УПРАВЛІННЯ ЗАДАЧАМИ 40

3.1. Формування основних вимог до системи та обґрунтування вибору підходу Kanban 40

3.2. Розробка концептуальної моделі системи 44

3.3. Вибір технологічного стеку 47

3.4. Висновки до розділу 3 48

РОЗДІЛ 4 ПРОЦЕС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ СИСТЕМИ ТА ЇЇ ІНТЕГРАЦІЯ З ОБРАНИМИ ТЕХНОЛОГІЯМИ 50

4.1. Вибір операційної системи та налаштування середовища розробки 50

4.2. Встановлення Node.js та Yarn 51

4.3. Встановлення Strapi 52

4.4. Налаштування Strapi та створення колекцій 54

4.4.1. Перший вхід у Strapi 54

4.4.2. Налаштування колекцій 55

4.4.2.1. Колекція Project 56

4.4.2.2. Колекція Board 57

4.4.2.3. Колекція Column 59

4.4.2.4. Колекція Task 59

4.4.2.5. Колекція Comment 59

4.4.2.6. Колекція User 60

4.4.3. Налаштування ролей 60

4.4.4. Генерація API-токену для тестування бекенду 62

4.5. Встановлення та налаштування основних фронтенд-компонентів 64

4.5.1. Встановлення Vue.js через використання Vue CLI 64

4.5.2. Створення та налаштування проєкту Vue.js 64

4.5.3. Налаштування Vue Router та запуск проєкту 64

4.5.4. Інтеграція фронтенду з бекендом та осовне налаштування Vue 65

4.6. Встановлення та налаштування Swagger для документування API 66

4.6.1. Процедура встановлення Swagger для Strapi: 66

4.6.2. Налаштування CORS (Cross-Origin Resource Sharing): 67

4.6.3. Перевірка роботи запитів через Swagger 68

4.7. Перевірка роботи запитів через Postman 69

4.8. Створення компонентів Vue для проєкту та їх налаштування 71

4.8.1. Компонент Login.vue 72

4.8.2. Компонент Register.vue 73

4.8.3. Компонент Home.vue 74

4.8.4. Компонент SideMenu.vue 74

4.8.5. Компонент UsersList.vue 75

4.8.6. Компонент Projects.vue 76

4.8.7. Компонент Boards.vue 76

4.8.8. Компонент TaskModal.vue 78

4.9. Висновки до розділу 4 79

ВИСНОВКИ 82

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ ТА ЛІТЕРАТУРИ 85

ДОДАТКИ 87

Додаток А 87

Додаток Б 101

ВСТУП

**Актуальність теми.** Ефективне управління задачами є ключовим фактором успішної реалізації IT-проєктів у сучасному світі. З розвитком інформаційних технологій та збільшенням складності програмних продуктів, управління задачами стає дедалі більш критичним. Сучасні IT-проєкти характеризуються високим рівнем динамізму, непередбачуваними змінами вимог та потребою в швидкому реагуванні на ринкові виклики.

Неефективне управління задачами може призвести до перевищення бюджету, затримок у виконанні проєктів та зниження якості кінцевого продукту. Традиційні підходи до управління проєктами, такі як Waterfall, часто не відповідають вимогам сучасних проєктів через їхню жорстку структуру та недостатню гнучкість. Це стимулює пошук нових методологій та інструментів, які б забезпечували адаптивність та ефективність управління задачами.

Зростання обсягів даних, інтеграція хмарних технологій та розподілених систем створюють додаткові виклики для управління IT-проєктами. Команди розробників потребують інструментів, які б забезпечували прозорість процесів, покращували комунікацію та сприяли підвищенню продуктивності. Важливим є також забезпечення можливості масштабування систем управління задачами відповідно до зростаючих потреб проєкту.

Існує значна кількість інструментів для управління задачами, проте багато з них не враховують специфіку сучасних IT-проєктів або є надто складними у використанні. Це підкреслює необхідність розробки нових систем управління задачами, які б поєднували в собі простоту використання, гнучкість та відповідність сучасним технологічним тенденціям.

Таким чином, дослідження в напрямку розробки ефективних систем управління задачами є актуальним і відповідає потребам сучасної IT-індустрії. Це сприятиме оптимізації робочих процесів, підвищенню якості програмних продуктів та конкурентоспроможності компаній на ринку інформаційних технологій.

**Об’єктом дослідження** є процеси організації та управління задачами в рамках IT-проєктів.

**Предметом дослідження** є принципи, підходи та інструменти розробки систем управління задачами, які відповідають сучасним викликам в IT-індустрії.

**Метою дослідження** є створення концептуальної основи та програмної реалізації системи управління задачами, яка поєднує гнучкість методології Kanban з ефективними технологічними рішеннями, що забезпечують оптимізацію робочих процесів, підвищення продуктивності команд та прозорість виконання задач.

**Завдання дослідження:**

* провести аналіз існуючих методологій та інструментів управління задачами в IT-проєктах;
* визначити функціональні та нефункціональні вимоги до системи управління задачами;
* обґрунтувати вибір методології Kanban для розробки системи;
* розробити концептуальну модель системи, визначити основні компоненти та їх взаємодію;
* вибрати технологічний стек для реалізації системи та обґрунтувати його вибір;
* реалізувати прототип системи та провести його тестування;
* оцінити ефективність запропонованого рішення та сформулювати рекомендації щодо його впровадження.

**Наукова новизна отриманих результатів** **дослідження** полягає у формуванні концепції системи управління задачами, адаптованої до специфіки сучасних IT-команд, з урахуванням особливостей методології Kanban та сучасних технологічних рішень. У дослідженні:

* взначено ключові аспекти для досягнення гнучкості та ефективності у роботі команд;
* запропоновано модель системи, яка інтегрує передові практики управління проєктами та новітні технології;
* розроблено підходи до побудови архітектури системи, яка відповідає вимогам масштабованості, безпеки та зручності інтеграції в існуючі процеси.

**Практичне значення дослідження** полягає у можливості використання запропонованої концепції та програмної реалізації для розробки інструментів управління задачами, які задовольняють потреби команд у різних IT-проєктах. Застосування таких рішень сприяє:

* скороченню часу на реалізацію проєктів завдяки оптимізації робочих процесів;
* зниженню витрат на розробку шляхом підвищення ефективності командної роботи;
* підвищенню якості взаємодії між учасниками команд через покращену комунікацію та прозорість процесів;
* покращенню результативності виконання завдань завдяки гнучкому управлінню та контролю над задачами.

**Апробація результатів дослідження.** Результати дослідження були апробовані через низку заходів та публікацій:

* + стаття у збірнику студентського наукового товариства Ніжинського державного університету імені Миколи Гоголя «Вісник студентського наукового товариства», Випуск 30, 2024 р. на тему «Оптимізація управління ІТ-проєктами: основні принципи і можливі інструменти для розробки гнучкої системи на основі Канбан» [1];
  + тези до Всеукраїнської науково-практичної конференції «Математичні, природничі, комп’ютерні науки та науки про управління, технології, навчання: науково-практичні рішення та підходи молодих науковців», що відбулася 28 листопада 2024 року на базі факультету математики, природничих наук та технологій Центральноукраїнського державного університету імені Володимира Винниченка у м. Кропивницькому, на тему «Application Programming Interface (API): основи та сучасні підходи», де представлено розроблені концепції інтеграції API в системи управління задачами [2];
  + доповідь на студентській науково-практичній конференції «Молодь у науці», яка відбувалася 16-24 травня 2024 року у Ніжинському державному університеті імені Миколи Гоголя;
* демонстрація архітектури та функціоналу системи: концептуальна модель системи була представлена колегам по роботі в копманїї Nova Digital під час внутрішньої презентації. Отримано позитивні відгуки щодо практичності і перспективності запропонованих рішень;
* практична перевірка: архітектура системи та її ключові елементи були протестовані у вигляді прототипу, що дозволило підтвердити відповідність рішень реальним умовам використання.

Ці заходи не лише підтвердили наукову обґрунтованість роботи, але й продемонстрували її практичну цінність для впровадження у IT-проєктах.

РОЗДІЛ 1

ТЕОРЕТИЧНИЙ ОГЛЯД ІСНУЮЧИХ МЕТОДОЛОГІЙ ТА ПІДХОДІВ У СФЕРІ СИСТЕМ УПРАВЛІННЯ ЗАДАЧАМИ ДЛЯ IT-ПРОЄКТІВ

* 1. Вступ до методологій управління проєктами

У сфері управління IT-проєктами існує широкий спектр методологічних підходів, кожен з яких має свої особливості, переваги та недоліки. Вибір конкретної методології залежить від специфіки проєкту, вимог замовника, складу команди та багатьох інших факторів. До найбільш відомих методологій належать Waterfall, Agile, Scrum, Kanban та Extreme Programming (XP). У цьому розділі буде проведено детальний аналіз цих підходів з метою визначення найбільш оптимального для розробки системи управління задачами.

* 1. Основний огляд підходу "Водоспад" (Waterfall)

Методологія "Водоспад" або Waterfall є одним із найстаріших підходів до розробки програмного забезпечення, що залишається актуальним у певних сферах і сьогодні. Вона виникла у 1970-х роках, коли Вінстон Ройс вперше представив цю модель як спосіб ефективного управління складними проєктами у сфері програмування [3]. Її концепція базується на структурованому та послідовному виконанні етапів, що є характерним для індустрій, таких як виробництво та будівництво, звідки цей підхід і був запозичений. У цих галузях проєкти зазвичай мають чітку структуру та передбачають виконання робіт у суворій послідовності, де кожен етап слугує основою для наступного.

Цей підхід передбачає поділ проєкту на кілька послідовних етапів, таких як збір вимог, аналіз, проєктування, розробка, тестування, впровадження та підтримка. Кожен етап завершується повністю, перш ніж розпочнеться наступний. Така структура забезпечує високу передбачуваність і дозволяє детально планувати проєкт на самому початку, включаючи всі ресурси, строки та результати. Крім того, методологія "Водоспад" робить великий акцент на документуванні всіх етапів, що забезпечує формалізацію вимог, прийнятих рішень і ключових показників успіху. Завдяки цьому підходу створюється детальна проєктна документація, яка служить довідковою базою для всіх учасників процесу і полегшує підтримку та передачу проєкту іншим командам.

Важливо зазначити, що в межах Waterfall вимоги до продукту зазвичай визначаються ще на початковій стадії і залишаються незмінними до завершення проєкту. Така стабільність особливо корисна для проєктів із чітко визначеними цілями та мінімальною ймовірністю змін, наприклад, у галузях із високим рівнем регуляції, таких як аерокосмічна, оборонна чи медична промисловість. Разом із тим, жорстка послідовність етапів має і свої обмеження. Негнучкість методології може стати значною проблемою у випадках, коли вимоги змінюються або не були повністю зрозумілими на початку, оскільки повернення до попередніх етапів пов'язане зі значними витратами часу і ресурсів.

Таким чином, методологія "Водоспад" є ефективним інструментом для управління проєктами з чіткими і стабільними вимогами, проте її застосування потребує ретельного планування та врахування можливих ризиків, пов'язаних із обмеженою гнучкістю та можливістю врахування змін на пізніх етапах. Це можуть бути проєкти у сферах з високим рівнем регуляції, наприклад, в авіації, космічній чи оборонній промисловості [4].

Основні характеристики Waterfall:

* пслідовність етапів: поєкт розбивається на логічні послідовні етапи: збір вимог, дизайн, реалізація, тестування, впровадження та підтримка;
* документування: кожен етап супроводжується детальною документацією, що забезпечує формалізацію вимог та рішень.
* фіксовані вимоги: вимоги до системи визначаються на початку проєкту та залишаються незмінними протягом усього циклу розробки.

Переваги Waterfall:

* постота управління: чітка структура дозволяє легко планувати та контролювати хід проєкту;
* визначеність результатів: кожен етап має чіткі вихідні дані, що спрощує оцінку прогресу;
* документована база: наявність детальної документації полегшує підтримку та передачу проєкту іншим командам.

Недоліки Waterfall:

* негнучкість: зміни вимог на пізніх етапах є складними та вартісними;
* ризик невідповідності потребам клієнта: користувачі бачать кінцевий продукт лише після завершення всіх етапів, що може призвести до невідповідності очікувань;
* відкладене тестування: помилки можуть бути виявлені лише на стадії тестування, що збільшує вартість їх виправлення;
* обмежене ззастосування Waterfall.
  1. Основний огляд підходу Agile

Agile виник як відповідь на недоліки традиційних методологій, таких як Waterfall, наприкінці 1990-х років, коли розробники програмного забезпечення дедалі частіше стикалися з потребою більшої гнучкості та швидкості у реагуванні на змінні вимоги замовників. Традиційні підходи, які наголошували на жорсткій послідовності етапів і фіксації вимог на початкових стадіях, виявилися малоефективними у швидко змінюваних умовах сучасного ринку. У відповідь на ці виклики у 2001 році група провідних розробників, які вже використовували менш формалізовані та більш гнучкі методи управління проєктами, зібралася у містечку Сноумасс, штат Юта, і сформулювала "Маніфест гнучкої розробки програмного забезпечення" (Agile Manifesto). Цей маніфест визначив ключові принципи та цінності Agile, які з того часу стали фундаментом для низки методологій гнучкої розробки [5].

Основою Agile є прагнення поставити клієнта та його потреби у центрі процесу розробки, забезпечуючи високу адаптивність до змін та швидкий цикл зворотного зв’язку. Цей підхід відмовляється від жорстких регламентів, властивих традиційним моделям, натомість акцентуючи увагу на ітеративному підході до створення продукту, постійній взаємодії з клієнтом і залученні самоорганізованих команд. Agile вважає, що зміни є невід’ємною частиною будь-якого проєкту, і тому замість їх уникнення пропонує механізми для ефективного реагування та інтеграції таких змін на будь-якому етапі розробки.

Цінності, задекларовані в Agile Manifesto, закликають розробників надавати перевагу індивідуальній взаємодії над суворими процесами, робочому програмному забезпеченню над вичерпною документацією, співпраці із замовником над формальними контрактами та готовності до змін над дотриманням початкового плану. Ці принципи дозволяють створювати гнучкі та ефективні процеси розробки, які адаптуються до змінних умов і зосереджуються на найважливішому – задоволенні потреб користувачів і швидкому наданні їм цінності.

Основні принципи Agile:

* ітеративна розробка: проєкт розбивається на короткі цикли (ітерації або спринти), кожен з яких завершується отриманням робочого продукту;
* співпраця з клієнтом: постійна взаємодія з замовником для уточнення вимог та отримання зворотного зв'язку;
* готовність до змін: легке впровадження змін на будь-якому етапі розробки;
* самоорганізовані команди: команди мають повноваження приймати рішення щодо реалізації.

Переваги Agile [6]:

* гучкість та адаптивність: швидка реакція на зміни вимог та ринкових умов;
* підвищена продуктивність: зосередження на пріоритетних завданнях та мінімізація непотрібної роботи;
* покращена комунікація: тісна співпраця між членами команди та замовником;
* раннє виявлення проблем: постійне тестування та інтеграція дозволяють швидко виявляти та виправляти помилки.

Недоліки Agile:

* мнша передбачуваність: складно прогнозувати кінцеві терміни та бюджет;
* вимога до високої кваліфікації команди: потрібні досвідчені фахівці, здатні до самоорганізації;
* можливість відхилення від мети: без чіткої документації є ризик втрати фокусу на кінцевому продукті.

Найпоширеніші підходи в рамках Agile:

* Scrum: використовує фіксовані ітерації та чітко визначені ролі;
* Kanban: зосереджується на візуалізації процесів та обмеженні роботи в процесі;
* Extreme Programming (XP): акцентує увагу на технічній майстерності та якості коду.

1.3.1. Scrum як фреймворк Agile

Scrum є одним із найпопулярніших фреймворків у межах методології Agile, який призначений для управління складними проєктами та контролю за їх виконанням [7]. Цей підхід було розроблено з метою покращення командної роботи та підвищення гнучкості процесів розробки програмного забезпечення. Його основою є ітеративний та інкрементальний підхід, що дозволяє розбивати роботу на короткі цикли, які називаються спринтами. Кожен спринт триває фіксований період часу, зазвичай від одного до чотирьох тижнів, і завершується створенням готового до випуску інкременту продукту. Така структура забезпечує регулярне отримання зворотного зв’язку від замовника, а також дозволяє швидко адаптуватися до змін у вимогах чи умовах ринку.

Однією з ключових переваг Scrum є чіткий розподіл ролей у команді, що сприяє ефективній організації процесу. У межах цього фреймворка виділяються три основні ролі: Scrum-майстер, власник продукту (англ. Product Owner) та команда розробників. Scrum-майстер відповідає за забезпечення дотримання принципів Scrum, допомагає команді усувати перешкоди та підтримує ефективну взаємодію між учасниками процесу. Власник продукту керує беклогом продукту – списком функціональних вимог і завдань, що мають бути реалізовані, та визначає пріоритети для кожного елемента. Команда розробників складається з самоорганізованої групи фахівців, яка виконує основну роботу зі створення продукту. Ці фахівці можуть мати різні спеціалізації, але в межах Scrum вони працюють як єдиний цілий організм. Важливо, що в межах Scrum всі члени команди є рівноправними, а рішення приймаються колективно.

Scrum також визначає три основні артефакти, які забезпечують прозорість і контроль над проєктом: беклог продукту, беклог спринту та інкремент продукту. Беклог продукту є основним інструментом планування та містить перелік усіх вимог, функціоналу та ідей, які потрібно реалізувати. Він є динамічним документом, що постійно еволюціонує та відображає змінні потреби ринку та клієнтів. Власник продукту регулярно переглядає та оновлює беклог, щоб забезпечити актуальність та пріоритетність завдань. Беклог спринту формується перед кожним спринтом і включає завдання, які команда планує виконати у визначений період. Під час планування спринту команда розробників разом з власником продукту визначає, які елементи з беклогу продукту можуть бути реалізовані в межах спринту, враховуючи потужність команди та складність завдань. Інкремент продукту – це кінцевий результат кожного спринту, що являє собою завершену та потенційно готову до випуску частину продукту. Цей інкремент повинен відповідати заздалегідь визначеним критеріям готовності, що забезпечує високу якість та функціональність продукту.

Процес роботи в Scrum організовується через низку подій, які сприяють плануванню, виконанню та оцінці виконаної роботи. Планування спринту є першою подією кожного циклу, де команда визначає цілі та завдання, що будуть виконуватися. Щоденні "cкрами" – це короткі зустрічі, що проводяться щодня, і дозволяють учасникам синхронізувати зусилля, обговорити прогрес і визначити можливі перешкоди. Наприкінці спринту команда проводить огляд, демонструючи виконану роботу, і ретроспективу, де аналізує процеси та шукає способи їхнього покращення. В подальшому, команда розробляє конкретні дії для вдосконалення роботи в наступних спринтах, що сприяє постійному розвитку та підвищенню ефективності.

Популярність Scrum значною мірою обумовлена його перевагами, серед яких особливо виділяються прозорість процесів, орієнтація на цінність для клієнта та постійне підвищення продуктивності команди. Регулярні зустрічі, звітність і постійна взаємодія між командою та замовником забезпечують відкритість і чітке розуміння прогресу проєкту. Однак Scrum також має певні недоліки, серед яких необхідність суворого дотримання правил і процедур, а також складність масштабування цього підходу для великих команд чи організацій. Незважаючи на це, Scrum залишається одним із найефективніших інструментів для управління гнучкими проєктами, особливо у сфері розробки програмного забезпечення.

1.3.2. Kanban як метод управління задачами

Kanban є методом управління проєктами, що зосереджується на візуалізації процесів, оптимізації потоку роботи та поступовому вдосконаленні [8]. Цей підхід виник у середині 20-го століття в рамках виробничої системи Toyota, де він використовувався для управління потоками матеріалів на виробничих лініях. Його назва походить від японського слова "канбан", що означає "вивіска" або "табличка", які використовувалися для позначення поточних потреб у матеріалах. Згодом Kanban був адаптований для використання у сфері інформаційних технологій та став популярним методом управління проєктами завдяки своїй простоті, ефективності та універсальності.

Однією з ключових особливостей Kanban є його орієнтація на візуалізацію роботи за допомогою Kanban-дошок. Ці дошки надають чітке уявлення про всі завдання, які виконує команда, а також про їхній поточний статус у процесі роботи. Kanban-дошка розділена на стовпці, які представляють різні стадії виконання завдань, наприклад, "Заплановано", "В роботі" та "Завершено". Кожна задача розміщується на дошці у вигляді картки, що дозволяє команді бачити загальну картину і оперативно реагувати на зміни. Це сприяє покращенню комунікації всередині команди та забезпечує прозорість процесів. Візуалізацію Kanban-дошки представлено на рисунку 1.1.

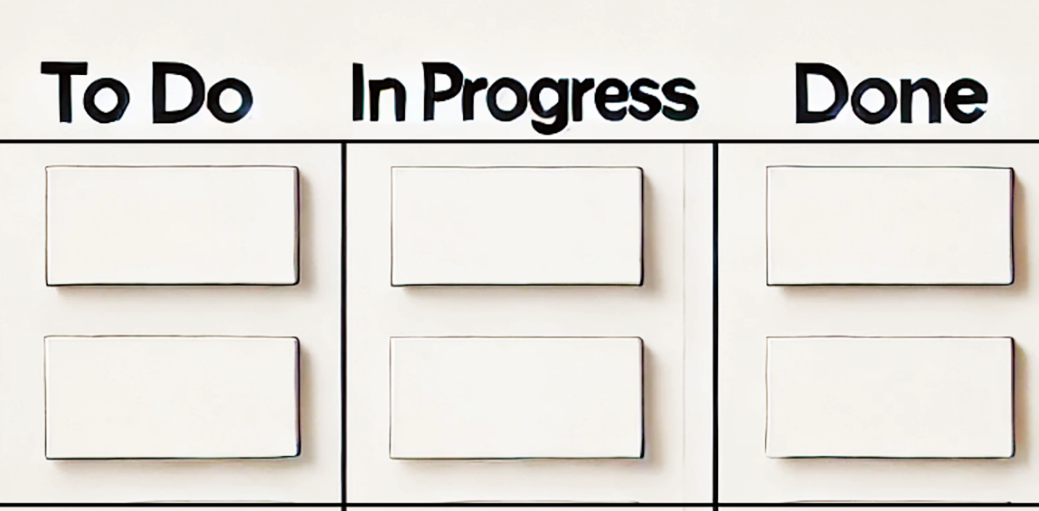


Рисунок 1.1 – Kanban-дошка для організації робочого процесу.

