МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ ВОЛИНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ЛЕСІ УКРАЇНКИ

Кафедра комп'ютерних наук та кібербезпеки

МИТКАЛИК МАКСИМ ОЛЕКСАНДРОВИЧ

РОЗРОБКА ВЕБ-ДОДАТКУ ДЛЯ УПРАВЛІННЯ ПРОЕКТАМИ НА ОСНОВІ NODE.JS

Спеціальність: 122 Комп'ютерні науки

Освітньо-професійна програма: Комп'ютерні науки та інформаційні технології

Кваліфікаційна робота на здобуття освітнього ступеня «бакалавр»

Науковий керівник:

Собчук Валентин Володимирович,
Професор кафедри комп'ютерних наук та кібербезпеки

РЕКОМЕНДОВАНО ДО ЗАХИСТУ

Протокол №_____ засідання
кафедри комп'ютерних наук та
кібербезпеки від
_____ 2025 р.

Завідувач кафедри

(_____) Гришанович Т. О.

Луцьк - 2025

ЗМІСТ

ВСТУП

У сучасному світі ми можемо спостерігати стрімкий розвиток цифрових технологій, які дедалі глибше інтегруються в наше повсякденне життя. Якщо раніше для виконання будь-яких завдань потрібно було бути присутнім фізично, довго чекати на відповіді або збиратися разом офлайн, то сьогодні майже всі ці процеси можна виконати дистанційно за допомогою сучасних технологій. Квитки, спілкування, навчання, робота — усе це вже давно перейшло в онлайн-формат. Проте залишаються сфери, в яких особливо важливою є правильна організація процесів — зокрема, управління проектами.

Для якісної реалізації будь-якого проекту важливо налагодити ефективну комунікацію між членами команди, розподілити завдання, слідкувати за дедлайнами та контролювати виконання. Саме тому дедалі більшої актуальності набувають веб-додатки для управління проектами — інструменти, які дозволяють оптимізувати всі етапи командної роботи.

Однією з технологій, яка активно використовується для створення сучасних веб-додатків, є Node.js — середовище виконання JavaScript на стороні сервера. Воно забезпечує високу продуктивність, масштабованість та гнучкість, що робить його чудовим вибором для реалізації веб-сервісів із високими вимогами до швидкодії та взаємодії з користувачем у реальному часі.

Метою даної роботи є створення веб-додатку на базі Node.js для управління проектами, який дозволятиме організовувати та контролювати виконання завдань, відстежувати статуси, взаємодіяти з членами команди та забезпечить зручний і зрозумілий інтерфейс для користувачів.

Для реалізації поставленої мети необхідно виконати наступне:

- дослідити поняття управління проектами та сучасні методики у цій сфері;
- ознайомитися з технологією Node.js та суміжними інструментами веб-розробки (Express.js, MongoDB, тощо);
- проаналізувати існуючі рішення для управління проектами, їх переваги та недоліки;

- визначити вимоги до майбутнього додатку та скласти технічне завдання;
- розробити архітектуру веб-додатку;
- реалізувати базовий функціонал (реєстрація користувачів, створення проектів, додавання та редагування задач, відстеження статусу);
- застосувати базову верстку для забезпечення зручного інтерфейсу;
- протестувати додаток на предмет працездатності та стабільності;
- провести аналіз результатів реалізації та сформувати висновки щодо ефективності застосування створеного інструменту.

Об’єкт дослідження – веб-додаток для управління проектами на базі технології Node.js.

Предмет дослідження – інструменти та методики організації проектної діяльності, їх реалізація у вигляді веб-застосунку та ефективність взаємодії між учасниками проєктів у цифровому середовищі.

РОЗДІЛ 1. ОСОБЛИВОСТІ АРХІТЕКТУРИ, БУДОВИ ТА РОБОТИ ВЕБ-ДОДАТКІВ ДЛЯ УПРАВЛІННЯ ПРОЕКТАМИ

1.1. Актуальність та критичний аналіз сучасних тенденцій у веб-додатках для управління проектами

В умовах стрімкої цифрової трансформації традиційні методи організації командної роботи поступаються місцем інтерактивним цифровим платформам. Управління проектами, як одна з ключових сфер діяльності в сучасному бізнесі та IT-середовищі, потребує особливої уваги до організації комунікацій, розподілу ресурсів, контролю термінів і аналітики результатів. Усе це вимагає використання спеціалізованих інструментів, що дозволяють централізовано керувати процесами в межах одного середовища.