Крім того, Kanban-дошки можуть бути розширені додатковими елементами, такими як кольорове кодування карток для позначення пріоритетності завдань або використання міток для категоризації роботи. Це дозволяє команді швидко ідентифікувати критичні завдання та сфокусувати зусилля на їхньому виконанні.

Також, складовою Kanban є обмеження роботи в процесі, які визначають максимальну кількість завдань, що можуть перебувати в певній стадії одночасно. Ці обмеження допомагають уникати перевантаження команди, забезпечують рівномірний розподіл роботи та сприяють більш ефективному управлінню часом і ресурсами. Управління потоком завдань є центральним аспектом Kanban, оскільки цей метод фокусується на забезпеченні постійного руху задач через процес, мінімізуючи затримки та простої.

Kanban наголошує на важливості чітко визначених політик процесу. Це означає, що всі правила, процедури та критерії виконання роботи повинні бути зрозумілими та прозорими для всіх учасників. Постійне вдосконалення є ще одним ключовим принципом Kanban, що передбачає регулярний аналіз процесів та впровадження змін для підвищення ефективності роботи.

Популярність Kanban значною мірою зумовлена його перевагами. Цей метод забезпечує високу гнучкість, дозволяючи командам адаптуватися до змін без порушення загального процесу. Нові завдання можуть бути додані до потоку роботи в будь-який момент, що особливо корисно в динамічних середовищах з часто змінними пріоритетами. Kanban є відносно простим у впровадженні, оскільки не вимагає радикальних змін у структурі команди чи існуючих процесах. Крім того, використання цього підходу сприяє покращенню ефективності роботи завдяки зменшенню часу циклу виконання завдань, оптимізації ресурсів та усуненню зайвих дій.

Однак, Kanban має і певні недоліки. Відсутність чітко визначених ролей та ітерацій може призвести до плутанини, якщо команда не має належної дисципліни або не дотримується встановлених політик. Без фіксованих термінів та відповідальності за певні ролі може бути складніше координувати зусилля та відстежувати прогрес. Крім того, цей метод вимагає високого рівня відповідальності та самоорганізації від команди, що може бути викликом для нових або менш досвідчених колективів.

Незважаючи на ці обмеження, Kanban залишається ефективним інструментом для управління процесами, особливо у середовищах, де потрібна висока адаптивність та зосередженість на оптимізації потоків роботи. Він успішно застосовується не тільки в розробці програмного забезпечення, але й у виробництві, маркетингу, фінансах та інших галузях. Організації, які впроваджують Kanban, часто відзначають покращення продуктивності, прозорості та співпраці в командах, що сприяє досягненню стратегічних цілей та підвищенню конкурентоспроможності на ринку.

1.3.3. Extreme Programming (XP)

Extreme Programming (XP) є однією з методологій Agile, яка робить акцент на технічній майстерності, високій якості коду та ефективній взаємодії між розробниками і замовником [9]. Вона була створена для проєктів, де вимоги можуть часто змінюватися, а кінцевий продукт повинен швидко адаптуватися до нових умов. XP передбачає часті релізи невеликих інкрементів продукту, що дозволяє замовнику регулярно отримувати результати роботи, надавати зворотний зв'язок та коригувати вимоги. Ця методологія спрямована на максимальну прозорість процесу розробки та швидке реагування на зміни. XP також підкреслює важливість постійної комунікації та співпраці між усіма учасниками проєкту, що сприяє швидкому виявленню та вирішенню проблем.

Основою XP є низка ключових практик, які забезпечують якість і стабільність продукту. Серед них – парне програмування, де два розробники працюють над одним кодом, що сприяє обміну знаннями і мінімізує ризик помилок. Цей підхід також підвищує якість коду, оскільки дві людини можуть виявити більше помилок і запропонувати кращі рішення. Розробка через тестування (англ. Test-Driven Development, TDD) передбачає створення тестів перед написанням самого коду, що гарантує відповідність програмного забезпечення вимогам. Це допомагає запобігти дефектам на ранніх стадіях та забезпечує впевненість у тому, що новий код не порушує існуючу функціональність. Неперервна інтеграція дозволяє регулярно вносити зміни в кодову базу та автоматично тестувати їх, знижуючи ризик конфліктів. Це сприяє швидкому виявленню інтеграційних проблем та підтримці стабільної версії продукту. Простий дизайн, рефакторинг коду та використання метафор для опису системи також є важливими компонентами цього підходу, які спрощують комунікацію та підтримку проєкту. Простий дизайн означає уникнення складності та надмірної абстракції, зосереджуючись на поточних вимогах. Рефакторинг передбачає постійне покращення коду без зміни його функціональності, що підвищує його якість та полегшує підтримку. Використання спільних метафор допомагає команді мати спільне розуміння системи та полегшує обговорення технічних аспектів.

Попри значні переваги XP, такі як висока якість коду, швидка реакція на зміни та тісна співпраця із замовником, методологія має і свої недоліки. Вона вимагає від команди високої кваліфікації, дисципліни та готовності до інтенсивної взаємодії. Команда повинна бути здатна дотримуватися строгих практик, таких як TDD і парне програмування, що може бути викликом для менш досвідчених розробників. Для великих або розподілених команд впровадження XP може бути складним, оскільки методологія передбачає тісну комунікацію між учасниками. У таких ситуаціях може виникнути потреба в адаптації практик XP або поєднанні їх з іншими методологіями. Тим не менш, XP є ефективним інструментом для проєктів з високими технічними вимогами, де якість коду і гнучкість мають вирішальне значення.

1.3.4. Порівняльний аналіз гнучких методологій

Для більш наочного розуміння особливостей розглянутих методологій наведемо порівняльну таблицю 1.1.

Висновки з порівняння:

* Scrum підходить для команд, які цінують структуру та чіткі рамки. Він забезпечує регулярність та передбачуваність, але може бути менш гнучким у порівнянні з Kanban;
* Kanban надає більшу гнучкість та дозволяє командам легко адаптуватися до змін без необхідності суворих ітерацій. Він особливо корисний для команд з непередбачуваним обсягом роботи;
* Extreme Programming акцентує увагу на технічній досконалості та підходить для команд, які прагнуть досягти високої якості коду. Він вимагає високого рівня дисципліни та досвіду.

Таблиця 1.1 – Порівняння гнучких методологій

|  |  |  |  |
| --- | --- | --- | --- |
| **Параметр** | **Scrum** | **Kanban** | **Extreme Programming (XP)** |
| **Основний фокус** | Ітеративна розробка через спринти | Неперервний потік роботи | Якість коду та адаптивність до змін |
| **Ролі у команді** | Scrum-майстер, власник продукту, команда | Ролі не формалізовані | Розробники, клієнт активно залучений |
| **Довжина ітерацій** | Фіксовані спринти  (2-4 тижні) | Немає фіксованих ітерацій | Короткі ітерації  (1-2 тижні) |
| **Метрики та вимірювання** | Велосіті команди, бурндаун-чарти | WIP ліміти, час циклу | Кількість пройдених тестів, частота рефакторингу |
| **Планування** | Планування спринтів | Планування за потребою | Ітераційне планування з клієнтом |
| **Зміни у проєкті** | Обмежені між спринтами | Можуть бути внесені будь-коли | Приймаються в будь-який момент |
| **Основні артефакти** | Беклог продукту, беклог спринту, інкремент продукту | Канбан-дошка з завданнями | Код, юніт-тести, система неперервної інтеграції |
| **Основні практики** | Щоденні скрами, ретроспективи | Візуалізація роботи, управління потоком | Парне програмування, TDD, рефакторинг |

Продовження таблиці 1.1 – Порівняння гнучких методологій

|  |  |  |  |
| --- | --- | --- | --- |
| **Параметр** | **Scrum** | **Kanban** | **Extreme Programming (XP)** |
| **Культура та цінності** | Співпраця, відповідальність, прозорість | Гнучкість, неперервне покращення | Співпраця, якість, сміливість, простота |
| **Залучення клієнта** | Регулярні зустрічі та демонстрації | Менш формалізовано, за потребою | Постійна залученість та зворотний зв'язок |
| **Управління проєктом** | Власник продукту, Scrum-майстер | Самоорганізація команди | Відсутність традиційних ролей менеджерів |
| **Співпраця в команді** | Інтенсивна співпраця всередині спринтів | Співпраця на основі поточних потреб | Парне програмування, колективна відповідальність |
| **Інструменти та техніки** | Scrum-дошки, бурндаун-чарти | Канбан-дошки з WIP лімітами | Інструменти для TDD, рефакторингу, CI |
| **Адаптація до змін** | Після кожного спринту | Неперервно | Постійна адаптація |
| **Основна мета** | Доставка інкременту продукту | Оптимізація потоку роботи | Доставка високоякісного продукту |

* 1. Висновки по розділу 1

У першому розділі було проведено детальний теоретичний огляд існуючих підходів у сфері систем управління задачами для IT-проєктів. Розглянуто традиційну методологію "Водоспад" (Waterfall) та гнучкі методології, такі як Agile, Scrum, Kanban та Extreme Programming (XP). Кожна з цих методологій має свої особливості, переваги та недоліки, які впливають на їхню придатність до різних типів проєктів та команд.

Основні висновки розділу.

1. Традиційна модель Waterfall забезпечує чітку структуру та послідовність етапів розробки, що підходить для проєктів з фіксованими та незмінними вимогами. Проте її негнучкість і складність внесення змін на пізніх етапах робить її менш придатною для сучасних динамічних IT-проєктів, де вимоги можуть швидко змінюватися.
2. Гнучкі методології Agile виникли як відповідь на потребу в більшій адаптивності та швидкості реагування на зміни. Вони акцентують увагу на ітеративній розробці, постійній співпраці з клієнтом та самоорганізованих командах. Це робить їх більш придатними для сучасних проєктів, але вимагає високої кваліфікації та дисципліни від команди.
3. Scrum як фреймворк Agile надає структурований підхід з чітко визначеними ролями, артефактами та подіями. Він сприяє прозорості процесів та регулярному отриманню зворотного зв'язку від клієнта. Однак, потребує суворого дотримання правил та може бути складним у масштабуванні для великих команд.
4. Kanban фокусується на візуалізації роботи, управлінні потоком та постійному вдосконаленні. Його гнучкість та простота впровадження роблять його привабливим для команд, які прагнуть оптимізувати свої процеси без радикальних змін. Проте, відсутність чітких ітерацій та ролей може вимагати високого рівня самоорганізації від команди.
5. Extreme Programming (XP) акцентує увагу на технічній майстерності, високій якості коду та тісній співпраці з клієнтом. Він підходить для проєктів, де вимоги часто змінюються, і якість коду є критичною. Впровадження XP вимагає високої кваліфікації та дисципліни команди, що може бути складним для деяких організацій.
6. Порівняльний аналіз методологій показав, що немає універсального підходу, який би підходив для всіх типів проєктів. Вибір методології залежить від специфіки проєкту, вимог замовника, складу та досвіду команди, а також від корпоративної культури організації.

Проведений аналіз методологій управління задачами дозволяє зробити висновок, що для успішної реалізації сучасних IT-проєктів необхідно обирати підхід, який забезпечує баланс між гнучкістю та структурованістю процесів. Гнучкі методології, такі як Agile, особливо Scrum та Kanban, краще відповідають вимогам динамічного ринку, де швидкість реакції на зміни та тісна співпраця з клієнтом є критичними факторами успіху.

РОЗДІЛ 2

ОГЛЯД ДЕЯКИХ СУЧАСНИХ ПРОГРАМНИХ РЕАЛІЗАЦІЙ У СФЕРІ СИСТЕМ УПРАВЛІННЯ ЗАДАЧАМИ ДЛЯ IT-ПРОЕКТІВ

2.1. Вступ до систем управління задачами

У сучасному світі управління IT-проєктами неможливе без застосування ефективних інструментів, які допомагають командам синхронізувати свої зусилля, відслідковувати прогрес та взаємодіяти один з одним. З безлічі доступних рішень, певні програмні інструменти виділяються своєю популярністю та ефективністю, завоювавши довіру розробників і менеджерів проєктів у всьому світі. Наведемо огляд деяких з найбільш використовуваних інструментів у сфері систем управління задачами для IT-проєктів.

2.2. Огляд основних програмних рішень

2.2.1. JIRA від Atlassian

JIRA від Atlassian — це надзвичайно потужний інструмент для управління проєктами, задачами та помилками, який став стандартом де-факто в багатьох IT-компаніях та командах розробників по всьому світу [10]. Розроблений з метою оптимізації процесів розробки програмного забезпечення, JIRA дозволяє командам ефективно відстежувати весь життєвий цикл завдань від моменту створення до вирішення.

Система не лише дозволяє організувати робочі процеси, але й забезпечує прозорість у взаємодії команди, полегшуючи співпрацю між різними ролями в проєкті. Використовуючи такі функції, як гнучке планування, дошки завдань, автоматизація процесів та інтеграція зі сторонніми сервісами, JIRA стає важливим інструментом для досягнення високої продуктивності.

На рисунку 2.1 наведено приклад інтерфейсу планування JIRA, що демонструє використання інструменту для відстеження прогресу задач у рамках Agile-процесів. У прикладі зображено хронологію виконання задач із зазначенням статусів, дедлайнів та залежностей між завданнями.

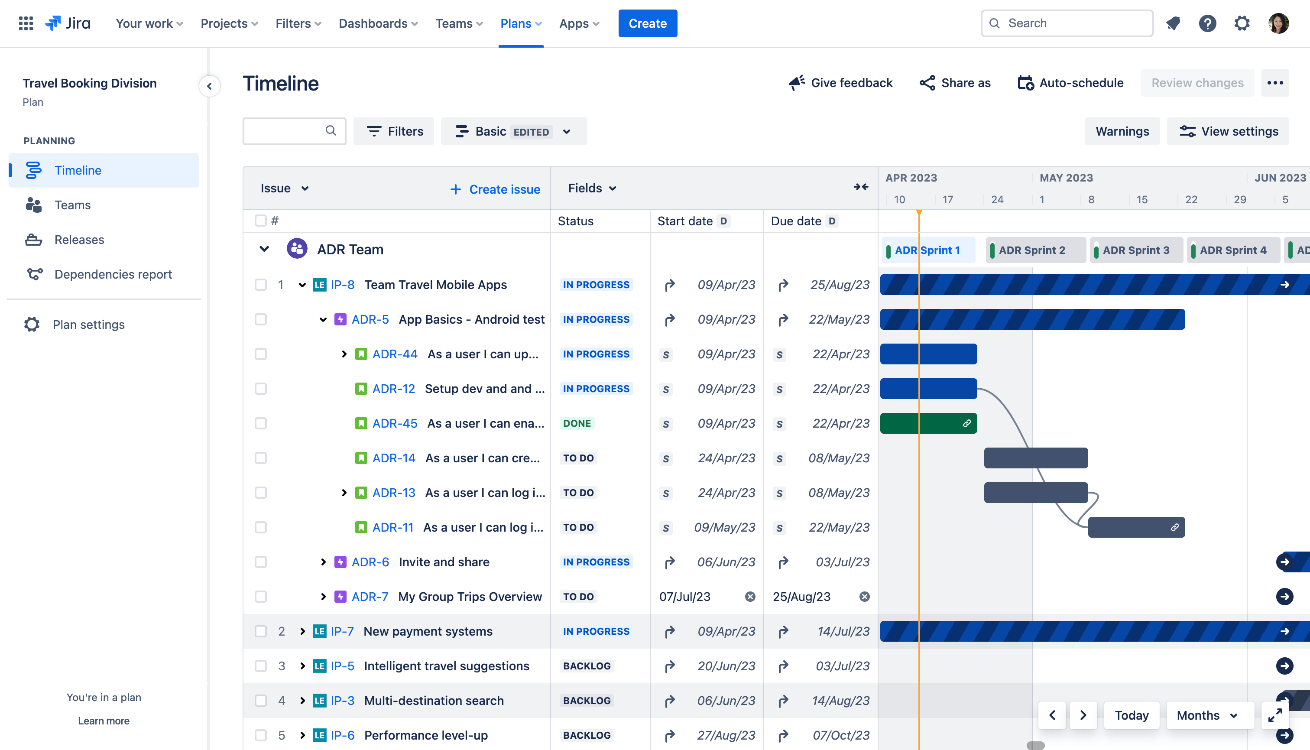


Рисунок 2.1 – Інтерфейс JIRA: приклад планування задач у хронологічному вигляді з прив'язкою до спринтів.

Основні можливості:

* гучкість у налаштуванні: JIRA пропонує величезні можливості для кастомізації, що дозволяє адаптувати інструмент під специфіку будь-якого проєкту чи команди, включно з налаштуванням робочих процесів, типів завдань, полів та дашбордів;
* підтримка методологій: JIRA підтримує різноманітні гнучкі методології, такі як Scrum та Kanban, дозволяючи командам використовувати спринти, беклоги, дошки завдань та інші інструменти для ефективного управління проєктами;
* інтеграція з іншими інструментами: JIRA може інтегруватися з великою кількістю інших інструментів та сервісів, включаючи системи контролю версій (наприклад, Git), системи неперервної інтеграції, інші продукти Atlassian (наприклад, Confluence, Bitbucket) та багато інших;
* звітність та аналітика: розширені функції звітності дозволяють командам аналізувати продуктивність, відстежувати прогрес та ідентифікувати проблемні зони в робочих процесах;
* безпека та масштабованість: JIRA забезпечує високий рівень безпеки та підтримує масштабування для використання великими організаціями зі складними проєктами.

Сценарії використання:

* розробка програмного забезпечення: управління проєктами, фіксація помилок, відстеження нових функцій та вдосконалень;
* управління проєктами та задачами: планування, відстеження та аналіз проєктних завдань і ресурсів;
* Agile-менеджмент: підтримка Agile-процесів, спринтів, ретроспектив тощо.

JIRA стала незамінним інструментом для багатьох команд, які прагнуть підвищити ефективність своєї роботи, забезпечити високу якість продуктів та успішне завершення проєктів.

2.2.2. Trello

Trello — це візуальний інструмент для управління проєктами, що базується на методології Kanban [11]. Він дозволяє користувачам створювати дошки для різних проєктів, організовувати завдання у вигляді карток та переміщати їх між колонками, що відображають різні стадії виконання робіт. Простота та інтуїтивно зрозумілий інтерфейс роблять Trello популярним серед різноманітних команд та індивідуальних користувачів.

Trello надає можливість інтеграції з іншими сервісами, такими як Google Drive, Slack, та іншими інструментами, що дозволяє покращити співпрацю в командах. Інструмент особливо зручний для застосування у невеликих командах або для індивідуального планування завдань. Основні функції включають створення чек-листів, додавання дедлайнів, коментарів, прикріплення файлів та багато іншого.

На рисунку 2.2 продемонстровано приклад типової дошки в Trello з різними колонками, такими як "To do", "Doing" та "Done", що відображають стадії виконання завдань.

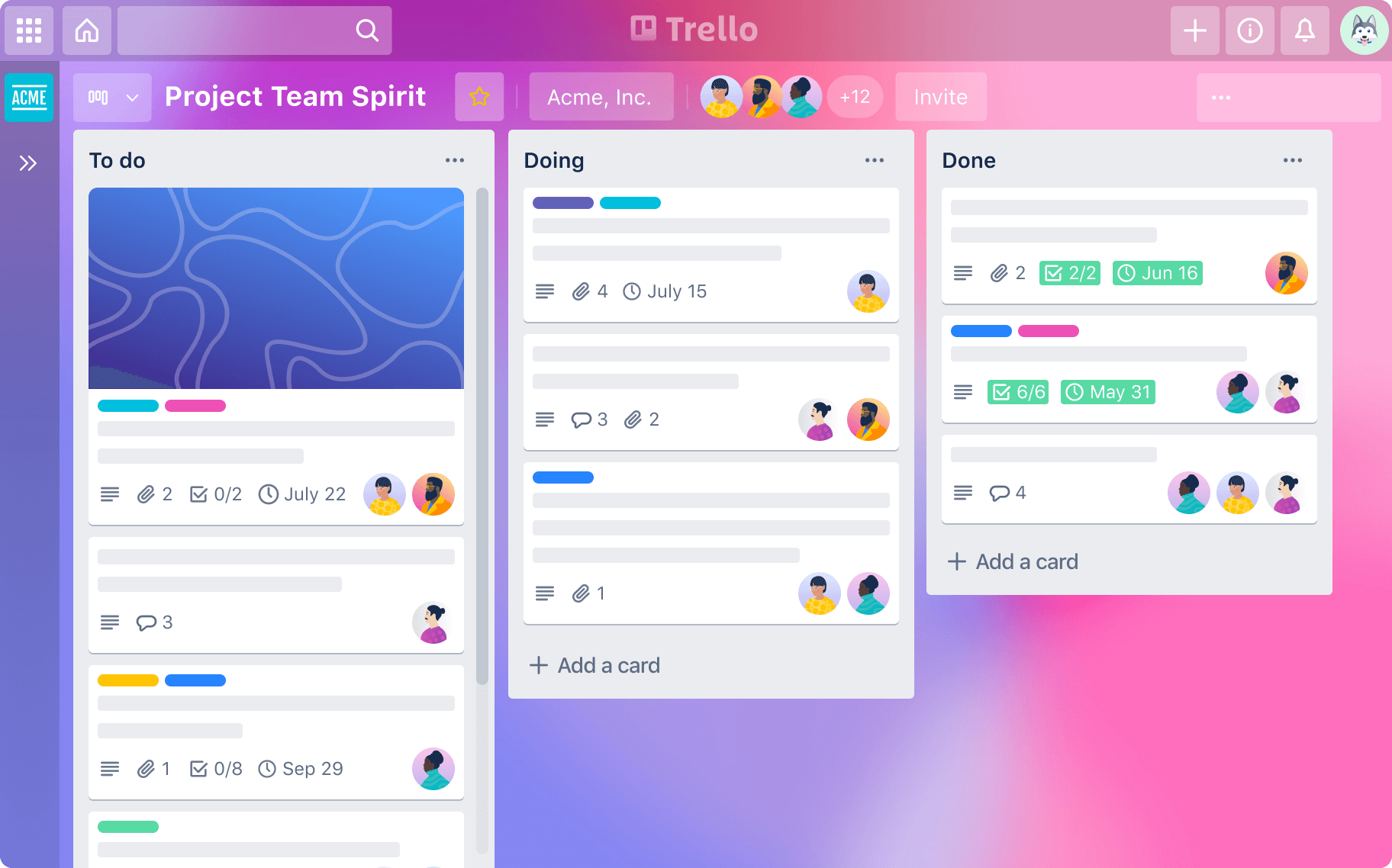


Рисунок 2.2 – Інтерфейс JIRA: приклад планування задач у хронологічному вигляді з прив'язкою до спринтів

Основні можливості:

* візуальне планування: Trello дозволяє візуалізувати весь проєкт на одній дошці, де кожна картка відображає конкретне завдання, а колонки представляють різні етапи процесу;
* гнучкість та адаптивність: користувачі можуть налаштовувати дошки, колонки та картки, додавати мітки, встановлювати терміни виконання та призначати відповідальних осіб, що забезпечує високу адаптивність до різних проєктів та робочих процесів;
* співпраця в реальному часі: команди можуть спільно працювати над проєктами, коментувати завдання, ділитися файлами та отримувати оновлення стану в реальному часі;
* інтеграція з іншими сервісами: Trello може інтегруватися з багатьма зовнішніми інструментами та сервісами, що розширює його функціональність та підвищує продуктивність команд.

Сценарії використання:

* уравління проєктами: організація робочих процесів, відстеження прогресу завдань та координація командних зусиль;
* планування заходів: створення планів заходів, розподіл завдань та відстеження виконання;
* списки завдань: управління особистими та командними списками завдань, планування щоденних завдань;
* брейнстормінг та ідеї: колективне генерування ідей та управління творчими проєктами.

Завдяки своїй простоті та водночас потужним можливостям для організації роботи, Trello є незамінним інструментом для будь-кого, хто прагне забезпечити ефективну співпрацю та високий рівень організації проєктних процесів.

2.2.3. Asana

Asana — це високофункціональний інструмент для управління задачами та проєктами, розроблений з метою оптимізації робочих процесів та підвищення продуктивності команд [12]. Він забезпечує користувачам потужні інструменти для планування, відстеження та координації проєктних завдань, роблячи Asana ідеальним рішенням для команд будь-якого розміру та складності проєктів.

Гнучкість налаштувань і простота використання роблять Asana універсальним вибором як для малих команд, так і для великих корпорацій. Завдяки можливості створювати персоналізовані робочі простори, кожен учасник команди може адаптувати платформу під свої потреби, зберігаючи при цьому цілісність командного підходу до роботи.

Asana дозволяє налаштовувати дашборди для отримання візуалізації ключових етапів проєкту, ефективно координувати дії між функціональними командами та автоматизувати деякі повторювані процеси. Інтеграція з такими інструментами, як Google Drive, Slack, Microsoft Teams, забезпечує seamless workflow, що значно підвищує продуктивність.

На рисунку 2.3 наведено приклад інтерфейсу Asana, який демонструє, як команди можуть візуалізувати свої проєкти, відслідковувати задачі та отримувати аналітичну інформацію щодо прогресу.

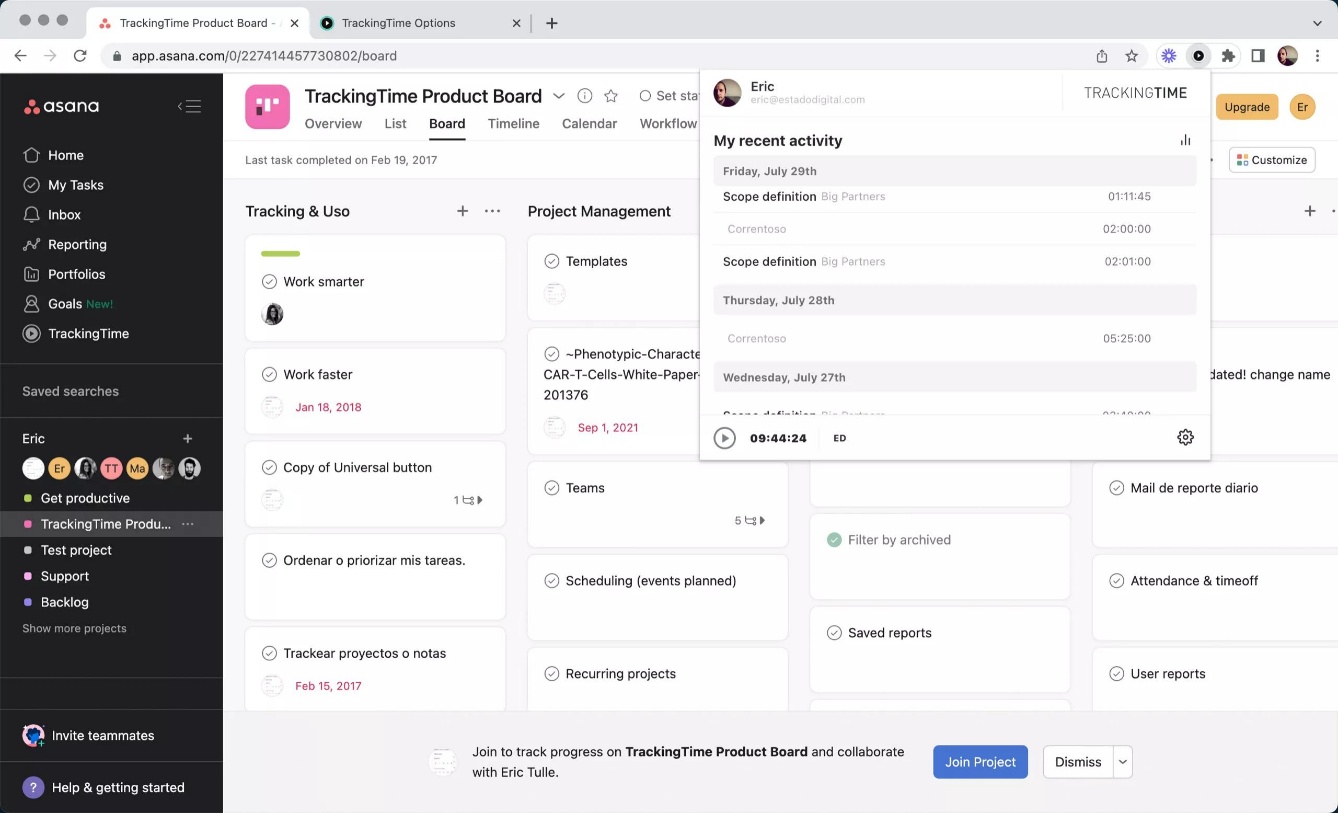


Рисунок 2.3 – Інтерфейс Asana: приклад організації завдань на командній дошці

Основні можливості:

* гучке управління завданнями: Asana дозволяє створювати проєкти, задачі та підзадачі, надаючи можливість детально планувати робочі процеси та відстежувати прогрес у реальному часі;
* співпраця в команді: завдяки інтегрованим інструментам для спілкування, таким як коментарі до завдань, команди можуть ефективно співпрацювати, обговорювати деталі проєктів та оперативно вирішувати виникаючі питання;
* персоналізовані дашборди: Asana пропонує гнучкі дашборди, які можна налаштовувати під конкретні потреби команди, забезпечуючи швидкий доступ до ключової інформації про проєкти;
* інтеграція з іншими сервісами: Asana легко інтегрується з багатьма зовнішніми інструментами та сервісами, такими як електронна пошта, календарі, файлообмінні сервіси, що робить її високоефективним рішенням для комплексного управління проєктами.

Сценарії використання:

* кмплексне управління проєктами: організація робочих процесів, відстеження прогресу та координація дій всієї команди на різних етапах проєкту;
* управління задачами: створення та відстеження короткострокових та довгострокових завдань, встановлення пріоритетів та термінів виконання;
* планування ресурсів: розподіл ресурсів, відстеження витрат часу та ефективне використання потенціалу команди для досягнення цілей проєкту;
* координація міжфункціональних команд: спрощення взаємодії між різними підрозділами та спеціалістами в рамках складних проєктів і процесів.

Asana є вибором багатьох команд і організацій, які прагнуть до високого рівня організації та ефективності у своїй роботі, надаючи всі необхідні інструменти для успішного управління проєктами та задачами.

2.2.4. Microsoft Project

Microsoft Project є одним із найвідоміших інструментів для управління проєктами, який активно використовується професійними менеджерами проєктів у всьому світі [13]. Цей продукт Microsoft був створений для забезпечення ефективного планування, організації та контролю за виконанням задач різного рівня складності. Інструмент є незамінним для великих організацій, де проєкти часто мають багатошарові структури, складні залежності та жорсткі дедлайни.

Завдяки інтуїтивно зрозумілому інтерфейсу та гнучким функціям, Microsoft Project дозволяє деталізувати плани проєктів, створювати взаємозв’язки між завданнями, розподіляти ресурси та контролювати бюджет. Його популярність зумовлена не лише високою функціональністю, але й інтеграцією з іншими продуктами Microsoft, такими як Excel, SharePoint та Teams, що робить цей інструмент надзвичайно зручним для великих корпоративних команд.

Microsoft Project підтримує різні методики управління проєктами, включаючи каскадну модель (Waterfall) та Agile, що дає змогу адаптувати його до потреб команди. Завдяки цьому інструменту можна створювати діаграми Ганта для візуалізації планів, прогнозувати ризики та формувати детальні звіти, що дозволяє стейкхолдерам приймати зважені рішення на основі актуальних даних.

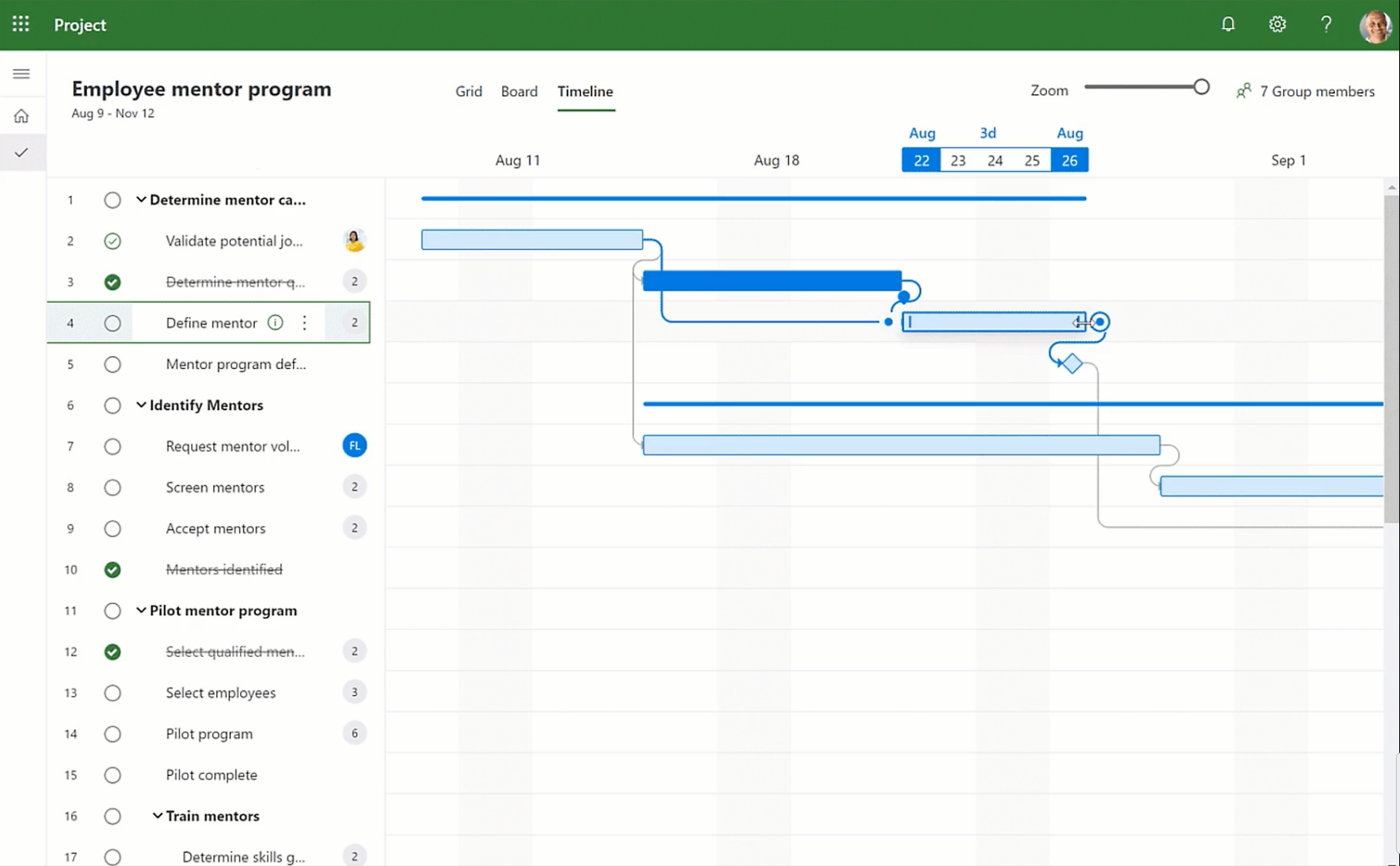


Рисунок 2.4 – Інтерфейс Microsoft Project: планування задач із використанням діаграми Ганта

На рисунку 2.4 продемонстровано приклад використання Microsoft Project для планування задач у межах програми наставництва. Інтерфейс включає використання діаграми Ганта, яка візуалізує взаємозв’язки між завданнями, їх терміни виконання та критичний шлях.

Основні можливості:

* дтальне планування проєктів: Microsoft Project дозволяє створювати детальні плани проєктів з використанням діаграм Ганта, визначати залежності між завданнями та встановлювати критичні шляхи;
* управління ресурсами: інструмент надає розширені можливості для планування та розподілу ресурсів, включаючи робочий час співробітників, обладнання та матеріали, а також дозволяє відстежувати використання ресурсів та їх наявність;
* моніторинг та звітність: завдяки потужним інструментам звітності менеджери проєктів можуть аналізувати прогрес, відстежувати виконання бюджету та оцінювати продуктивність команди;
* співпраця та інтеграція: Microsoft Project інтегрується з іншими продуктами Microsoft, такими як Office 365 та SharePoint, що забезпечує високий рівень співпраці в командах та легкий доступ до проєктної документації.

Сценарії використання:

* кмплексне управління проєктами: від планування та розподілу ресурсів до моніторингу виконання та аналізу результатів, Microsoft Project підходить для управління великими проєктами з високою складністю;
* управління портфелем проєктів: можливість одночасно керувати кількома проєктами дозволяє великим організаціям ефективно розподіляти ресурси та пріоритизувати проєкти на рівні портфеля;
* розширена звітність: генерація детальних звітів допомагає стейкхолдерам зрозуміти стан проєктів та приймати обґрунтовані рішення.

Microsoft Project відомий своєю надійністю та ефективністю в управлінні складними проєктами, що робить його незамінним інструментом для професійних менеджерів проєктів та великих організацій.

2.3. Порівняння програмних рішень

2.3.1. Підтримка методологій програмними рішеннями

JIRA від Atlassian є одним із найбільш адаптивних інструментів на ринку, призначеним для управління проєктами і задачами. Вона ідеально підходить для команд, що працюють за принципами Scrum та Kanban, пропонуючи велику кількість налаштувань для будь-якого робочого процесу [10]. Завдяки своїй гнучкості, JIRA дозволяє моделювати практично будь-яку методологію, включаючи Waterfall та Extreme Programming (XP), хоча для цього може знадобитися додаткова кастомізація.

Trello є візуально орієнтованим інструментом управління проєктами, який використовує принципи Kanban для організації завдань і проєктів [11]. Його простий інтерфейс дозволяє легко створювати, переміщати та керувати картками завдань між колонками, що представляють різні стадії процесу. Хоча Trello не має вбудованої підтримки для Waterfall або XP, його можна ефективно адаптувати для використання в різних методологіях завдяки гнучкості налаштувань і можливості інтеграції з іншими інструментами, включаючи JIRA.

Asana забезпечує широкий спектр функціональності для управління проєктами та задачами, підтримуючи основні принципи Agile та Scrum [12]. Завдяки своїй гнучкій системі задач, підзадач та проєктів, Asana дозволяє командам структурувати свою роботу таким чином, щоб вона відповідала різним робочим процесам і методологіям, включаючи XP. Інтерфейс Asana, орієнтований на забезпечення максимальної зручності та ефективності в роботі команд, робить її відмінним вибором для тих, хто шукає гнучке та потужне рішення для управління проєктами.

Microsoft Project традиційно використовується для управління проєктами за методологією Waterfall, пропонуючи потужні інструменти для детального планування, розподілу ресурсів, відстеження прогресу та аналізу ефективності проєктів [13]. Хоча програма спочатку не була розроблена для Agile-методології, вона пропонує інструменти та функції, які можуть бути адаптовані для підтримки Agile та XP в рамках гнучкого управління проєктами, особливо коли використовуються разом з іншими інструментами Microsoft та застосунками.

В таблиці 2.1 наведемо порівняння між вищезазначеними програмними рішеннями.

Результат порівняння:

* мобільні додатки: наявність мобільних застосунків для всіх чотирьох інструментів забезпечує гнучкість та доступність проєктів для команд, що постійно в русі;
* співпраця з зовнішніми користувачами: Trello та Asana виділяються можливістю легко залучати зовнішніх користувачів до співпраці над проєктами, що є важливим аспектом для роботи з клієнтами та партнерами;
* підтримка багатомовності: ця функція особливо актуальна для міжнародних команд, і JIRA, Asana та Microsoft Project пропонують широку підтримку різних мов.
* вбудовані аналітичні інструменти: потужні аналітичні можливості JIRA, Asana та Microsoft Project дозволяють глибоко аналізувати продуктивність проєктів і команд;
* кастомізація інтерфейсу: високий рівень кастомізації, який пропонують JIRA та Microsoft Project, дозволяє детально налаштовувати робочий простір під потреби конкретного проєкту або команди.

Варто зауважити, що JIRA пропонує безкоштовний тарифний план для невеликих команд до 10 користувачів. У цьому безкоштовному плані команди мають доступ до основних функцій JIRA, включаючи управління завданнями та проєктами, а також можливість використовувати Scrum і Kanban дошки. Однак, деякі розширені можливості та інтеграції доступні лише в платних планах. Це робить JIRA доступною опцією навіть для стартапів та малого бізнесу, які шукають потужне рішення для управління проєктами без значних витрат [14].

Таблиця 2.1 – Порівняльна таблиця між JIRA, Trello, Asana та Microsoft Project

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Назва | Підтримка | Тарифи | Переваги | Недоліки | Мобільні додатки | Співпраця з зовнішніми користувачами | Підтримка багатомовності | Аналітика | Кастомізація | Інтеграції |
| JIRA | Scrum, Kanban, інші | Від $7.50/користувач/місяць\* | Гнучкість, розширена кастомізація, інтеграція з іншими інструментами | Може бути складним для новачків, висока вартість для великих команд | Є | Обмежена | Є | Є | Висока | Широкі інтеграції з Atlassian продуктами та іншими сервісами |
| Trello | Kanban | Безкоштовний, преміум від $9.99/користувач/місяць | Інтуїтивний інтерфейс, простота використання, візуалізація проєктів | Обмежені можливості для складних проєктів, відсутність деяких функцій управління | Є | Є | Обмежена | Ні | Обмежена | Інтеграції через Power-Ups та API |
| Asana | Agile, Scrum (адаптовано) | Безкоштовний, преміум від $10.99/користувач/місяць | Гнучке управління завданнями, інтуїтивний дизайн, співпраця команд | Може бути недостатньо функціональним для дуже великих проєктів | Є | Є | Є | Є | Середня | Інтеграції з різними сервісами через API та додатки |
| Microsoft Project | Традиційне управління проєктами, Agile (з застосунками) | Від $10 користувач/місяць (для хмарної версії) | Потужні інструменти планування, детальне управління ресурсами | Висока вартість, складність в освоєнні для новачків | Є | Обмежена | Є | Є | Висока | Інтеграції з продуктами Microsoft та іншими сервісами |

2.4. Висновки до розділу 2

У другому розділі було проведено детальний огляд деяких сучасних програмних реалізацій у сфері систем управління задачами для IT-проєктів. Розглянуто чотири основні інструменти: JIRA, Trello, Asana та Microsoft Project. Кожен із цих інструментів має свої особливості, переваги та недоліки, які впливають на їхню придатність для різних типів проєктів та команд.

Основні висновки розділу:

1. JIRA від Atlassian є потужним та гнучким інструментом, який підтримує різні методології управління проєктами, зокрема Scrum та Kanban. Вона ідеально підходить для великих команд та складних проєктів, де потрібна розширена кастомізація та інтеграція з іншими сервісами. Однак, для новачків JIRA може бути складною в освоєнні, а вартість її використання для великих команд може бути високою.
2. Trello відзначається простотою та інтуїтивністю інтерфейсу, що робить його привабливим для невеликих команд та індивідуальних користувачів. Базуючись на принципах Kanban, Trello забезпечує візуалізацію робочого процесу та легку співпрацю. Проте, його функціональність може бути обмеженою для складних проєктів, а відсутність розширених інструментів управління може стати перешкодою для великих команд.
3. Asana пропонує баланс між гнучкістю та потужністю. Підтримуючи Agile та Scrum методології, вона підходить для команд різного розміру. Asana забезпечує ефективне управління завданнями, співпрацю та інтеграцію з іншими сервісами. Проте, для дуже великих або складних проєктів її функціональність може бути недостатньою.
4. Microsoft Project є традиційним інструментом для детального планування та управління ресурсами, особливо ефективним при використанні методології Waterfall. Він пропонує потужні інструменти планування та аналітики, але може бути складним в освоєнні для новачків. Висока вартість та складність використання роблять його більш придатним для великих організацій та професійних менеджерів проєктів.

Вибір системи управління залежить від специфіки проєкту, розміру та досвіду команди, використовуваної методології та фінансових можливостей організації. Жоден із розглянутих інструментів не є універсальним рішенням для всіх ситуацій. Командам необхідно ретельно аналізувати свої потреби та обирати інструмент, який найкраще відповідає їхнім вимогам та сприятиме ефективному управлінню проєктами.

РОЗДІЛ 3

ФОРМУВАННЯ ВИМОГ ТА АРХІТЕКТУРИ ДО СИСТЕМИ УПРАВЛІННЯ ЗАДАЧАМИ

3.1. Формування основних вимог до системи та обґрунтування вибору підходу Kanban

У процесі розробки системи управління задачами було проведено глибокий аналіз функціональних та нефункціональних вимог, який базується на багаторічному особистому досвіді управління різноманітними ІТ-проєктами. Серед цих проєктів — розробка систем відеоспостереження, клієнтських карткових програм та сервісів, інтернет-магазинів, мобільних застосунків, масштабування фінансових продуктів та сервісів, автоматизованих банківських систем, систем автоматизації процесів діловодства, а також ERP та CRM систем. Цей практичний досвід дозволив не лише зрозуміти специфічні потреби та виклики, з якими стикаються команди під час реалізації проєктів, але й визначити найбільш ефективні підходи до їх вирішення.