Поява веб-додатків для управління проектами стала відповіддю на низку викликів сучасності:

- необхідність організації роботи в розподілених або гібридних командах;
- потреба в оперативному доступі до задач та їхнього статусу з будь-якого пристрою;
- потреба в прозорій структурі завдань та гнучкому плануванні;
- високі вимоги до швидкості взаємодії та автоматизації робочих процесів.

Сучасні тенденції у сфері веб-додатків для управління проектами демонструють орієнтацію на:

- інтуїтивно зрозумілий інтерфейс (UX/UI), що знижує бар’єр входу для нових користувачів;
- адаптивність і масштабованість, що дозволяє використовувати додатки як малими командами, так і великими організаціями;
- інтеграцію з іншими сервісами, такими як пошта, хмарні сховища, календарі, системи комунікації;

- реальний час — підтримка оновлень без перезавантаження сторінки, сповіщень та миттєвого відгуку системи;
- гнучкість у кастомізації, яка дає змогу підлаштовувати інструменти під конкретну методологію (Agile, Kanban, Scrum тощо).

Водночас із розвитком технологій виникає і низка критичних моментів. Часто веб-додатки мають надлишкову або, навпаки, недостатню функціональність, не забезпечують належного рівня безпеки або є складними у використанні без попередньої підготовки. Важливим залишається також питання приватності даних, особливо в умовах, коли робота з проектами передбачає обмін конфіденційною інформацією.

Крім того, не всі існуючі рішення враховують специфіку окремих команд або типів проектів. Це створює попит на розробку гнучких веб-додатків, адаптованих до конкретних потреб користувача, що можливо реалізувати за допомогою сучасних технологій веб-розробки — зокрема, таких як Node.js, Express.js та інші суміжні інструменти.

Отже, на сьогодні актуальність створення нових, функціонально збалансованих, швидких і безпечних веб-додатків для управління проектами зростає. Такий підхід не лише покращує ефективність роботи команд, а й відповідає глобальним тенденціям цифрового розвитку.

1.2. Методологічні підходи до управління проектами в контексті веб-додатків

У сфері веб-розробки управління проектами має ключове значення, адже від організації процесу безпосередньо залежить якість, терміни та ефективність створення продукту. Вибір підходу до управління проектом визначається багатьма факторами: технічними особливостями проєкту, вимогами замовника, масштабом задач, доступними ресурсами та навіть культурою роботи команди. Різні методології пропонують свої принципи організації процесу, що дозволяє адаптувати управлінську стратегію під конкретні умови реалізації веб-додатків.

Традиційний каскадний підхід, або Waterfall, передбачає послідовне проходження етапів розробки — від початкового збору вимог до остаточного тестування і впровадження. У цьому підході кожен етап повинен бути завершений до початку наступного, що забезпечує структурованість і передбачуваність. Waterfall застосовується у випадках, коли проект має чітко визначені цілі, стабільні вимоги, а зміни в процесі розробки є малоймовірними. Однак у сфері веб-додатків, де часто трапляються динамічні зміни та уточнення функціоналу вже в процесі реалізації, цей підхід не завжди виявляється достатньо гнучким.

У відповідь на потребу в адаптивності з'явились гнучкі методології, зокрема Agile. Agile-методології базуються на ітеративному підході: розробка відбувається у коротких циклах, після кожного з яких проводиться оцінка результатів і, за потреби, коригування подальших дій. Це дозволяє команді постійно враховувати зворотній зв'язок від клієнта, швидко реагувати на зміну вимог, оновлювати функціонал і покращувати якість продукту. У веб-проєктах, де пріоритет має швидке виведення результату на ринок і постійне вдосконалення функцій, Agile є одним із найефективніших підходів.

Однією з найпоширеніших реалізацій Agile є Scrum. Він передбачає поділ розробки на спринти — короткі часові проміжки (зазвичай 1–4 тижні), в межах яких команда фокусується на завершенні конкретного набору задач. Рольова структура Scrum включає власника продукту, скрам-майстра та команду розробників, кожен з яких має свої функції у забезпеченні безперервного руху проекту вперед. Завдяки постійним ретроспективам та плануванню, Scrum дозволяє зберігати високу продуктивність команди та ефективно реагувати на зміни.