Бачення оптимальної системи управління задачами сформувалося на основі безпосередніх спостережень за динамікою роботи команд, аналізу чинників, що впливають на продуктивність та ефективність процесів, а також врахування потреб у зручних інструментах для планування, координації та контролю виконання проєктних завдань. Врахування власного досвіду дозволило ідентифікувати ключові аспекти, які часто упускаються при стандартному підході до розробки систем управління задачами, такі як важливість гнучкості управління змінами в проєкті, необхідність інтуїтивно зрозумілого інтерфейсу для користувачів різного рівня, та значення ефективної комунікації всередині команди. Враховуючи це, функціональні вимоги були сформульовані з опорою на власний досвід та узагальнені відповідно до загальноприйнятих підходів у галузі розробки систем управління проєктами [4, 7, 8].

Функціональні вимоги до системи включають:

1. реєстрація та авторизація користувачів: система повинна забезпечувати можливість реєстрації нових користувачів та авторизації існуючих. Це необхідно для контролю доступу до функціоналу системи та забезпечення безпеки даних.
2. управління користувачами:

* адміністрування користувачів: повинна бути реалізована можливість для адміністраторів керувати користувачами системи, включаючи перегляд списку користувачів, їх деталей;
* зміна ролей: система повинна дозволяти змінювати ролі користувачів (наприклад, призначати адміністратора, менеджера проєкту або звичайного користувача), що впливає на їхні права доступу до різних функцій системи;
* блокування та розблокування користувачів: адміністратори повинні мати можливість блокувати або розблоковувати облікові записи користувачів у разі необхідності.

1. створення та управління проєктами:

* створення проєктів: користувачі з відповідними правами повинні мати можливість створювати нові проєкти, задавати їхні параметри та опис;
* редагування та видалення проєктів: повинна бути можливість змінювати деталі проєкту та видаляти його при потребі.

1. створення та управління задачами:

* користувачі повинні мати можливість створювати, редагувати та видаляти задачі, а також змінювати стадії виконання;
* можливість призначати відповідального за виконання конкретних задач.

1. візуалізація процесів:

* система повинна надавати візуальні дошки для кожного проєкту, що відображають поточний стан задач та їхній прогрес;
* стан задач: відображення статусів задач (наприклад, "Нові", "В роботі", "Завершені") для швидкого розуміння стану проєкту.

1. контроль термінів виконання задач:

* дата завершення (Due Date): Можливість встановлення термінів виконання для кожної задачі;
* візуальні індикатори термінів: автоматична зміна кольору задачі при наближенні або перевищенні терміну виконання, що допомагає користувачам швидко ідентифікувати критичні задачі.

Детально розглянемо характеристики, на яких базуються функціональні вимоги.

1. Інтуїтивність інтерфейсу:

* зручність використання: інтерфейс системи повинен бути простим та зрозумілим, щоб користувачі з різним рівнем технічної підготовки могли легко освоїти систему;
* консистентність дизайну: використання єдиних принципів дизайну та навігації по всій системі.

1. Надійність та безпека:

* стабільна робота: система повинна працювати без збоїв та забезпечувати швидкий доступ до даних;
* захист даних: використання безпечних методів зберігання та передачі даних, шифрування паролів, захист від загроз, таких як SQL-ін'єкції та Cross-Site Scripting (XSS);
* контроль доступу: реалізація механізмів аутентифікації та авторизації для захисту конфіденційних даних.

1. Масштабованість та гнучкість:

* можливість розширення: архітектура системи повинна дозволяти додавати новий функціонал та модулі без значних змін в існуючому коді;
* рознесення фронтенду та бекенду: планування можливості в майбутньому розділити фронтенд та бекенд на окремі сервіси, що покращить масштабованість та гнучкість системи;
* підготовка до навантаження: система повинна бути здатна обробляти збільшену кількість користувачів та даних без втрати продуктивності;
* використання стандартів: дотримання загальноприйнятих стандартів та найкращих практик розробки програмного забезпечення.

1. Відповідність сучасним технологіям:

* актуальність технологій: використання сучасних фреймворків та бібліотек, які підтримуються спільнотою та мають регулярні оновлення;
* кросбраузерність: система повинна коректно працювати в основних веббраузерах.

Перед прийняттям рішення про використання Kanban як основи для системи управління задачами, було проведено ретельний аналіз різних підходів до управління проєктами, зокрема Scrum, Kanban та Extreme Programming (XP). Метою аналізу було визначити, яка з цих методологій найкраще відповідає потребам проєкту та команди, з урахуванням таких факторів, як гнучкість, простота імплементації, вартість розробки та підтримки, а також ефективність управління поточними завданнями.

Scrum вимагав би впровадження чіткої структури спринтів, визначених ролей у команді та регулярних зустрічей для планування та ретроспективи. Цей підхід, хоча і ефективний для деяких команд та проєктів, вважався надмірно складним для поточних потреб, особливо з огляду на відсутність необхідності в інтенсивному масштабуванні або складних інтеграціях.

Extreme Programming (XP), зосереджений на технічній дисципліні та високій якості коду, вимагає високого рівня дисципліни та досвіду, а також включає практики, які можуть бути надмірними для проєкту, зосередженого на управлінні задачами.

У результаті аналізу було обрано Kanban через його простоту, візуальну відкритість та адаптивність до різних видів проєктів і робочих процесів. Kanban ідеально підходить для створення гнучкої та ефективної системи управління задачами, яка не потребує складних налаштувань або детального планування спринтів, що робить його оптимальним вибором для поточних потреб.

Переваги вибору Kanban:

* постота та зрозумілість: візуальне відображення процесів сприяє швидкому розумінню стану проєкту та полегшує комунікацію в команді;
* адаптивність: Kanban дозволяє легко адаптуватися до змін у проєкті без необхідності складного перепланування, що важливо в динамічному середовищі;
* мінімізація перевантаження: обмеження роботи в процесі допомагає уникнути перевантаження команди завданнями та покращує управління ресурсами.

Таким чином, обравши Kanban як основу для системи управління задачами, було створено інструмент, який відповідає всім вимогам проєкту, забезпечуючи простоту, гнучкість та ефективність управління проєктами та задачами.

3.2. Розробка концептуальної моделі системи

У процесі створення концептуальної моделі майбутньої системи управління задачами, яка базується на принципах Kanban, основна увага була зосереджена на визначенні ключових компонентів, що забезпечать ефективну взаємодію та функціональність системи відповідно до обраної методології. Головною метою було розробити систему, яка не лише відповідатиме сучасним вимогам управління проєктами, але й буде інтуїтивно зрозумілою та зручною для кінцевих користувачів.

Розглянемо основні компоненти системи.

1. Фронтенд (інтерфейс користувача). Основні вимоги до інтерфейсу включають:

* інтерфейс користувача є критичним компонентом системи, оскільки саме через нього користувачі взаємодіють із системою;
* інтуїтивність та зручність використання: Інтерфейс повинен бути зрозумілим для користувачів з різним рівнем технічної підготовки. Це передбачає логічну організацію елементів управління, використання зрозумілих символів та підказок;
* візуалізація Kanban-дошки: Особливу увагу приділено реалізації Kanban-дошки, яка дозволяє користувачам візуально відстежувати прогрес завдань, переміщувати задачі між стадіями виконання та швидко оцінювати стан проєкту;
* адаптивний дизайн: Інтерфейс повинен коректно відображатися на різних пристроях та роздільних здатностях екрана, забезпечуючи зручність використання як на стаціонарних комп'ютерах, так і на мобільних пристроях.

1. Бекенд (серверна частина системи). Бекенд відповідає за обробку запитів від клієнтської частини, управління бізнес-логікою та взаємодію з базою даних. Основні функції бекенду включають:

* обробку даних: реалізація CRUD-операцій (створення, читання, оновлення, видалення) для всіх основних об'єктів системи, таких як користувачі, проєкти, задачі;
* аутентифікацію та авторизацію: забезпечення безпечного доступу до системи, управління правами користувачів та ролями;
* API для взаємодії з фронтендом: надання інтерфейсів для обміну даними з клієнтською частиною, використовуючи стандартизовані протоколи (наприклад, RESTful API або GraphQL).

1. База даних. Вимоги до бази даних:

* база даних є центральним сховищем інформації про користувачів, проєкти, задачі та їхні статуси;
* продуктивність та надійність: швидкий доступ до даних та стабільна робота при високих навантаженнях;
* цілісність даних: забезпечення узгодженості та цілісності збережених даних;
* безпека: захист даних від несанкціонованого доступу та втрати.

Розглянемо взаємодію компонентів.

* Фронтенд ↔ Бекенд: клієнтська частина відправляє запити до серверної частини через API, отримуючи необхідні дані для відображення та надсилаючи зміни, зроблені користувачами.
* Бекенд ↔ База даних: серверна частина взаємодіє з базою даних для зберігання та отримання інформації, використовуючи відповідні драйвери та ORM (Object-Relational Mapping) технології.
* Реактивність та оновлення в реальному часі: для забезпечення актуальності даних на клієнтській частині можуть використовуватися технології вебсокетів або механізми періодичного оновлення даних.

Ця структура дозволяє створити гнучку та ефективну систему управління задачами, яка не тільки відповідає основним принципам Kanban, але й забезпечує високу адаптивність та зручність використання для кінцевих користувачів.

* 1. Вибір технологічного стеку

При розробці системи управління задачами, яка втілює принципи методології Kanban, було важливо обрати технологічний стек, що забезпечить високу продуктивність, надійність та гнучкість для адаптації до змінних потреб користувачів і проєктів.

Фронтенд: Vue.js. Для фронтенду було обрано фреймворк Vue.js завдяки його компонентній архітектурі, яка дозволяє розділяти інтерфейс користувача на незалежні та повторно використовувані компоненти [15]. Це сприяє чистоті коду, спрощує процес розробки та полегшує підтримку і масштабування застосунку. Реактивність Vue.js забезпечує миттєве відображення змін даних у користувацькому інтерфейсі без необхідності перезавантаження сторінки, що є критично важливим для систем, де стан задач і проєктів динамічно змінюється. Гнучкість та легкість інтеграції з іншими інструментами і бібліотеками дозволяють розширювати функціональність системи та інтегрувати її з різними бекенд-сервісами. Швидкість розробки і висока продуктивність Vue.js сприяють прискоренню процесу створення застосунку без втрати якості та ефективності.

Бекенд: Strapi. Для бекенду було обрано Strapi – headless CMS (система управління контентом без прив'язаного фронтенду), яка відзначається універсальністю, продуктивністю та можливістю глибокої кастомізації [16]. Strapi значно прискорює процес розробки завдяки автоматизації створення RESTful API та інтуїтивно зрозумілому інтерфейсу для управління контентом і структурою даних. Потужні засоби моделювання даних дозволяють легко створювати та налаштовувати моделі даних відповідно до потреб проєкту, забезпечуючи гнучкість і масштабованість системи. Розширені можливості управління доступом та аутентифікації забезпечують високий рівень безпеки, дозволяючи гнучко керувати правами користувачів і захищати конфіденційну інформацію. Headless архітектура Strapi відкриває широкі можливості для інтеграції з різноманітними фронтенд-платформами та іншими сервісами, що сприяє подальшому розвитку та масштабуванню проєкту.

* 1. Висновки до розділу 3

У третьому розділі було проведено детальний аналіз вимог до системи управління задачами, який базувався на багаторічному особистому досвіді управління різноманітними ІТ-проєктами. Це дозволило чітко визначити функціональні та нефункціональні вимоги, які є критичними для успішної роботи системи та задоволення потреб користувачів.

Функціональні вимоги охоплюють такі аспекти, як реєстрація та авторизація користувачів, управління користувачами та проєктами, створення та управління задачами, візуалізація процесів за допомогою Kanban-дошок та контроль термінів виконання задач. Нефункціональні вимоги зосереджені на інтуїтивності інтерфейсу, надійності та безпеці, масштабованості та гнучкості системи, а також відповідності сучасним технологіям.

Проведений аналіз різних підходів до управління проєктами, зокрема Scrum, Kanban та Extreme Programming (XP), дозволив обґрунтувати вибір методології Kanban як основи для розробки системи. Kanban відзначається простотою, візуальною відкритістю та адаптивністю до різних видів проєктів і робочих процесів, що робить його оптимальним вибором для поточних потреб. Переваги Kanban, такі як простота та зрозумілість, адаптивність до змін та мінімізація перевантаження команди, сприяють ефективному управлінню проєктами та задачами.

Було розроблено концептуальну модель системи, яка складається з трьох основних компонентів: фронтенду, бекенду та бази даних. Фронтенд забезпечує інтуїтивний та зручний інтерфейс користувача з акцентом на візуалізацію Kanban-дошки, що дозволяє ефективно відстежувати прогрес завдань. Бекенд відповідає за обробку запитів, управління бізнес-логікою та забезпечує безпеку через аутентифікацію та авторизацію користувачів. База даних слугує центральним сховищем інформації про користувачів, проєкти та задачі, забезпечуючи продуктивність, надійність та безпеку даних.

Взаємодія між компонентами системи організована таким чином, щоб забезпечити ефективність та актуальність даних. Фронтенд і бекенд обмінюються інформацією через API, що дозволяє оперативно відображати зміни та реагувати на дії користувачів. Така архітектура забезпечує гнучкість та ефективність системи, дозволяючи легко адаптувати її до змінних потреб користувачів та проєктів.

Також було обрано технологічний стек, що включає Vue.js для фронтенду та Strapi для бекенду.

Vue.js був обраний завдяки своїй компонентній архітектурі, реактивності та гнучкості. Це забезпечує створення інтуїтивного та зручного інтерфейсу користувача з акцентом на візуалізацію Kanban-дошки. Реактивність Vue.js дозволяє миттєво відображати зміни даних, що покращує користувацький досвід та підвищує ефективність роботи з системою.

Strapi, як headless CMS, був обраний для бекенду через свою універсальність, продуктивність та можливість гнучкого моделювання даних. Він забезпечує швидку розробку за рахунок автоматизації створення RESTful API, а також надає розширені можливості з управління безпекою, включаючи аутентифікацію та авторизацію користувачів.

Таким чином, проведений у розділі аналіз підтвердив, що використання методології Kanban є оптимальним рішенням для розробки системи управління задачами, яка відповідає визначеним вимогам та потребам користувачів. Розроблена концептуальна модель системи закладає міцну основу для її реалізації, забезпечуючи гнучкість, масштабованість та зручність використання. Це сприятиме підвищенню ефективності управління проєктами та покращенню командної роботи, відповідаючи сучасним стандартам розробки програмного забезпечення та сприяючи успішному впровадженню системи в практичну діяльність.

РОЗДІЛ 4

ПРОЦЕС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ СИСТЕМИ ТА ЇЇ ІНТЕГРАЦІЯ З ОБРАНИМИ ТЕХНОЛОГІЯМИ

4.1. Вибір операційної системи та налаштування середовища розробки

У процесі реалізації системи управління задачами було прийнято рішення використовувати операційну систему Debian Linux версії 6.1.0-21-amd64. Цей вибір обумовлений кількома ключовими факторами:

* стабільність та надійність: операційна система Debian Linux відома своєю стабільністю та довготривалою підтримкою, що є критично важливим для розробки та розгортання вебзастосунків;
* відкритий код та спільнота: відкрита природа системи та активна спільнота розробників забезпечують широкий доступ до ресурсів, документації та підтримки;
* гнучкість та налаштовуваність: Debian Linux дозволяє тонко налаштовувати систему під специфічні потреби проєкту, включаючи вибір необхідних пакетів та сервісів.

Використання консольної версії Debian без графічного інтерфейсу було додатковим рішенням, спрямованим на оптимізацію ресурсів сервера. Це дозволяє:

* зизити споживання системних ресурсів: відсутність графічного інтерфейсу зменшує навантаження на оперативну пам'ять та процесор;
* підвищити безпеку та стабільність: менша кількість встановленого програмного забезпечення зменшує потенційні вразливості та необхідність регулярних оновлень;
* сфокусуватися на серверних завданнях: консольна версія ідеально підходить для серверів, де основними завданнями є робота з базами даних, вебсерверами та бекенд-застосунками.

Для доступу до сервера використовується консоль OpenSSH, яка забезпечує безпечне з'єднання через протокол SSH. Вона дозволяє ефективно працювати з віддаленим сервером, виконувати адміністрування системи, встановлювати пакети, оновлювати конфігурації та контролювати стан серверного середовища.

На рисунку 4.1 представлено поточний вигляд консолі після входу в систему, де зазначено версію ОС та дані останнього підключення.

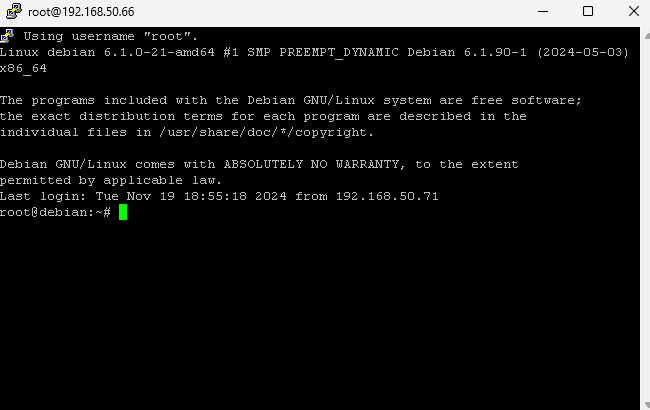


Рисунок 4.1 – Консольний інтерфейс Debian Linux

4.2. Встановлення Node.js та Yarn

Відповідно до документації Strapi станом на 10 листопада 2024 року, для запуску Strapi необхідно:

Node.js:

* версії: v18 або v20 (LTS).

Пакетний менеджер Node.js:

* npm (версія v6 і вище) або yarn. Я обрав yarn

Підтримувана база даних:

* MySQL: мінімум 5.7.8, рекомендовано 8.0.
* MariaDB: мінімум 10.3, рекомендовано 10.6.
* PostgreSQL: мінімум 11.0, рекомендовано 14.0.
* SQLite: Версія 3.

Встановлення NVM (Node Version Manager), який дозволяє керувати різними версіями Node.js на одній системі, що можливо знадобиться, для вибору різних версій при оновленні або оптимізації:

curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.1/install.sh | bash

source ~/.bashrc

Встановлення Node.js через NVM:

nvm install 18.19.0

nvm use 18.19.0

Встановлення Yarn:

curl -o- -L https://yarnpkg.com/install.sh | bash

source ~/.bashrc

Перевірка встановлених версій:

node -v *# Повинно вивести v18.19.0*

yarn -v *# Повинно вивести 1.22.22*

* 1. Встановлення Strapi

Розглянувши доступні опції, було вирішено використовувати PostgreSQL як основну базу даних для проєкту. PostgreSQL відома своєю надійністю та доведеною стабільністю у промислових застосуваннях. Встановлення PostgreSQL [17] як бази даних для проєкту Strapi виконується наступною командою:

apt-get install postgresql

Налаштування бази даних і доступу, що відбувається з використанням користувача postgres виконується наступними командами:

createdb my\_strapi\_db;

psql

CREATE USER my\_strapi\_user WITH PASSWORD 'secure\_password';

GRANT ALL PRIVILEGES ON DATABASE my\_strapi\_db TO my\_strapi\_user;

\q

Слід зазначити, що у рамках проєкту використовується PostgreSQL версії 15.6, що забезпечує покращену продуктивність та безпеку.

Використовуючи Yarn, створюємо новий проєкт з кастомними налаштуваннями, та під час налаштування обираємо базу даних PostgreSQL:

yarn create strapi-app my-project –manual

У процесі створення проєкту вводимо параметри підключення до бази даних:

Host: localhost

Port: 5432

Database name: my\_strapi\_db

Username: my\_strapi\_user

Password: secure\_password

Enable SSL connection: No

Переходимо до каталогу проєкту та запускаємо Strapi в режимі розробки, успішність чого відображено на рисунку 4.2:

cd my-project

yarn develop

Основні команди для управління проєктом:

* yarn develop: запуск у режимі розробки з автоматичним перезавантаженням;
* yarn start: запуск у продакшн-режимі;
* yarn build: компіляція адміністративної панелі для продакшну;
* yarn strapi: виведення списку доступних команд Strapi.

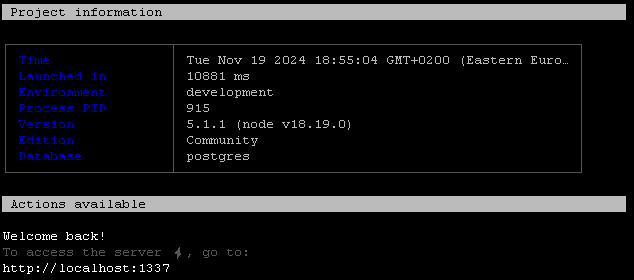


Рисунок 4.2 – Відображення доступності застосунку Strapi після запуску

* 1. Налаштування Strapi та створення колекцій
     1. Перший вхід у Strapi

При першому вході в адміністративну панель Strapi необхідно створити адміністративного користувача, який буде використовуватися для входу в систему та керування контентом. Інтерфейс налаштування представлено на рисунку 4.3.

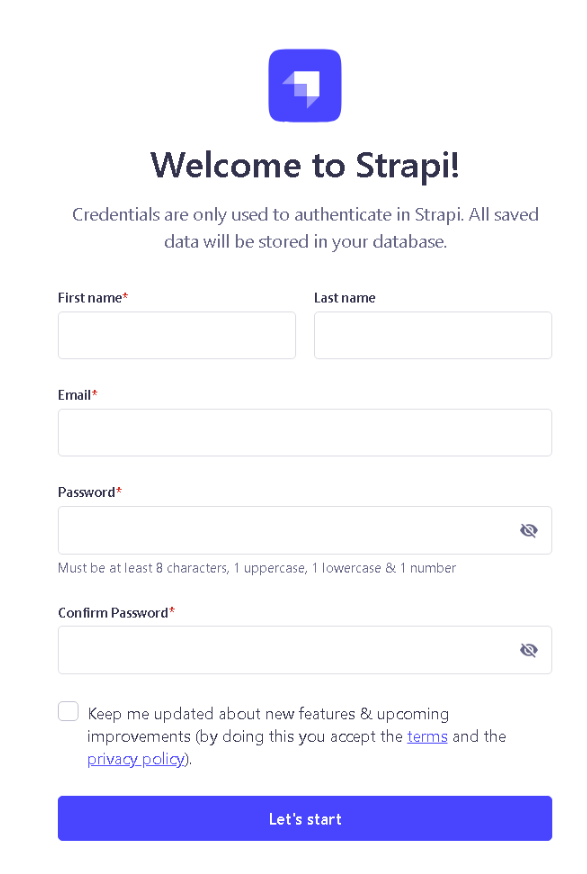


Рисунок 4.3 – Відображення доступності застосунку Strapi після запуску

Для створення користувача, необхідно ввести наступні поля:

* First name – поле для введення імені адміністратора;
* Last name – поле для введення прізвища адміністратора;
* Email – поле для введення електронної пошти адміністратора. Це має бути унікальне значення
* Password – поле для введення пароля. Пароль повинен містити щонайменше 8 символів:
  + одну велику літеру;
  + одну маленьку літеру;
  + одну цифру;
* Confirm Password – поле для підтвердження пароля шляхом повторного введення.

Після введення цих даних потрібно натиснути кнопку "Let’s start", щоб завершити створення адміністративного користувача.

* + 1. Налаштування колекцій

На сторінці Content-Type Builder представлено можливості для створення та налаштування типів контенту. Створення першої колекції розпочинається з вибору опції "Collection Type" та подальшого додавання необхідних полів. Це відображено на рисунку 4.4.

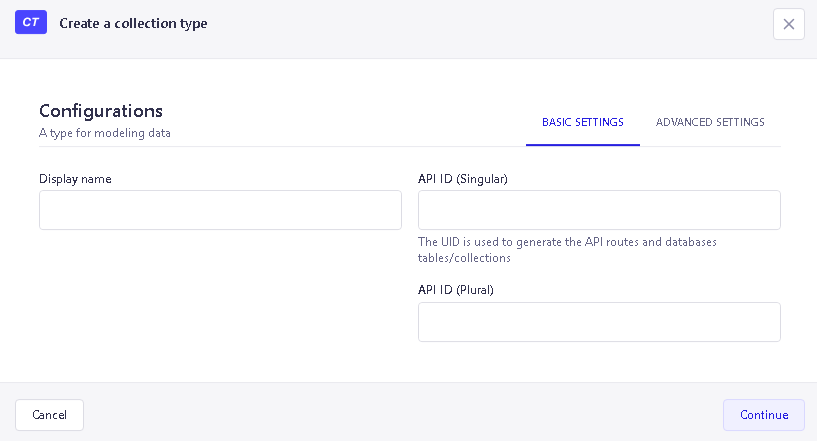


Рисунок 4.4 – Інтерфейс створення типу контенту

* + - 1. Колекція Project

Колекція Project використовується для управління проєктами. Основні параметри налаштувань (див. рисунок 4.5):

* Display name: Project — назва, яка буде відображатися в панелі адміністратора;
* API ID (Singular): project — використовується для роботи з одиничними записами через API;
* API ID (Plural): projects — використовується для роботи з множинними записами через API;
* Тип: Collection Type — вибрано для створення багаторазових екземплярів, таких як статті, товари чи коментарі.

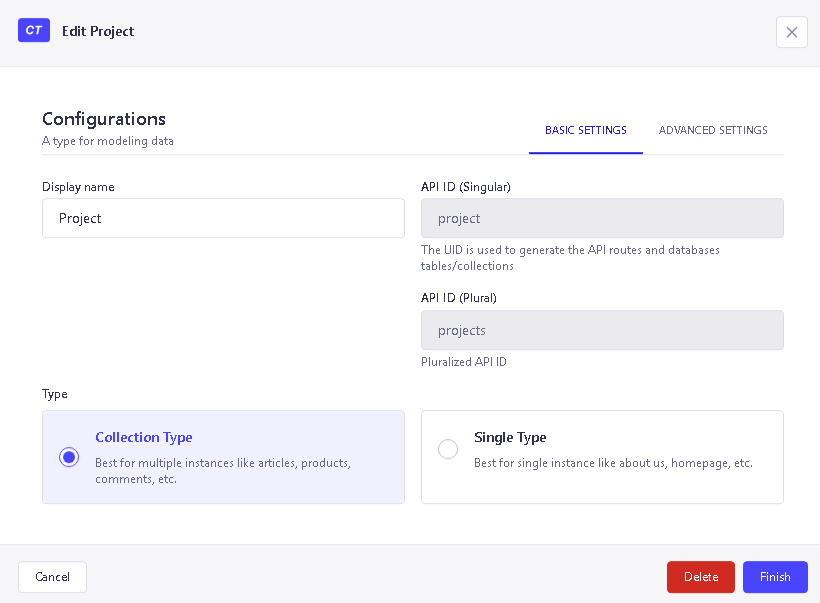


Рисунок 4.5 – Налаштування колекції Project

Поля колекції Project (див. рисунок 4.6):

* title: текстове поле для введення назви проєкту;
* description: текстове поле для опису проєкту.

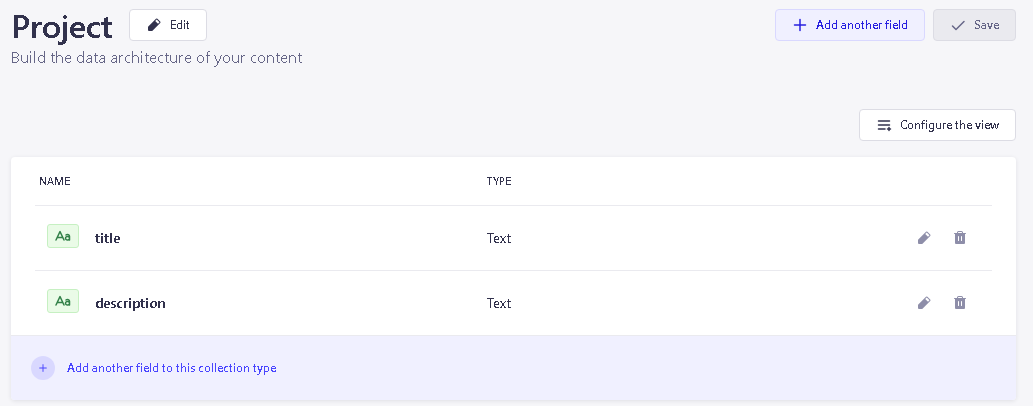


Рисунок 4.6 – Налаштування колекції Project

* + - 1. Колекція Board

Колекція Board використовується для створення та управління дошками у системі Kanban (див. рисунок 4.7). Основні поля:

* title: текстове поле для назви дошки;
* description: текстове поле для опису дошки;
* color: поле "Enumeration" для вибору кольору дошки;
* is\_archive: булеве поле для визначення, чи дошка архівована;
* project: відношення з колекцією Project (див. рисунок 4.8).

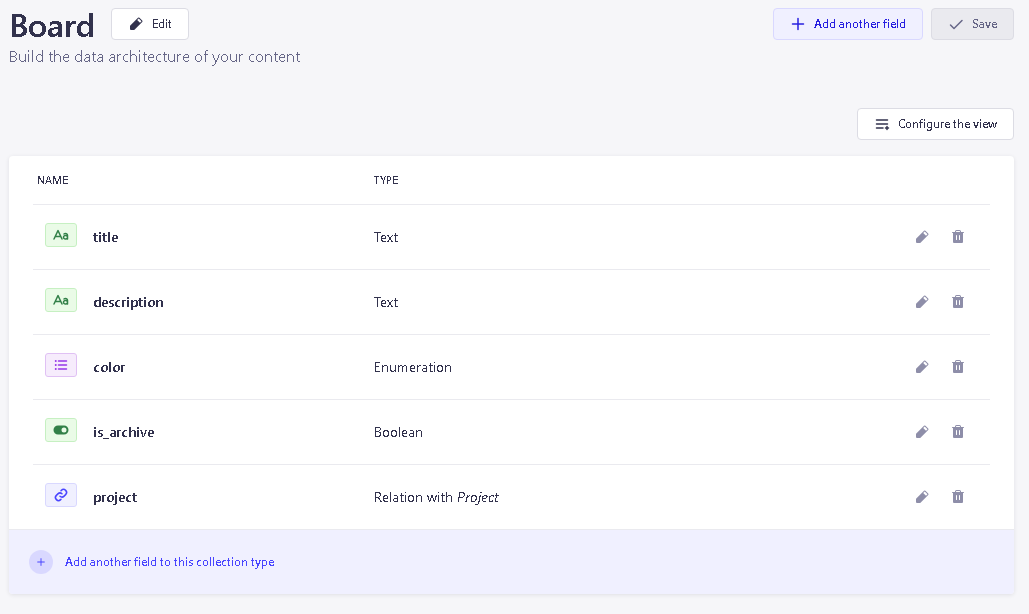


Рисунок 4.7 – Налаштування колекції Board

Strapi надає потужний механізм для створення зв’язків між колекціями. Відношення використовуються для моделювання зв’язків між різними типами даних у системі. Це дозволяє ефективно зберігати та маніпулювати взаємопов’язаними записами в базі даних. Типи відношень у Strapi:

* один до одного (One-to-One): це відношення використовується, коли один запис однієї колекції відповідає одному запису іншої; наприклад: один користувач може мати один профіль;
* один до багатьох (One-to-Many): це відношення дозволяє одному запису бути пов’язаним із багатьма записами іншої колекції; наприклад: один проєкт може містити багато дошок (див. рисунок 4.8);
* багато до одного (Many-to-One): це обернене до "Один до багатьох". Багато записів однієї колекції можуть бути прив’язані до одного запису іншої; наприклад: багато дошок можуть бути прив’язані до одного проєкту;
* багато до багатьох (Many-to-Many): це відношення дозволяє багатьом записам однієї колекції бути пов’язаними з багатьма записами іншої; наприклад: користувачі можуть бути додані до багатьох груп і кожна група може мати багато користувачів;
* поліморфні відношення (Polymorphic): це відношення дозволяє одному запису бути прив’язаним до записів різних типів колекцій; наприклад: зображення можуть бути прив’язані до користувачів, публікацій або коментарів.

Переваги використання зв’язків між колекціями:

* автоматичне формування маршрутів API: для кожного відношення Strapi автоматично створює маршрути API, що полегшує запити;
* зв’язки через адмін-панель: усі зв’язки між записами можна редагувати через інтерфейс адміністратора;
* динамічність взаємодії з базою даних: відношення автоматично реалізуються на рівні бази даних, спрощуючи роботу з таблицями.

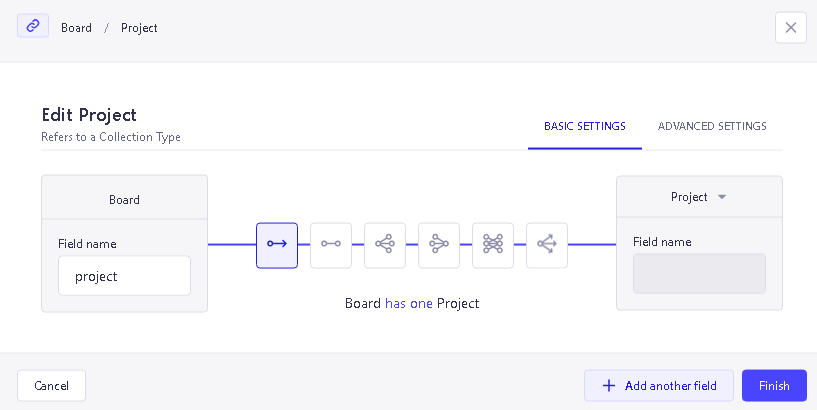


Рисунок 4.8 – Налаштування відношення з колекцією Project

* + - 1. Колекція Column

Колекція Column створена для управління колонками на дошках:

* name: Текстове поле для назви колонки;
* board: Відношення з колекцією Board;
* color: Поле "Enumeration" для вибору кольору;
* position: Числове поле для визначення порядку колонок.
  + - 1. Колекція Task

Колекція Task використовується для збереження даних про задачі:

* title: Назва задачі (текстове поле);
* column: Відношення з колекцією Column;
* assignedTo: Відношення з User для призначення виконавця;
* dueDate: Поле типу "Datetime" для дати виконання задачі;
* position: Числове поле для позиції задачі в колонці;
* description: Текстове поле для опису задачі;
  + - 1. Колекція Comment

Колекція Comment призначена для збереження коментарів:

* text: Текстове поле для змісту коментаря;
* task: Відношення з колекцією Task;
* user: Відношення з колекцією User (з plugin Users & Permissions).
  + - 1. Колекція User

Колекція User є системною та використовується для управління обліковими записами користувачів. Ця колекція є частиною плагіна Users & Permissions, і її конфігурацію не можна змінювати безпосередньо через адмін-панель.

Поля колекції призначені для забезпечення базової функціональності автентифікації, авторизації та управління ролями. Перелік полів для даної колекції відображено на рисунку 4.9.

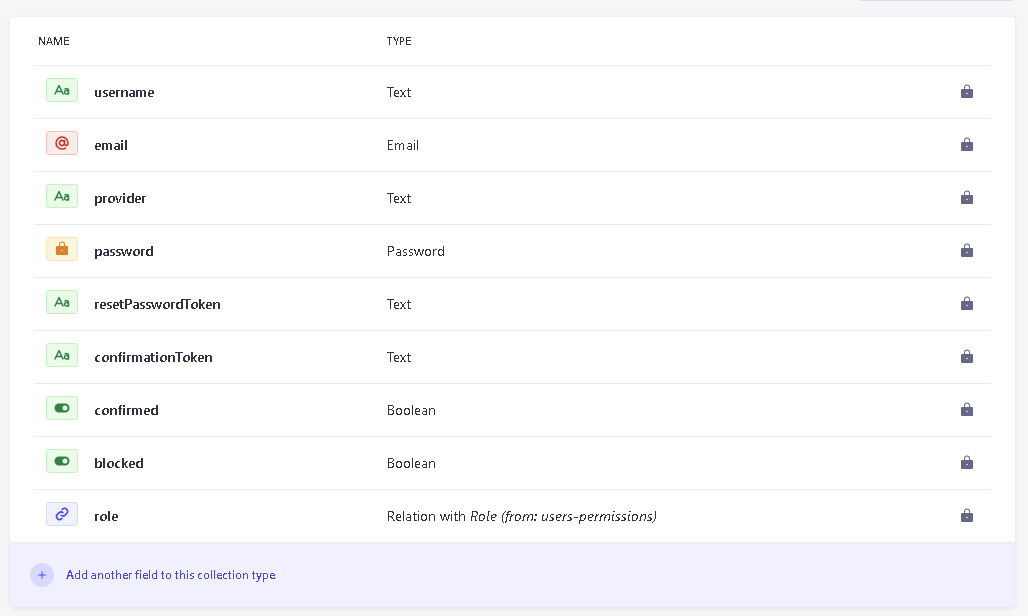


Рисунок 4.9 – Налаштування колекції User

* + 1. Налаштування ролей

Налаштування ролей користувачів є важливим аспектом забезпечення безпеки та ефективності роботи системи управління задачами. У системі Strapi сторінка Roles (див. рисунок 4.10) використовується для управління ролями користувачів, надаючи можливість створювати нові ролі, редагувати існуючі, налаштовувати їхні дозволи та визначати, які дії можуть виконувати користувачі з певною роллю.

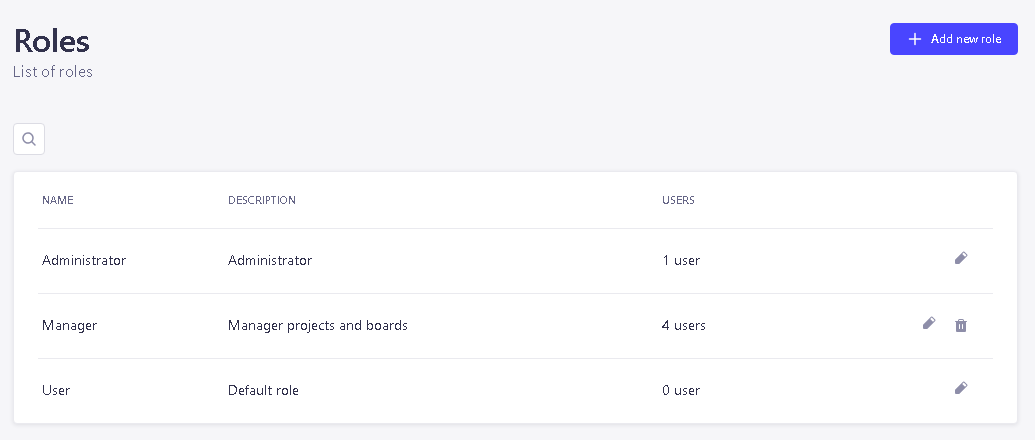


Рисунок 4.10 – Адміністрування ролей у Strapi

У процесі налаштування було реалізовано гнучкий механізм визначення прав доступу для кожної ролі. Кожна роль характеризується власною назвою, описом та кількістю користувачів, які до неї належать. Для створення нової ролі передбачено використання інтуїтивно зрозумілого інтерфейсу, який дозволяє вводити необхідні параметри та детально налаштовувати дозволи, що надаються користувачам.

Роль Administrator була налаштована з повним доступом до всіх функцій системи. Це включає можливість створення, перегляду, оновлення та видалення записів для всіх типів контенту, таких як проєкти, дошки та задачі. У налаштуваннях дозволів для цієї ролі були активовані всі доступні дії, що забезпечує адміністраторам максимальні права для управління системою та її конфігурацією.

Для ролі User доступ був свідомо обмежений. Користувачі з цією роллю мають у певних випадках можливість лише переглядати записи без права на редагування або видалення (див. рисунок 4.11). Таке обмеження гарантує, що звичайні користувачі можуть ознайомлюватися з інформацією, необхідною для виконання їхніх завдань, але не можуть вносити зміни, що підвищує цілісність та безпеку даних у системі.

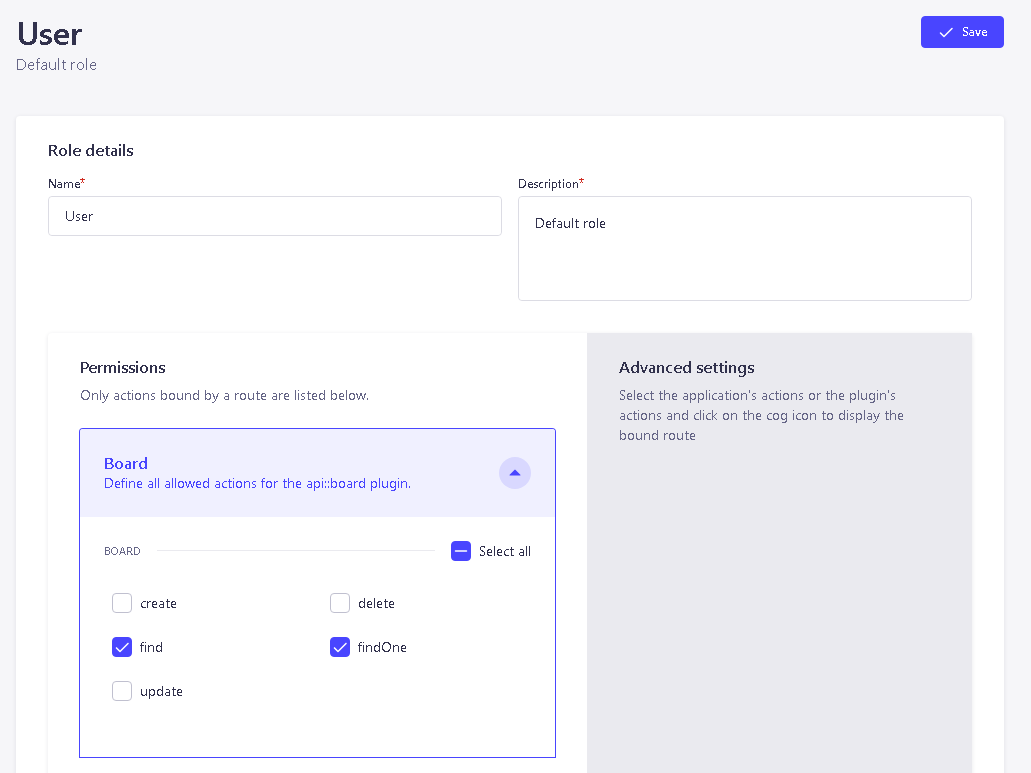


Рисунок 4.11 – Налаштування обмеження ролі User для колекції Board

Крім того, була створена роль Manager, яка надає користувачам можливість керувати певними частинами контенту, зокрема проєктами та дошками, без доступу до адміністративних функцій системи. Це дозволяє менеджерам проєктів ефективно контролювати процеси в межах їхньої компетенції, не порушуючи загальну безпеку системи.

* + 1. Генерація API-токену для тестування бекенду

Strapi надає додатковий механізм генерації API-токенів, що дозволяє забезпечити доступ до API з можливістю гнучкого налаштування прав і дій. У рамках проєкту було створено API-токен під назвою "root", який отримав повний доступ до всіх ресурсів API (див. рисунок 4.12). Це рішення було прийнято для проведення тестування бекенду системи, не використовуючи стандартну систему аутентифікації користувачів.

Під час генерації токена був обраний тип доступу "Custom", що надає розширений контроль над дозволами. Це дозволило точно визначити, які дії можуть виконуватися за допомогою даного токена, включаючи створення, оновлення, перегляд та видалення записів у різних колекціях даних. Такий підхід забезпечує високий рівень гнучкості та безпеки, оскільки дозволяє налаштувати доступ до ресурсів відповідно до конкретних потреб інтеграції.