Іншим популярним підходом, що часто використовується у веб-розробці, є Kanban. Його головна ідея полягає в постійному візуальному контролі над потоком задач. Кожна задача переміщується між етапами на дошці, що дозволяє миттєво бачити навантаження, виявляти вузькі місця у процесі та підтримувати стабільний ритм виконання. Kanban не вимагає жорсткого планування або поділу

на спринти, тому він часто використовується там, де робота над проектом є безперервною, або коли потрібно оперативно перерозподіляти ресурси між задачами.

Усі ці методології мають право на існування у сфері веб-додатків. Залежно від типу продукту та бізнес-цілей, команди можуть використовувати один підхід або поєднувати кілька. Наприклад, стратегічне планування може базуватись на Waterfall, у той час як етап розробки реалізується з використанням Scrum або Kanban. Такий гібридний підхід дозволяє гнучко поєднувати чіткість класичних схем з адаптивністю гнучких підходів.

Таким чином, методології управління проектами є не просто технікою організації праці, а фундаментальним елементом, що впливає на ефективність створення веб-додатків. Знання й правильне застосування цих підходів дозволяє мінімізувати ризики, скоротити витрати, покращити якість продукту та забезпечити задоволення потреб замовника і кінцевого користувача. У сучасних умовах, коли вимоги до цифрових продуктів постійно змінюються, вибір гнучкої та адаптивної методології стає запорукою успішної реалізації проектів у сфері веб-розробки.

1.3. Вплив архітектури та UX/UI на ефективність управління проектами

У сучасному цифровому середовищі, де проекти дедалі частіше реалізуються з використанням веб-додатків, ефективність управління проектною діяльністю значною мірою залежить не лише від організаційних підходів, а й від якості програмного забезпечення. Два ключові аспекти, що мають критичне значення у цьому контексті, — це архітектура веб-додатку та рівень розробки UX/UI. Їхній синергічний вплив забезпечує зручність, надійність і адаптивність системи, що в кінцевому результаті відображається на продуктивності команд, ефективності прийняття рішень і швидкості реалізації проєктних завдань.

Архітектура веб-додатку — це фундамент, на якому будується вся система. Вона визначає, яким чином функціональні модулі взаємодіють між собою, як

обробляються запити, як забезпечується збереження даних, безперервність роботи та безпека користувачів. У сфері управління проєктами, де важливо забезпечити одночасну роботу багатьох користувачів з різними правами доступу, велике навантаження на базу даних, часті оновлення й інтеграції з іншими сервісами, вибір правильної архітектури визначає успішність усього технічного рішення. Застарілі або невдалі архітектурні моделі можуть призвести до повільної роботи системи, втрати даних, складності з масштабуванням та оновленням функціоналу.

Серед найбільш перспективних підходів — мікросервісна архітектура, яка дозволяє розділити систему на незалежні модулі. Кожен модуль відповідає за окрему функцію, наприклад, роботу з завданнями, управління користувачами або генерацію звітів. Така структура забезпечує гнучкість, полегшує масштабування, дозволяє впроваджувати нові функції без зупинки всієї системи. Крім того, це суттєво полегшує тестування та обслуговування, оскільки помилки в одному сервісі не впливають на роботу інших частин. Це особливо актуально у великих командах з інтенсивною розробкою. Водночас, у проєктах із менш складною логікою все ще може використовуватися монолітна архітектура, яка простіша у реалізації на стартових етапах і вимагає менше ресурсів для запуску.

Однак технічна сторона — це лише частина картини. Рівень взаємодії користувача з системою — не менш важливий показник, який прямо впливає на ефективність використання веб-додатку в щоденній роботі. Саме UX (user experience) та UI (user interface) формують загальне враження користувача від роботи з системою. Навіть найтехнологічніша платформа може залишитися невикористаною, якщо її інтерфейс складний для розуміння, а логіка взаємодії — заплутана чи неочевидна.

Добре спроєктований UX забезпечує логічну структуру платформи, інтуїтивне розміщення елементів, зручну навігацію, а також зменшує кількість кроків, необхідних для досягнення цілі. Це особливо важливо для інструментів, які активно використовуються кожного дня — як-от системи управління завданнями, планування часу, моніторингу прогресу чи звітності. Висока якість

UX сприяє швидкому засвоєнню платформи новими користувачами, зменшує кількість помилок, підвищує загальний рівень задоволеності роботою в системі.