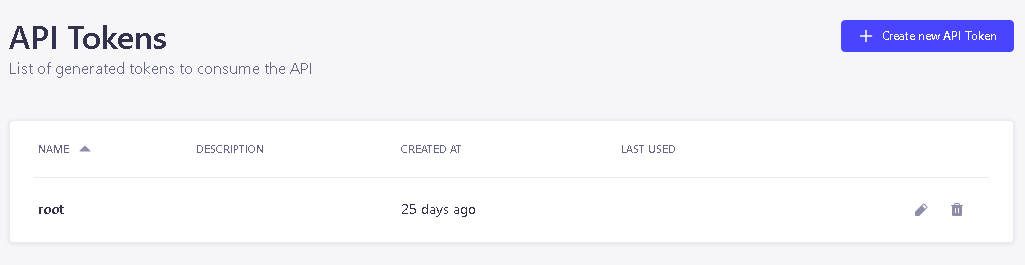


Рисунок 4.12 – Сторінка для генерації токенів

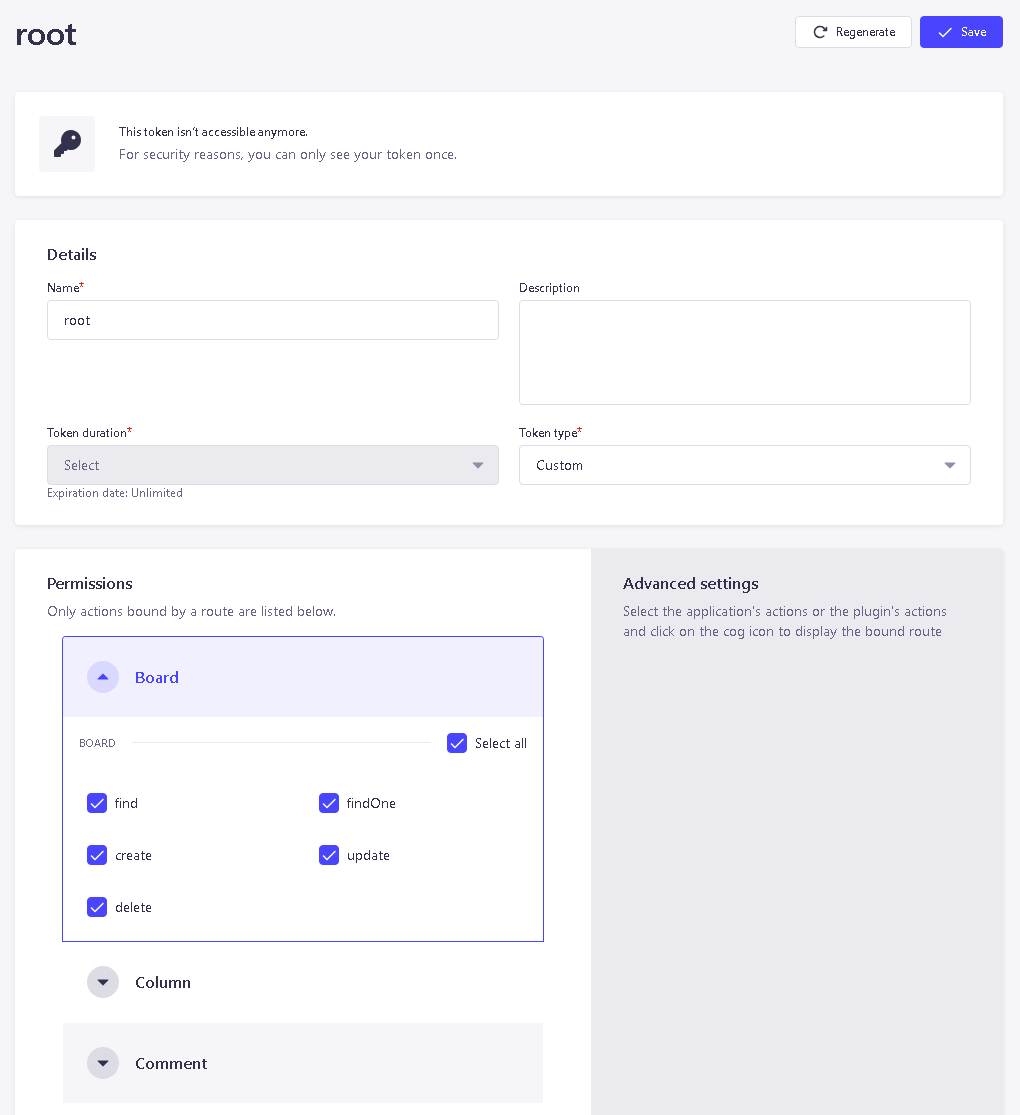


Рисунок 4.13 – Налаштування параметрів токену

Для токена був встановлений необмежений термін дії, що є оптимальним рішенням для тривалого тестування або для використання в автоматизованих процесах, де регулярне оновлення токена може бути незручним або недоцільним (див. рисунок 4.13). Однак у продакшн-середовищі рекомендується встановлювати обмежений термін дії токенів та регулярно їх оновлювати з метою підвищення безпеки.

Взагалом, використання API-токенів у Strapi значно спрощує процес взаємодії з бекендом для зовнішніх сервісів. Це особливо корисно при розробці мобільних застосунків, інтеграції з третіх сторін сервісами або при налаштуванні мікросервісної архітектури. Гнучкі налаштування дозволів забезпечують контроль над доступом до ресурсів, запобігаючи несанкціонованому використанню API та захищаючи конфіденційні дані.

* 1. Встановлення та налаштування основних фронтенд-компонентів
     1. Встановлення Vue.js через використання Vue CLI

Встановлення Vue CLI:

yarn global add @vue/cli

Перевірка встановленої версії:

vue --version # Повинно вивести @vue/cli 5.0.8

* + 1. Створення та налаштування проєкту Vue.js

Створення нового проєкту:

vue create my-frontend

Під час створення проєкту обираємо необхідні опції:

* додаємо підтримку TypeScript, Vue Router, Pinia для керування станом;
* вибираємо ESLint для забезпечення якості коду.

Переходимо до каталогу проєкту та встановлюємо залежності:

cd my-frontend

yarn install

* + 1. Налаштування Vue Router та запуск проєкту

Встановлення Vue Router:

yarn add vue-router

Запуск проєкту в режимі розробки:

yarn dev –host

Консоль виведе інформацію про доступність застосунку, що відображено на рисунку 4.14.

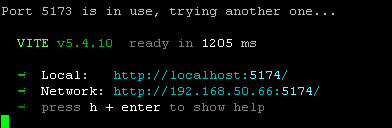


Рисунок 4.14 – Відображення доступності застосунку Vue CLI після запуску

* + 1. Інтеграція фронтенду з бекендом та осовне налаштування Vue

Для забезпечення взаємодії між клієнтською та серверною частинами використовується RESTful API, наданий Strapi.

У файлі .env для Vue вказуємо URL бекенду:

VITE\_API\_URL=http://192.168.50.66:1337/api

Встановлюємо бібліотеку Axios для HTTP-запитів:

yarn add axios

У папці src необхідно створити файл api.js для роботи з API, в якому створюємо інстанс Axios з базовим URL:

import axios from 'axios';

const api = axios.create({

baseURL: import.meta.env.VITE\_API\_URL,

});

export default api;

Повний код файлу api.js відображено у додатку А.

Файл main.js у Vue.js є критичним для запуску та налаштування вашого застосунку. Він слугує мостом між вашими Vue-компонентами та HTML-сторінкою, дозволяючи вам контролювати поведінку та структуру вашого застосунку з єдиного місця. Повний код файлу main.js відображено у додатку Б.

* 1. Встановлення та налаштування Swagger для документування API

Swagger є інструментом для автоматизації документування API, який значно спрощує та стандартизує процес створення документації для RESTful API. Він допомагає розробникам і користувачам зрозуміти, як працює API, та надає можливість взаємодіяти з ним безпосередньо через вебінтерфейс. Swagger є частиною проєкту OpenAPI, який визначає стандарти для опису API.

У рамках розробки системи управління задачами було вирішено інтегрувати Swagger для:

* + огляд доступних ендпоінтів та методів: Це спрощує розуміння структури та можливостей API;
  + тестування запитів без додаткового коду: можливість взаємодіяти з API безпосередньо через веб-інтерфейс Swagger UI дозволяє швидко перевіряти роботу та поведінку API;
  + автоматичне оновлення документації: при зміні API документація оновлюється автоматично, що знижує ризики невідповідностей та спрощує підтримку.
    1. Процедура встановлення Swagger для Strapi:

Для початку було виконано команду для встановлення плагіна strapi-plugin-documentation, який забезпечує підтримку Swagger у Strapi:

yarn add strapi-plugin-documentation

Для активації плагіна необхідно було налаштувати файл конфігурації../config/plugins.js з додаванням наступного коду:

module.exports = {

'documentation': {

config: {

servers: [

{

url: 'http://192.168.50.66:1337/api/',

description: 'API server'

},

{

url: 'http://localhost:1337/api/',

description: 'Local API server'

}

]

}

}

};

Це дозволяє Strapi активувати плагін документації при запуску та забезпечує доступ до Swagger UI за вказаною адресою.

Після успішного запуску Strapi з плагіном Swagger, документація API стала доступною за адресою:

http:// 192.168.50.66:1337/documentation

* + 1. Налаштування CORS (Cross-Origin Resource Sharing):

Для забезпечення безпечного доступу до API з різних доменів необхідно було налаштувати CORS. Це важливо, оскільки браузери за замовчуванням обмежують крос-доменні запити для безпеки.

Було внесено відповідні зміни до файлу конфігурації ./config/middlewares.js

У процесі налаштування CORS були виконані наступні кроки:

Визначення дозволених джерел запитів (Origin): Було налаштовано параметри, які визначають, з яких доменів дозволено здійснювати запити до API. Для цілей розробки та тестування було встановлено дозвіл на запити з будь-якого домену. Це досягається шляхом встановлення параметра origin на значення \*, що означає дозвіл всім доменам. Однак, у виробничому середовищі рекомендується обмежити цей параметр до конкретних доменів, з яких планується доступ до API, з метою підвищення рівня безпеки.

Налаштування дозволених методів HTTP: Було визначено, які методи HTTP можуть використовуватися при запитах до API. До дозволених були включені основні методи, такі як GET, POST, PUT, PATCH, DELETE та OPTIONS. Це забезпечує контроль над тим, які операції можуть виконуватися клієнтськими застосунками, і запобігає несанкціонованим діям.

Визначення дозволених заголовків (Headers): Було налаштовано список дозволених заголовків, які можуть бути використані в запитах до API. Включення необхідних заголовків, таких як Authorization та Content-Type, забезпечує правильну передачу аутентифікаційних даних та форматів контенту. Це важливо для роботи з токенами доступу та обробки різних форматів даних.

Налаштування політики безпеки контенту (Content Security Policy): Були внесені налаштування, які визначають, які ресурси можуть завантажуватися та використовуватися застосунком. Це включає дозволи на підключення до певних джерел даних, завантаження зображень та медіафайлів з визначених доменів. Таке налаштування допомагає запобігти потенційним атакам та забезпечує додатковий рівень безпеки.

* + 1. Перевірка роботи запитів через Swagger

Було проведено перевірку роботи API через інтегровану Swagger-документацію, яка автоматично генерується Strapi [18]. Для тестування використовувався токен, попередньо згенерований у адміністративному інтерфейсі, що забезпечило автентифікацію та доступ до захищених ендпоінтів. У процесі тестування було виконано запити на отримання списку записів, створення нових елементів, оновлення існуючих та їх видалення. Наприклад, для колекції Board були перевірені запити типу GET, POST, PUT та DELETE. Це підтвердило правильність налаштування ролей і дозволів, а також коректність взаємодії API з базою даних.

На рисунку 4.15 представлено інтерфейс Swagger, через який здійснювалися запити. Він надає зручний інструмент для перевірки роботи API, забезпечуючи детальний опис кожного ендпоінту, доступні методи запитів та їх параметри.

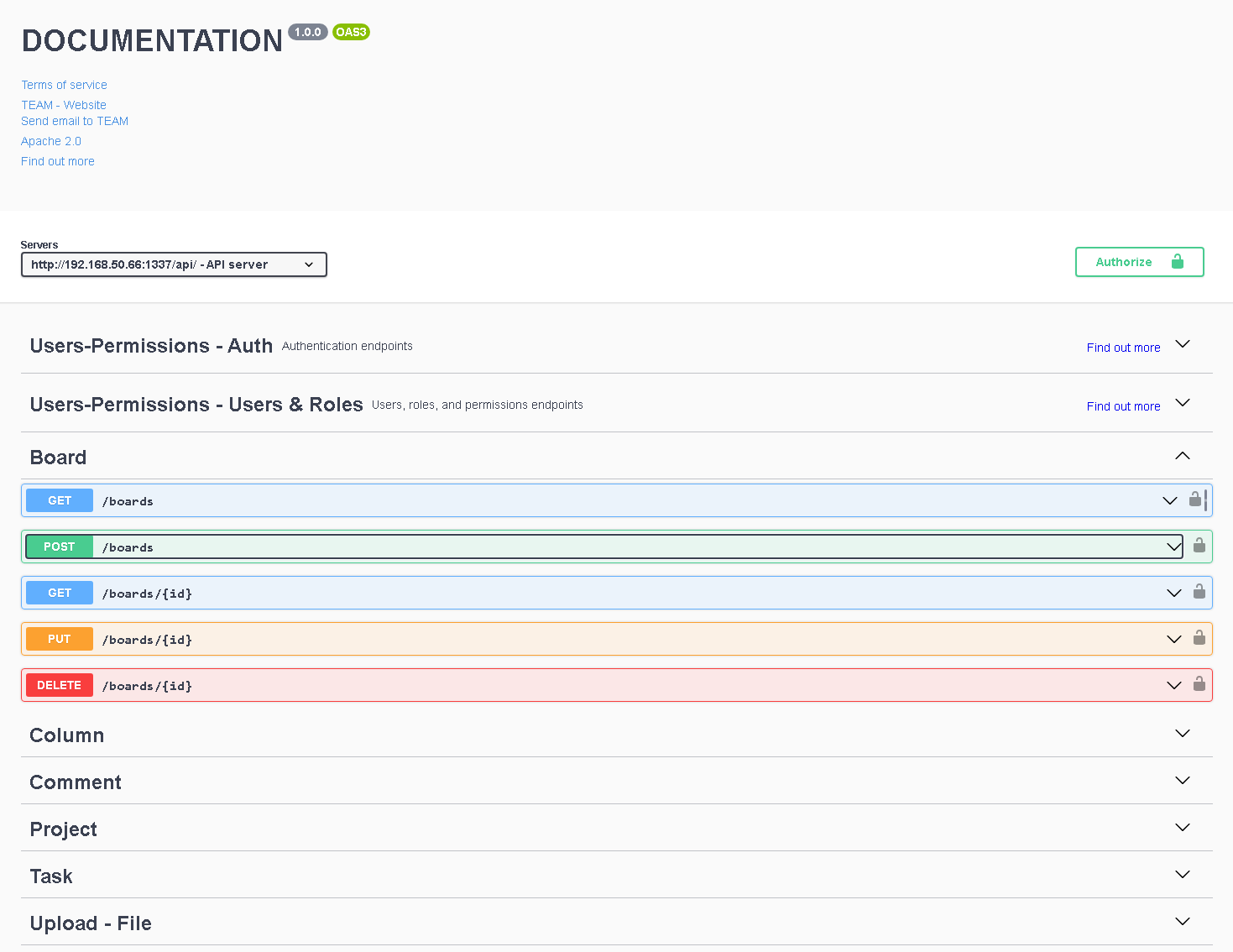


Рисунок 4.15 – Тестування API через Swagger-документацію

* 1. Перевірка роботи запитів через Postman

Для перевірки роботи API-запитів у магістерській роботі використано інструмент Postman [19]. Postman є потужним засобом для тестування та роботи з API, що спрощує процес створення, організації та автоматизації запитів.

Основні можливості Postman включають:

* створення та тестування запитів: підтримуються різні типи HTTP-запитів (GET, POST, PUT, DELETE) з можливістю налаштування URL, параметрів, заголовків та тіла запиту;
* організація запитів: запити можна зберігати у колекціях, що полегшує їх повторне використання та структурування за проектами;
* авторизація: підтримуються різні методи авторизації, такі як Bearer Token та OAuth, що забезпечує безпечний доступ до API;
* автоматизація тестування: можливість написання тестів на JavaScript дозволяє автоматизувати перевірку відповідей сервера та валідацію даних;
* імпорт та експорт: полегшує обмін колекціями запитів між членами команди.

Для ефективної роботи з API у Postman було створено середовище (Environment) з двома змінними:

base\_url: Зберігає базову URL-адресу API (http://192.168.50.66:1337), що дозволяє уникнути повторного введення адреси в кожному запиті.

auth\_token: Зберігає токен для авторизації, забезпечуючи автоматичне додавання його до запитів без необхідності ручного введення. Токен використовується той, що було раніше створено через UI інтерфейс адміністрування Strapi.

При створенні запитів використовуються ці змінні, наприклад: {{base\_url}}/api/projects. Це спрощує процес тестування та підвищує ефективність роботи з API. Приклад запиту відображено на рисунку 4.16.

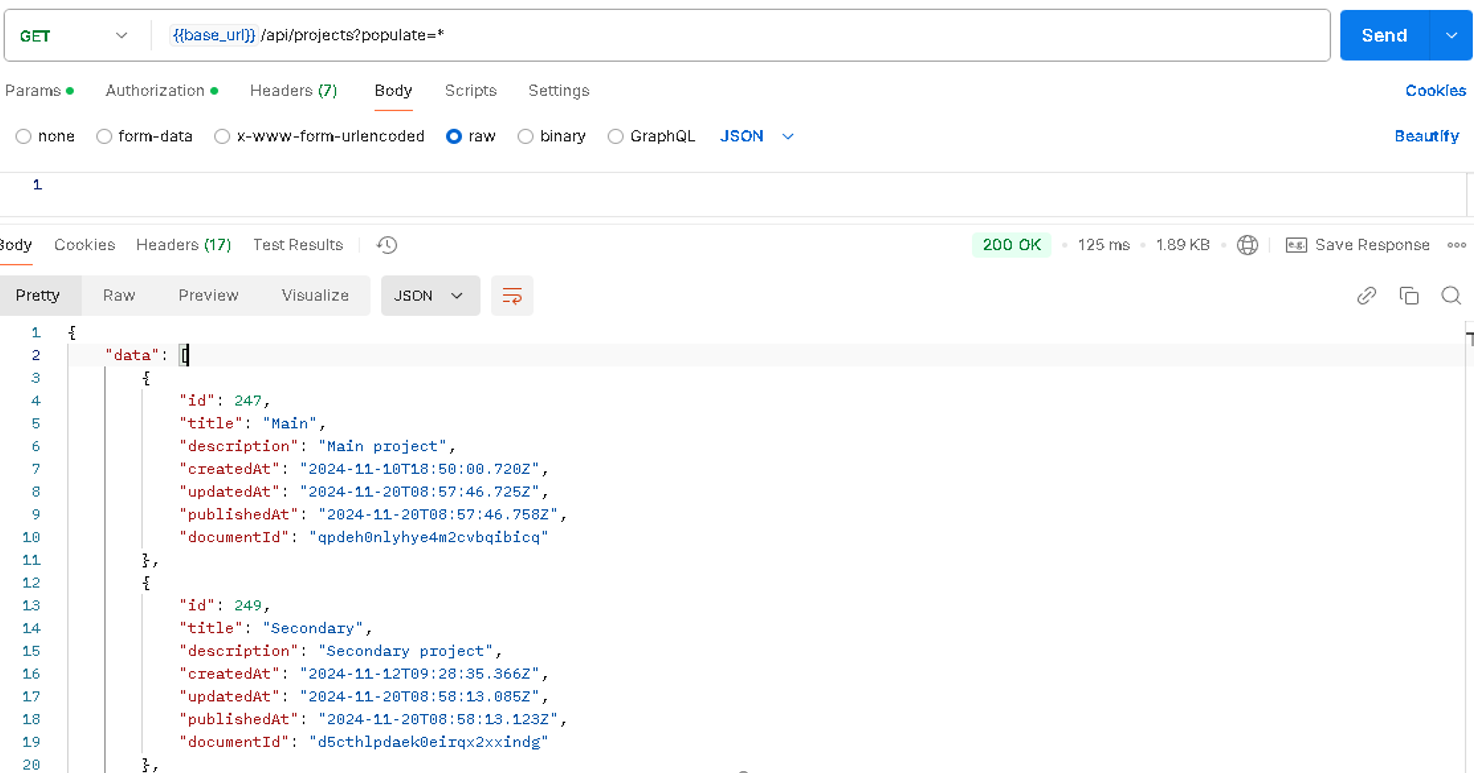


Рисунок 4.16 – Приклад запиту через Postman

* 1. Створення компонентів Vue для проєкту та їх налаштування

У процесі розробки клієнтської частини системи управління задачами було використано фреймворк Vue.js для створення реактивного та інтуїтивно зрозумілого інтерфейсу користувача. Це дозволило забезпечити високий рівень взаємодії з користувачем та створити гнучку архітектуру застосунку. Основні компоненти, які були розроблені та налаштовані, включають:

* Login.vue: сторінка авторизації користувачів;
* Register.vue: сторінка реєстрації нових користувачів;
* Home.vue: головна сторінка застосунку;
* SideMenu.vue: бічне меню для навігації між сторінками;
* UsersList.vue: сторінка адміністрування користувачів (доступна лише адміністраторам);
* Projects.vue: сторінка управління проєктами (доступна лише адміністраторам та менеджерам);
* Boards.vue: сторінка управління дошками та завданнями (адміністрування дошок доступне лише адміністраторам та менеджерам, користувачі можуть змінювати тільки параметри колонок);
* TaskModal.vue: модальне вікно для роботи з окремими завданнями та додаванням коментарів.
  + 1. Компонент Login.vue

Компонент Login.vue відповідає за реалізацію сторінки авторизації користувачів у системі. Основна функція компонента полягає у забезпеченні можливості користувачам входити в систему через форму авторизації, яка включає поля для введення електронної пошти та пароля.

Компонент використовує двостороннє зв'язування даних (v-model) для забезпечення реактивності полів вводу, що дозволяє автоматично оновлювати значення змінних при введенні даних користувачем. При поданні форми викликається метод handleLogin, який відповідає за обробку процесу авторизації. Цей метод здійснює API-запит до бекенду з використанням введених електронної пошти та пароля. У випадку успішної автентифікації сервер повертає JWT-токен, який зберігається в локальному сховищі браузера. Це дозволяє ідентифікувати користувача при подальших запитах до API, забезпечуючи безперебійну роботу системи.

Компонент має зручний та інтуїтивно зрозумілий інтерфейс, що включає форму для введення облікових даних та кнопки для входу та реєстрації (див. рисунок 4.17). Після успішного входу користувача перенаправляє на головну сторінку застосунку, а також надає можливість перейти на сторінку реєстрації для нових користувачів. Збереження JWT-токена в локальному сховищі забезпечує збереження стану автентифікації між сесіями користувача.

Обробка помилок реалізована за допомогою блоків try-catch, що дозволяє відслідковувати та інформувати користувача про можливі проблеми під час входу, такі як некоректні облікові дані або технічні неполадки.

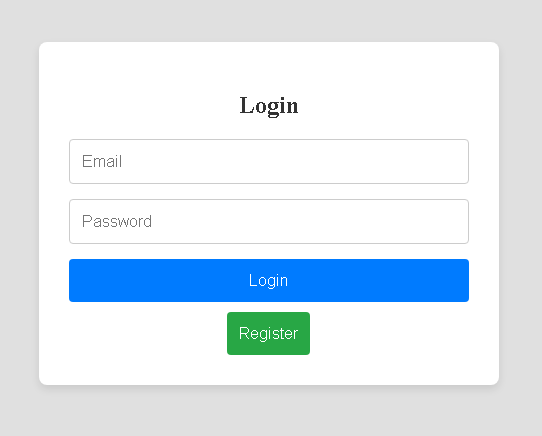


Рисунок 4.17 – Форма для авторизації

* + 1. Компонент Register.vue

Сторінка реєстрації надає можливість новим користувачам створити обліковий запис (див. рисунок 4.18). Форма реєстрації включає поля для введення імені користувача, електронної пошти та пароля. Реалізована валідація введених даних як на фронтенді (перевірка на порожні поля, співпадіння паролю), так і на бекенді (перевірка унікальності електронної пошти).

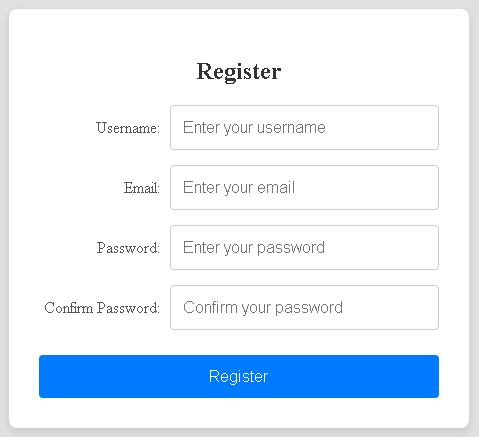


Рисунок 4.18 – Форма для реєстрації

Після успішної реєстрації користувач автоматично авторизується та перенаправляється на сторінку дошок. Відображається повідомлення з підтвердженням успішної реєстрації та зазначенням призначеної ролі. Користувачу автоматично надається роль звичайного користувача, яку в подальшому адміністратор може змінити на менеджера або адміністратора.

* + 1. Компонент Home.vue

Головна сторінка застосунку служить початковою точкою взаємодії користувача з системою (див. рисунок 4.19). Вона містить привітальне повідомлення та детальний опис методології Kanban, яка використовується для організації робочого процесу. У дизайні сторінки використано чисті та сучасні стилі, з акцентом на читабельність та простоту сприйняття інформації.

Особливістю компонента є його інтеграція з SideMenu.vue, що забезпечує постійну доступність навігації для користувача. Це сприяє зручності користування та швидкому переходу між розділами системи.

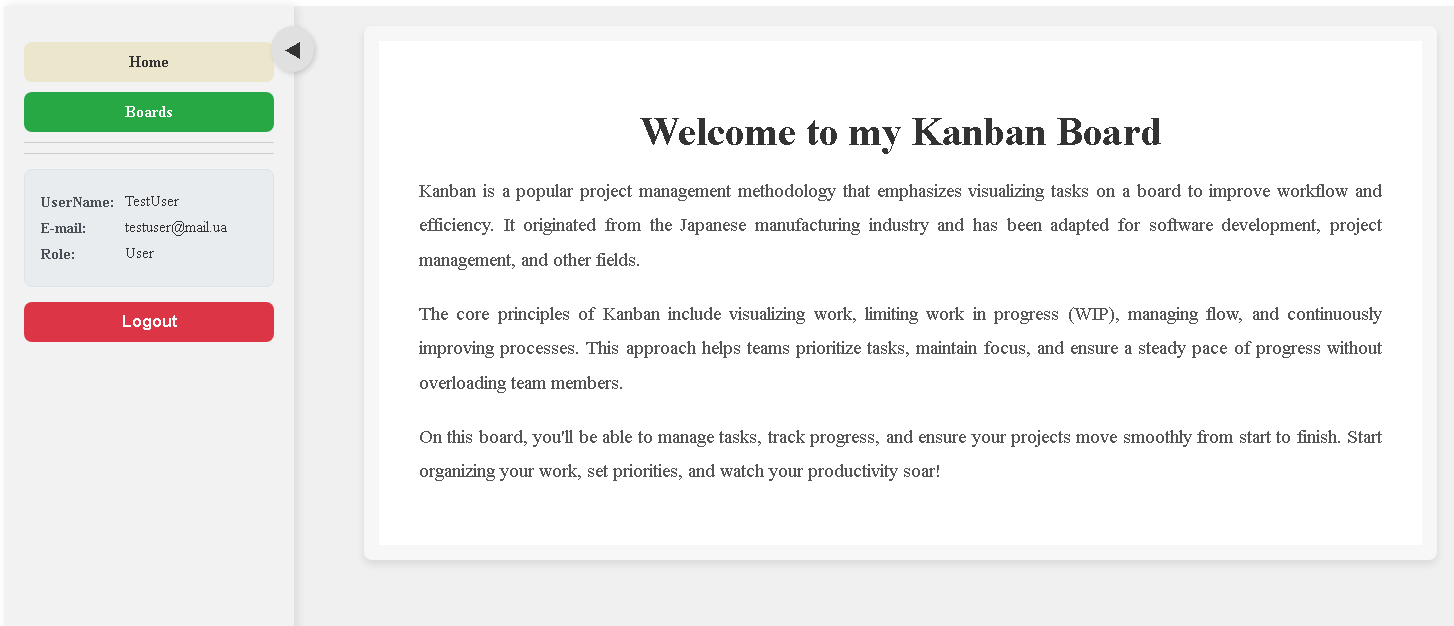


Рисунок 4.19 – Головна сторінка

* + 1. Компонент SideMenu.vue

Бічне меню є ключовим елементом навігації в застосунку. Воно відображається на всіх сторінках і динамічно змінює свій вміст залежно від ролі користувача. Наприклад, для адміністраторів доступні додаткові пункти меню, такі як "Admin Panel", тоді як звичайні користувачі бачать лише основні розділи.

Використання Vue Router дозволяє забезпечити плавний перехід між сторінками без перезавантаження, що покращує користувацький досвід. Крім того, меню містить інформацію про поточного користувача, включаючи його ім'я, електронну пошту та роль, що додає персоналізації інтерфейсу.

* + 1. Компонент UsersList.vue

Ця сторінка доступна лише адміністраторам і призначена для управління користувачами системи (див. рисунок 4.20). Він відображає таблицю з детальною інформацією про кожного користувача, включаючи:

* ID: унікальний ідентифікатор користувача;
* Username: ім'я користувача;
* Email: електронна пошта;
* Role: роль користувача в системі;
* Blocked Status: статус блокування;
* Created At / Updated At: дати створення та останнього оновлення облікового запису.

Реалізовано функції сортування таблиці за будь-яким з полів шляхом натискання на заголовки стовпців. Адміністратор може змінювати роль користувача через випадаючий список, блокувати/розблоковувати облікові записи та видаляти їх. Перед виконанням критичних дій система запитує підтвердження, що запобігає випадковим змінам.

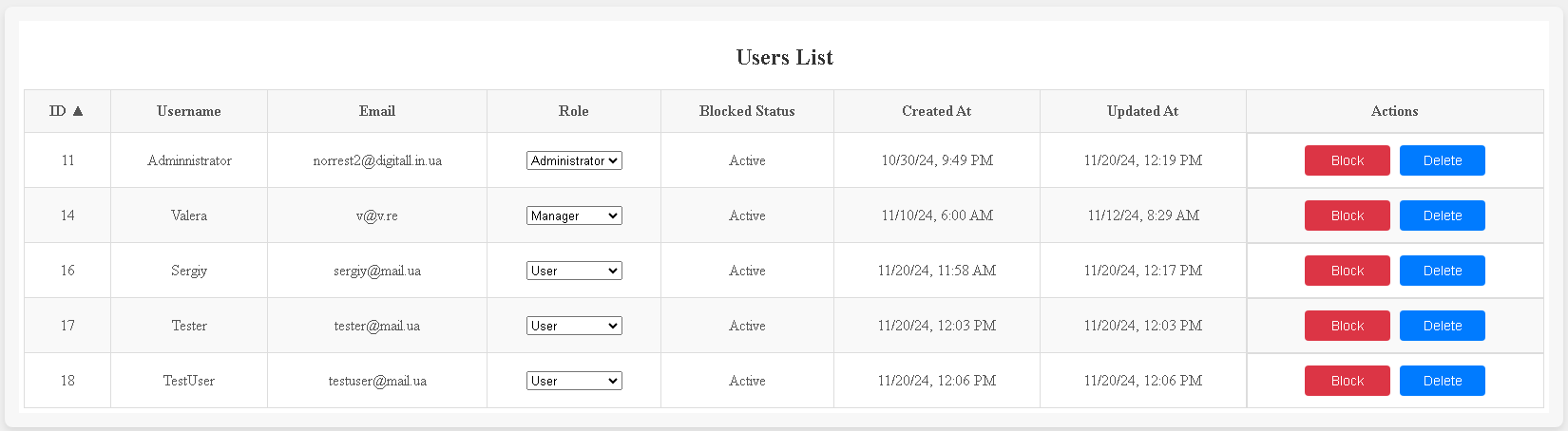


Рисунок 4.20 – Адміністрування користувачів

* + 1. Компонент Projects.vue

Сторінка управління проєктами дозволяє менеджерам та адміністраторам створювати нові проєкти, редагувати існуючі та видаляти їх (див. рисунок 4.21). Відображається список проєктів з можливістю сортування за різними параметрами.

Особливості компонента:

* редагування в лінію: назву та опис проєкту можна редагувати безпосередньо в таблиці, що підвищує зручність та швидкість роботи;
* валідація введених даних: під час редагування або створення проєкту проводиться перевірка довжини полів та наявності обов'язкових даних. При створенні та зміни назви проєкта відбувається додаткова перевірка на унікальність;
* обмеження на видалення: проєкт можна видалити лише якщо він не містить пов'язаних дошок, що запобігає втраті важливих даних.

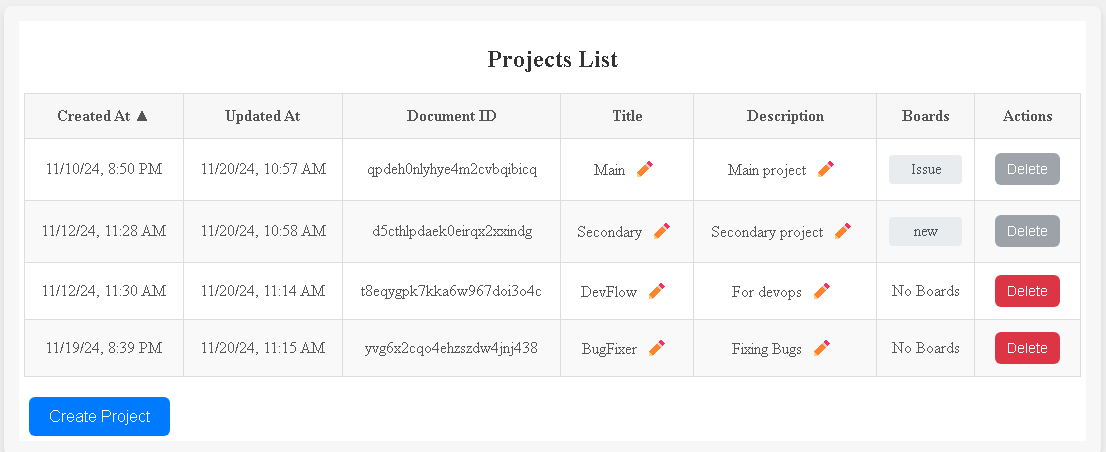


Рисунок 4.21 – Адміністрування проєктів

* + 1. Компонент Boards.vue

Цей компонент є центральним у застосунку, оскільки реалізує функціональність Kanban-дошок (див. рисунок 4.22). Користувачі можуть обирати проєкт та переглядати пов'язані з ним дошки. Для кожної дошки, відповідно до ролі, доступні наступні дії:

* створення та редагування дошок: з можливістю вказати назву, опис, колір;
* редагування дошок з можлисовтями архівації. Архівовані дошки позначаються спеціальним індикатором;
* додавання та управління колонками: користувач може додавати нові колонки, змінювати їх назви та кольори, а також видаляти їх (якщо вони не містять завдань);
* додавання та управління завданнями: завдання додаються до колонок з вказанням назви, опису та дати завершення. Дата завершення підсвічується відповідним кольором. Зелений колір означає, що до дедлайну залишилося більше трьох днів, оранжевий — один-два дні, а червоний сигналізує про нагальність або пропущений дедлайн;
* перетягування елементів: реалізовано можливість перетягування як колонок, так і завдань за допомогою плагіна Vue.Draggable. Це дозволяє користувачам інтуїтивно змінювати порядок елементів та переміщувати завдання між колонками. Після завершення операції перетягування відправляється запит до бекенду для оновлення позицій елементів у базі даних, що забезпечує збереження актуального стану дошки та гарантує, що зміни будуть відображені для всіх користувачів у режимі реального часу.

Компонент Boards.vue інтегрується з іншими частинами застосунку, такими як бокове меню (SideMenu.vue) та модальні вікна для управління завданнями (TaskModal.vue). Це забезпечує єдину та послідовну взаємодію користувача з системою, дозволяючи легко перемикатися між різними функціями та отримувати доступ до необхідної інформації.

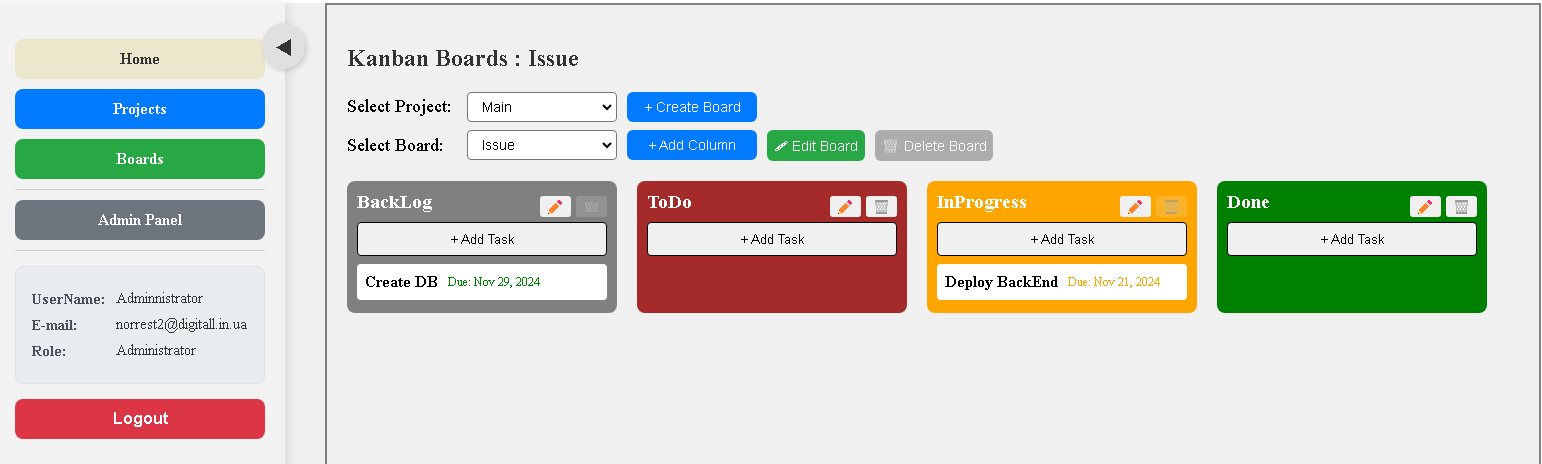


Рисунок 4.22 – Робота з дошками Kanban

* + 1. Компонент TaskModal.vue

Компонент TaskModal.vue забезпечує інтерактивне модальне вікно для роботи з окремими завданнями у системі управління проектами (див. рисунок 4.23). Основні функції компонента:

* перегляд та редагування деталей завдання: компонент дозволяє користувачам змінювати назву, опис та дату завершення завдання; при виборі дати завершення автоматично розраховується кількість днів до дедлайну, і відображається відповідний кольоровий індикатор;
* призначення виконавця: користувачі можуть призначати завдання конкретним учасникам проекту зі списку доступних користувачів, що сприяє ефективному розподілу завдань серед членів команди;
* коментування: компонент підтримує додавання, редагування та видалення коментарів до завдання; користувачі можуть залишати коментарі, редагувати тільки свої власні за необхідності; адміністратори мають можливість видаляти будь-які коментарі, що забезпечує контроль за змістом комунікації та підтримку порядку;
* управління завданням: компонент надає можливість зберігати внесені зміни або видаляти завдання з підтвердженням дії; це гарантує, що важливі зміни не будуть внесені випадково, а також забезпечує безпеку даних при видаленні завдань.

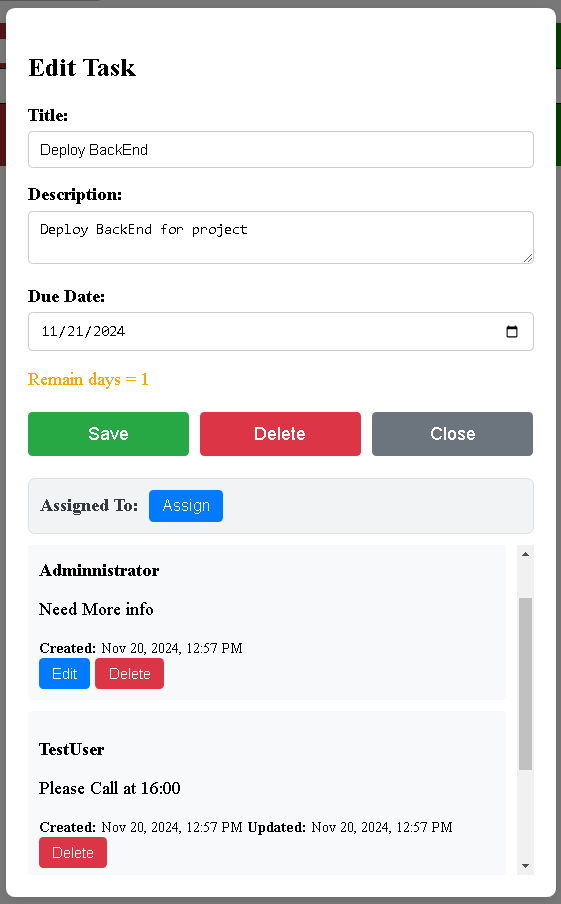


Рисунок 4.23 – Робота з задачами Kanban

* 1. Висновки до розділу 4

У четвертому розділі було детально розглянуто процес програмної реалізації системи управління задачами та її інтеграцію з обраними технологіями. На початку було обґрунтовано вибір операційної системи Debian Linux версії 6.1.0-21-amd64 для серверного середовища, що забезпечує стабільність, надійність та ефективне використання ресурсів. Використання консольної версії без графічного інтерфейсу дозволило оптимізувати продуктивність та підвищити безпеку системи.

Подальші кроки включали встановлення та налаштування необхідного програмного забезпечення, зокрема Node.js та пакетного менеджера Yarn, які стали основою для роботи з платформою Strapi. Встановлення та конфігурація Strapi як бекенд-фреймворку дозволили швидко створити RESTful API та зручну адміністративну панель для управління контентом.

У процесі налаштування Strapi було створено основні колекції даних: Project, Board, Column, Task, Comment та User. Це забезпечило структурування даних відповідно до логіки системи управління задачами. Налаштування відношень між колекціями та детальна конфігурація прав доступу для різних ролей користувачів (Administrator, Manager, User) гарантували безпеку та гнучкість системи.

Для покращення документування та тестування API було інтегровано Swagger, що спростило процес ознайомлення з ендпоінтами та перевірки їхньої роботи. Налаштування CORS забезпечило безпечний доступ до API з різних джерел, а тестування запитів через Postman підтвердило коректність та надійність роботи бекенду.

На фронтенд-частині було встановлено та налаштовано Vue.js з використанням Vue CLI, що дало можливість створити реактивний та інтуїтивно зрозумілий інтерфейс користувача. Розроблено ключові компоненти застосунку, такі як Login.vue, Register.vue, Home.vue, SideMenu.vue, UsersList.vue, Projects.vue, Boards.vue та TaskModal.vue. Кожен з цих компонентів був детально налаштований та інтегрований, забезпечуючи повну функціональність системи.

Особливу увагу було приділено інтеграції фронтенду з бекендом через RESTful API. Використання бібліотеки Axios для виконання HTTP-запитів та налаштування механізмів аутентифікації з використанням JWT-токенів забезпечили безперебійну та безпечну взаємодію між клієнтською та серверною частинами.

Розробка компонентів Vue.js дозволила створити зручний та сучасний інтерфейс, який відповідає принципам методології Kanban. Реалізовано можливість динамічного управління дошками, колонками та завданнями, включаючи функції перетягування елементів за допомогою плагіна Vue.Draggable. Це значно покращило користувацький досвід та підвищило ефективність роботи з системою.

У результаті проведеної роботи було створено повнофункціональну систему управління задачами, яка відповідає сучасним вимогам до вебзастосунків. Використання передових технологій та фреймворків забезпечило гнучкість, масштабованість та легкість подальшого розвитку системи. Цей розділ закладає міцну основу для наступних етапів проєкту, таких як тестування, оптимізація та впровадження системи у робоче середовище.

Таким чином, виконані завдання четвертого розділу підтверджують доцільність обраних технологій та підходів, а також демонструють практичну реалізацію системи, готової до подальшого розгортання та використання в реальних умовах.

**ВИСНОВКИ**

У магістерській роботі було проведено комплексне дослідження та розробку системи управління задачами для IT-проєктів, що базується на методології Kanban. Метою роботи було створення концептуальної основи та програмної реалізації системи, яка забезпечує гнучкість, ефективність та адаптивність в управлінні задачами сучасних IT-команд.

У першому розділі проведено детальний теоретичний огляд існуючих методологій та підходів у сфері систем управління задачами. Розглянуто традиційну модель "Водоспад" та гнучкі методології Agile, зокрема Scrum, Kanban та Extreme Programming (XP). На основі порівняльного аналізу визначено переваги та недоліки кожної методології, що дозволило обґрунтовано обрати Kanban як оптимальний підхід для розробки системи.

У другому розділі здійснено огляд сучасних програмних рішень у сфері управління задачами, таких як JIRA, Trello, Asana та Microsoft Project. Проведено порівняння їх функціональних можливостей, підтримки методологій, гнучкості налаштувань та інтеграцій. Це дало можливість виявити недоліки існуючих систем та визначити вимоги до розроблюваної системи.

У третьому розділі сформульовано основні функціональні та нефункціональні вимоги до системи управління задачами. Обґрунтовано вибір методології Kanban, що забезпечує простоту, візуальну відкритість та адаптивність до різних проєктів і робочих процесів. Розроблено концептуальну модель системи, яка складається з фронтенду, бекенду та бази даних, що взаємодіють між собою через RESTful API.

У четвертому розділі детально описано процес програмної реалізації системи. Обрано технологічний стек, що включає Node.js, Strapi для бекенду та Vue.js для фронтенду. Проведено встановлення та налаштування необхідних компонентів, створено основні колекції даних та налаштовано ролі користувачів з відповідними правами доступу. Реалізовано інтеграцію фронтенду з бекендом, розроблено ключові компоненти інтерфейсу користувача, що забезпечують функціональність системи згідно з вимогами.

Результатом роботи є повнофункціональна система управління задачами, яка відповідає сучасним вимогам до вебзастосунків та потребам IT-команд. Система забезпечує ефективне управління проєктами та задачами, покращує комунікацію між членами команди, надає інтуїтивно зрозумілий інтерфейс та підтримує гнучкість в управлінні робочими процесами.

Основні наукові та практичні результати роботи:

* аналіз методологій: проведено глибокий аналіз існуючих методологій управління задачами та обґрунтовано вибір Kanban як оптимальної для розробки системи;
* розробка концептуальної моделі: створено концептуальну модель системи, яка інтегрує принципи Kanban з сучасними технологічними рішеннями, забезпечуючи гнучкість та адаптивність;
* програмна реалізація: реалізовано прототип системи з використанням Node.js, Strapi та Vue.js, що демонструє практичну застосовність запропонованої концепції;
* безпека та цілісність даних: налаштовано механізми аутентифікації та авторизації, забезпечено захист даних та контроль доступу відповідно до ролей користувачів;
* інтуїтивний інтерфейс: створено зручний інтерфейс користувача, який відповідає принципам UX/UI дизайну та сприяє підвищенню ефективності роботи команди;

Практична значимість роботи полягає у можливості впровадження розробленої системи в реальні IT-проєкти, що сприятиме оптимізації робочих процесів, підвищенню продуктивності та якості виконання задач. Система може бути адаптована до потреб різних організацій та масштабована для роботи з великими командами.

Подальший розвиток роботи може бути спрямований на:

* розширення функціональності: додавання модулів аналітики, звітності, інтеграції з іншими інструментами (наприклад, системами контролю версій, CI/CD);
* оптимізацію продуктивності: поліпшення продуктивності системи при високих навантаженнях, впровадження механізмів кешування та оптимізації запитів до бази даних;
* покращення користувацького досвіду: впровадження адаптивного дизайну для мобільних пристроїв, локалізації інтерфейсу, налаштування повідомлень та сповіщень;
* апробацію в реальних умовах: проведення тестування системи в реальних проєктах, збір зворотного зв’язку від користувачів та подальше вдосконалення на основі отриманих даних;

Виконана магістерська робота підтверджує актуальність та важливість досліджень у сфері управління задачами для IT-проєктів. Запропоновані рішення та розроблена система мають практичну цінність та можуть бути використані як основа для подальших досліджень та розробок у цій галузі.

**СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ ТА ЛІТЕРАТУРИ**

1. Мандрика Ю.В. Оптимізація управління ІТ-проєктами: основні принципи і можливі інструменти для розробки гнучкої системи на основі Канбан. *Вісник студентського наукового товариства:* збірник наукових праць студентів, магістрантів і аспірантів. *Випуск 30.* Ніжин: НДУ ім. М. Гоголя*.* 2024. C.34-38. URL: [http://www.ndu.edu.ua/storage/styd\_visnuk/visnuk](http://www.ndu.edu.ua/storage/styd_visnuk/visnuk_stud_tov_30_2024.pdf) (дата звернення: 10.10.2024).
2. Мандрика Ю.В. Application Programming Interface (API): основи та сучасні підходи. *Наукові записки молодих учених* / за матеріалами Всеукраїнської науково-практичної конференції «Математичні, природничі, комп’ютерні науки та науки про управління, технології, навчання: науково-практичні рішення та підходи молодих науковців», 28 листопада 2024 р., Центральноукраїнський державний університет імені Володимира Винниченка у м. Кропивницькому. № 13. 2024. URL: [https://phm.cuspu.edu.ua/ojs/ index.php/SNYS/issue/view/59/showToc](https://phm.cuspu.edu.ua/ojs/%20index.php/SNYS/issue/view/59/showToc) (дата звернення: 30.11.2024).
3. Royce W. W. Managing the Development of Large Software Systems. Proceedings of IEEE WESCON, 1970. URL: <https://www.cs.umd.edu/class/spring2003/cmsc838p/Process/waterfall.pdf> (дата звернення: 12.10.2024).
4. Pressman R. S., Maxim B. R. Software Engineering: A Practitioner's Approach. 8th Edition. McGraw-Hill, 2014.
5. Beck K. та ін. Маніфест гнучкої розробки програмного забезпечення. Agile Alliance. URL: <https://agilemanifesto.org/iso/uk/manifesto.html> (дата звернення: 12.10.2024).
6. VersionOne. 14th Annual State of Agile Report. 2020. URL: <https://stateofagile.com/> (дата звернення: 12.10.2024).
7. Schwaber K., Sutherland J. The Scrum Guide. Scrum.org, 2020. URL: <https://scrumguides.org/scrum-guide.html> (дата звернення: 12.10.2024).
8. Anderson D. J. Kanban: Successful Evolutionary Change for Your Technology Business. Blue Hole Press, 2010.
9. Beck K. Extreme Programming Explained: Embrace Change. Addison-Wesley Professional, 2000.
10. Atlassian. JIRA Software Overview. URL: <https://www.atlassian.com/software/jira> (дата звернення: 12.10.2024).
11. Trello. About Trello. URL: <https://trello.com/about> (дата звернення: 12.10.2024).
12. Asana. Product Overview. URL: <https://asana.com/product> (дата звернення: 12.10.2024).
13. Microsoft. Microsoft Project. URL: <https://www.microsoft.com/uk-ua/microsoft-365/project/project-management-software> (дата звернення: 12.10.2024).
14. Atlassian. JIRA Software Pricing. URL: <https://www.atlassian.com/software/jira/pricing> (дата звернення: 12.10.2024).
15. Vue документація URL: <https://vuejs.org/guide/introduction.html> (дата звернення: 10.10.2024).
16. Strapi документація для розробників URL: <https://docs.strapi.io/dev-docs/intro> (дата звернення: 10.10.2024).
17. PostgreSQL документація для розробників URL: <https://www.postgresql.org/docs/> (дата звернення: 10.10.2024).
18. Swagger офіціний URL: <https://swagger.io/> (дата звернення: 12.10.2024).
19. Postman офіціний URL: <https://www.postman.com/> (дата звернення: 12.10.2024).

ДОДАТКИ

Додаток А

**Конфігурація api.js для Vue**

Файл api.js є основним модулем для взаємодії з API в застосунку. Він організовує запити до бекенду за допомогою бібліотеки Axios. У ньому реалізовано функціонал для авторизації, роботи з користувачами, проєктами, дошками, колонками, тасками та коментарями.

Розглянемо структуру та ключові функції:

1. Налаштування Axios:

* api — екземпляр Axios з базовою URL-адресою, яка визначається змінною середовища VITE\_API\_URL;
* Interceptors: перехоплювач додає токен авторизації (JWT) до заголовків кожного запиту.

2. Авторизація:

* login(identifier, password): використовується для входу користувача, відправляючи identifier і password до /auth/local;
* register(username, email, password): Реєструє нового користувача через /auth/local/register.

3. Користувачі та ролі:

* getUserInfo(): отримує інформацію про поточного користувача (з полями, включаючи роль);
* getUsers(): отримує список всіх користувачів з інформацією про їх ролі;
* getRoles(): Отримує список доступних ролей;
* updateUser(userId, data): оновлює інформацію користувача (зміна ролі чи статусу);
* deleteUser(userId): видаляє користувача за його ID.

4. Проєкти:

* getProjects(): отримує список усіх проєктів;
* createProject(data): створює новий проєкт;
* updateProject(projectId, data): оновлює існуючий проєкт;
* deleteProject(projectId): видаляє проєкт.

5. Дошки (Boards.vue):

* fetchProjectsListForBoards(): отримує список проєктів для відображення у вигляді дошок;
* getBoardsForProject(projectId): отримує дошки, пов’язані з конкретним проєктом;
* createBoard(projectId, data): створює нову дошку;
* getBoardsByDocumentId(documentId): отримує дошки за унікальним ідентифікатором проєкту;
* updateBoard(documentId, data): оновлює інформацію про дошку.
* deleteBoard(documentId): видаляє дошку за її унікальним ідентифікатором.

6. Колонки:

* getColumnsForBoard(boardId): отримує колонки певної дошки, відсортовані за позицією;
* createColumn(boardId, data): створює нову колонку.
* updateColumnPosition(documentId, data): оновлює позицію колонки.
* updateColumnTitle(documentId, data): оновлює назву колонки.
* deleteColumnByDocumentId(documentId): видаляє колонку за унікальним ідентифікатором.

7. Таски:

* createTask(columnId, data): створює новий таск у колонці;
* getTasksForColumn(columnId): отримує таски певної колонки разом із інформацією про користувача, відповідального за них;
* updateTask(taskId, data): оновлює інформацію про таски;
* updateTaskPosition(taskId, data): оновлює позицію таски;
* deleteTask(taskId): видаляє таску.

8. Коментарі:

* getCommentsForTask(taskDocumentId): отримує коментарі до конкретної таски разом із даними користувачів;
* createComment(text, taskDocumentId, userDocumentId): створює новий коментар до таски;
* updateComment(commentDocumentId, text, taskId, userId): оновлює текст коментаря;
* deleteComment(commentDocumentId): видаляє коментар.

Додаткова функціональність:

* запити для зв'язків: наприклад, getBoardsWithProject() — дозволяє отримати дошки разом із їхніми проєктами;
* використання: цей файл забезпечує централізовану обробку запитів до API, що спрощує їх інтеграцію в компоненти, такі як Projects.vue та Boards.vue.

// api.js

import axios from 'axios';

const api = axios.create({

baseURL: import.meta.env.VITE\_API\_URL,

});

// Функція для логіну

export const login = async (identifier, password) => {

try {

const response = await api.post('/auth/local', {

identifier,

password,

});

return response.data;

} catch (error) {

console.error('Login error:', error);

throw error;

}

};

// Функція для реєстрації

export const register = async (username, email, password) => {

try {

const response = await api.post('/auth/local/register', {

username,

email,

password,

});

return response.data;

} catch (error) {

console.error('Registration error:', error);

throw error;

}

};

// Додавання interceptor для автоматичного додавання токена до запитів

api.interceptors.request.use(

(config) => {

const token = localStorage.getItem('jwt');

if (token) {

config.headers.Authorization = `Bearer ${token}`;

}

return config;

},

(error) => Promise.reject(error)

);

// Функція для отримання інформації про поточного користувача

export const getUserInfo = async () => {

try {

const response = await api.get('/users/me?populate=role');

return response.data;

} catch (error) {

console.error('Error fetching user info:', error);

throw error;

}

};

// Функція для отримання списку користувачів з ролями

export const getUsers = async () => {

return await api.get('/users?populate=role');

};

// Функція для отримання списку ролей

export const getRoles = async () => {

return await api.get('/users-permissions/roles');

};

// Оновлення інформації про користувача

export const updateUser = async (userId, data) => {

return await api.put(`/users/${userId}`, data);

};

// Видалення користувача

export const deleteUser = async (userId) => {

return await api.delete(`/users/${userId}`);

};

// Функція для отримання списку проєктів

export const getProjects = async () => {

try {

const response = await api.get('/projects');

return response.data; // Повертаємо об'єкт з властивістю data

} catch (error) {

console.error('Error fetching projects:', error);

throw error;

}

};

// Функція для створення нового проєкту

export const createProject = async (data) => {

try {

const response = await api.post('/projects', { data });

return response.data;

} catch (error) {

console.error('Error creating project:', error);

throw error;

}

};

// Функція для оновлення існуючого проєкту

export const updateProject = async (projectId, data) => {

try {

const response = await api.put(`/projects/${projectId}`, { data });

return response.data;

} catch (error) {

console.error('Error updating project:', error);

throw error;

}

};

// Функція для видалення проєкту

export const deleteProject = async (projectId) => {

try {

const response = await api.delete(`/projects/${projectId}`);

return response.data;

} catch (error) {

console.error('Error deleting project:', error);

throw error;

}

};

// Функція для отримання списку проєктів для Boards.vue

export const fetchProjectsListForBoards = async () => {

try {

const response = await api.get('/projects');

return response.data.data; // Повертаємо масив проєктів

} catch (error) {

console.error('Error fetching projects list for boards:', error);

throw error;

}

};

// Функція для отримання дошок певного проєкту для Boards.vue

export const getBoardsForProject = async (projectId) => {

try {

const response = await api.get(

`/boards?filters[project][$eq]=${projectId}`

);

return response.data.data; // Повертаємо масив дошок

} catch (error) {

console.error('Error fetching boards for project:', error);

throw error;

}

};

// Функція для створення нової дошки

export const createBoard = async (projectId, data) => {

try {

const response = await api.post('/boards', {

data: {

...data,

project: projectId,

},

});

return response.data;

} catch (error) {

console.error('Error creating board:', error);

throw error;

}

};

// Функція для отримання колонок певної дошки для Boards.vue

export const getColumnsForBoard = async (boardId) => {

try {

const response = await api.get(

`/columns?filters[board][$eq]=${boardId}&sort=position:asc`

);

return response.data.data; // Повертаємо масив колонок

} catch (error) {

console.error('Error fetching columns for board:', error);

throw error;

}

};

// Функція для створення нової колонки

export const createColumn = async (boardId, data) => {

try {

const response = await api.post('/columns', {

data: {

...data,

board: boardId,

},

});

return response.data;

} catch (error) {

console.error('Error creating column:', error);

throw error;

}

};

// Створення таски

export const createTask = async (columnId, data) => {

try {

const response = await api.post(`/tasks`, {

data: {

...data,

column: columnId

}

});

return response.data.data;

} catch (error) {

console.error('Error creating task:', error);

throw error;

}

};

// Отримання тасків для колонки з додатковими полями assignedTo і comments

export const getTasksForColumn = async (columnId) => {

try {

// Запит з додатковими параметрами для завантаження полів assignedTo і comments

const response = await api.get(`/tasks?filters[column][id][$eq]=${columnId}&sort=position&populate[assignedTo]=\*`);

// Мапимо отримані дані, щоб повернути потрібну структуру

return response.data.data.map((task) => ({

id: task.id,

title: task.title,

description: task.description,

position: task.position,

documentId: task.documentId,

createdAt: task.createdAt,

updatedAt: task.updatedAt,

dueDate: task.dueDate,

// Додаємо поля assignedTo та comments

assignedTo: task.assignedTo,

}));

} catch (error) {

throw error;

}

};

// Оновлення позиції таски

export const updateTaskPosition = async (taskId, data) => {

try {

const response = await api.put(`/tasks/${taskId}`, { data });

return response.data.data;

} catch (error) {

throw error;

}

};

// Оновлення позиції колонки за documentId

export const updateColumnPosition = async (documentId, data) => {

try {

const response = await api.put(`/columns/${documentId}`, { data });

return response.data;

} catch (error) {

console.error('Error updating column position:', error);

throw error;

}

};

export const deleteColumnByDocumentId = async (documentId) => {

try {

const response = await api.delete(`/columns/${documentId}`);

return response.data;

} catch (error) {

console.error('Error deleting column:', error);

throw error;

}

};

// Функція для отримання дошок за documentId проєкту

export const getBoardsByDocumentId = async (documentId) => {

try {

const response = await api.get(`/boards?filters[project][documentId][$eq]=${documentId}`);

return response.data.data; // Повертаємо масив дошок

} catch (error) {

console.error('Error fetching boards by documentId:', error);

throw error;

}

};

export const deleteBoard = async (documentId) => {

try {

const response = await api.delete(`/boards/${documentId}`);

return response.data;

} catch (error) {

console.error('Error deleting board:', error);

throw error;

}

};

// Оновлення назви колонки

export const updateColumnTitle = async (documentId, data) => {

try {

const response = await api.put(`/columns/${documentId}`, { data });

return response.data;

} catch (error) {

console.error('Error updating column title:', error);

throw error;

}

};

// Оновлення дошки

export const updateBoard = async (documentId, data) => {

try {

const response = await api.put(`/boards/${documentId}`, { data });

return response.data;

} catch (error) {

console.error('Error updating board:', error);

throw error;

}

};

/\*\*

\* Функція для оновлення таски

\* @param {String} taskId - ID таски

\* @param {Object} data - Дані для оновлення

\*/

export const updateTask = async (taskId, data) => {

try {

const response = await api.put(`/tasks/${taskId}`, { data });

return response.data;

} catch (error) {

console.error('Error updating task:', error);

throw error;

}

};

// Видалення таски

export const deleteTask = async (taskId) => {

try {

const response = await api.delete(`/tasks/${taskId}`);

return response.data;

} catch (error) {

console.error('Error deleting task:', error);

throw error;

}

};

// Функція для отримання коментарів за task.documentId

export const getCommentsForTask = async (taskDocumentId) => {

try {

const response = await api.get(`/comments?filters[task][documentId][$eq]=${taskDocumentId}&populate=user`);

return response.data.data;

} catch (error) {

console.error('Error fetching comments:', error);

throw error;

}

};

// Функція для створення нового коментаря

export const createComment = async (text, taskDocumentId, userDocumentId) => {

try {

const response = await api.post('/comments', {

data: {

text,

task: taskDocumentId,

user: userDocumentId,

},

});

return response.data.data;

} catch (error) {

console.error('Error creating comment:', error);

throw error;

}

};

/\*\*

\* Оновлює коментар

\* @param {string} commentDocumentId - Ідентифікатор коментаря

\* @param {string} text - Новий текст коментаря

\* @param {string} taskId - Ідентифікатор завдання

\* @param {string} userId - Ідентифікатор користувача

\* @returns {Object} Оновлений коментар

\*/

export async function updateComment(commentDocumentId, text, taskId, userId) {

const data = {

text: text,

task: taskId,

user: userId,

};

try {

const response = await api.put(`/comments/${commentDocumentId}`, { data });

return response.data;

} catch (error) {

console.error('Error updating comment:', error);

throw error;

}

}

/\*\*

\* Видаляє коментар

\* @param {string} commentDocumentId - Ідентифікатор коментаря

\*/

export async function deleteComment(commentDocumentId) {

try {

await api.delete(`/comments/${commentDocumentId}`);

} catch (error) {

console.error('Error deleting comment:', error);

throw error;

}

}

export const getBoardsWithProject = async () => {

try {

const response = await api.get('/boards?populate=project');

return response.data.data; // Повертає масив дошок

} catch (error) {

console.error('Error fetching boards:', error);

throw error;

}

};

export default api;

Додаток Б

**Конфігурація main.js для Vue**

Файл main.js є початковою точкою входу для Vue-застосунку. Він налаштовує додаток, додає роутер, інтегрує API-запити та забезпечує глобальні перевірки авторизації, а також налаштовує бібліотеку для індикації завантаження (NProgress).

Розглянемо структуру та ключові функції:

1. Імпорти:

* createApp — метод створення Vue-додатка.
* App.vue — головний компонент додатка.
* router — модуль маршрутизації для навігації по сторінках.
* api — кастомний інстанс Axios для виконання API-запитів.
* NProgress — бібліотека для візуалізації прогресу завантаження.
* nprogress.css — стилі для індикатора прогресу.
* main.css — глобальні стилі додатка.

2. Налаштування NProgress

* showSpinner: false — вимикає відображення спінера.
* speed: 400 — швидкість анімації (мс).
* easing: 'ease' — плавність анімації.

3. Перевірки маршрутизації (beforeEach). Виконання логіки перед переходом на нову сторінку:

* якщо токен зберігається в localStorage, він додається до заголовків авторизації;
* виконується API-запит на /users/me?populate=role, щоб отримати інформацію про поточного користувача.

4. Реалізація доступу:

* /login та /register:
  + заборонено авторизованим користувачам (переадресація на /).
* доступ до сторінок з обмеженням за ролями:
  + наприклад, сторінка /users (адміністрування користувачів) дозволена лише для ролі з id === 2;
  + для/projects доступ мають користувачі з ролями id === 2 або id === 3.
* відсутність токена:
  + дозволено доступ лише до /login та /register, всі інші маршрути перенаправляються на /login.

import { createApp } from 'vue';

import App from './App.vue';

import router from './router';

import api from './api';

import NProgress from 'nprogress';

import 'nprogress/nprogress.css';

import './assets/main.css';

NProgress.configure({ showSpinner: false, speed: 400, easing: 'ease' });

router.beforeEach(async (to, from, next) => {

NProgress.start();

const token = localStorage.getItem('jwt');

if (token) {

try {

api.defaults.headers.common['Authorization'] = `Bearer ${token}`;

const response = await api.get('/users/me?populate=role');

const user = response.data;

console.log('Current User:', user);

const userRoleId = user.role?.id;

// Забороняємо доступ до сторінки /register, якщо користувач авторизований

if (to.path === '/login' || to.path === '/register') {

NProgress.done();

next('/'); // Переадресовуємо на головну сторінку

} else if (to.path === '/users' && userRoleId !== 2) {

alert('Access denied. You do not have permission to view this page.');

NProgress.done();

next(from.fullPath); // Повертаємо користувача на попередню сторінку

} else if (to.path === '/projects' && ![2, 3].includes(userRoleId)) {

alert('Access denied. You do not have permission to view this page.');

NProgress.done();

next(from.fullPath); // Повертаємо користувача на попередню сторінку

} else {

next(); // Дозволяємо доступ

}

} catch (error) {

console.error('Error fetching user info:', error);

localStorage.removeItem('jwt'); // Видаляємо некоректний токен

NProgress.done();

if (to.path !== '/login' && to.path !== '/register') {

next('/login'); // Переадресовуємо на сторінку входу

} else {

next();

}

}

} else {

// Дозволяємо неавторизованим користувачам доступ до сторінок входу та реєстрації

if (to.path !== '/login' && to.path !== '/register') {

NProgress.done();

next('/login'); // Переадресовуємо на сторінку входу, якщо користувач неавторизований

} else {

next(); // Дозволяємо доступ до /login і /register

}

}

});

router.afterEach(() => {

NProgress.done();

});

createApp(App)

.use(router)

.mount('#app');