UI — тобто зовнішній вигляд платформи, її графічний інтерфейс — є візуальною оболонкою, яка допомагає користувачу орієнтуватися в системі. Це включає кольорову гаму, шрифти, іконки, кнопки, форми, повідомлення — усе, що впливає на естетику та комфорт використання. Вдалий UI не лише приваблює візуально, а й полегшує сприйняття інформації, виділяє пріоритетні елементи, формує логічну послідовність дій. Збалансований, продуманий інтерфейс зменшує когнітивне навантаження, що є критично важливим для користувачів, які щоденно працюють із великою кількістю інформації.

Важливо також враховувати, що UX/UI безпосередньо впливає на швидкість виконання проєктних операцій. Чим менше часу витрачає користувач на навігацію системою, пошук необхідного інструменту чи внесення даних — тим вища загальна ефективність його роботи. Це, у свою чергу, скорочує часові витрати всієї команди, знижує ризики помилок і прискорює досягнення цілей проєкту. Крім того, якісний дизайн інтерфейсу дозволяє адаптувати систему під мобільні пристрої, що значно розширює можливості управління проектами з будь-якої точки світу.

Загалом, взаємозв’язок між архітектурними рішеннями та дизайном UX/UI є визначальним у формуванні ефективної платформи для управління проектами. Вони не лише забезпечують технічну надійність і зручність у користуванні, а й створюють умови для гнучкого, адаптивного та комфортного середовища командної роботи. У довгостроковій перспективі саме поєднання стабільної архітектури та продуманого дизайну визначає успішність цифрових інструментів у сфері управління проектами.

1.4. Огляд аналогічних програмних розробок

У рамках дослідження було розглянуто низку популярних веб-додатків для управління проектами, які вже довели свою ефективність у сфері організації командної роботи, контролю прогресу виконання завдань та забезпечення

внутрішньої комунікації. Аналіз цих рішень дозволяє виявити ключові тенденції у розробці подібних систем, визначити найбільш затребувані функції та сформувати обґрунтовані вимоги до власного продукту.

Серед найбільш відомих конкурентів на ринку варто виокремити такі рішення:

- ClickUp — платформа, що забезпечує багаторівневу організацію завдань, підтримує різні подання (списки, дошки, діаграми Ганта), має широкі можливості для автоматизації процесів, інтеграції зі сторонніми сервісами та налаштування прав доступу.
- Asana — сервіс, орієнтований на візуалізацію задач та прозоре управління командами. Пропонує зручні інструменти для планування, спільної роботи, контролю термінів та генерації звітів, що робить його придатним як для невеликих, так і для середніх команд.
- Trello — застосунок, заснований на методології Kanban. Дозволяє організовувати завдання у вигляді карток, які можна переміщати між колонками, що відповідають етапам виконання. Простота інтерфейсу та можливість швидкого налаштування робочого простору сприяють його популярності.
- Jira — рішення, орієнтоване переважно на команди розробників. Підтримує складну структуру завдань, інтеграцію з системами контролю версій, гнучке налаштування робочих процесів, а також можливість використання Scrum та Kanban-дошок.
- Zoho Projects — система, що поєднує управління завданнями з інструментами для комунікації, тайм-трекінгу, побудови діаграм Ганта та контролю ресурсів. Має адаптивний інтерфейс та інтегрується з іншими продуктами екосистеми Zoho.

Порівняльний аналіз зазначених систем дозволив виділити як базовий, так і додатковий функціонал, який користувачі вважають важливим для ефективної роботи. Ці результати стали підґрунтям для формування функціональних вимог до розроблюваного веб-додатку, з урахуванням реальних потреб потенційної аудиторії.

Функціонал	ClickUp	Asana	Trello	Jira	Zoho Projects
Управління завданнями	✓	✓	✓	✓	✓
Канбан-дошка	✓	✓	✓	✓	✓
Тайм-трекінг	✓	✗	✗	✓	✓
Календар	✓	✓	✓	✗	✓
Спільна робота	✓	✓	✓	✓	✓
Гант-діаграма	✓	✓	✗	✗	✓
Інтеграції з іншими сервісами	✓	✓	✓	✓	✓
Автоматизація процесів	✓	✓	✗	✓	✓
API для розширень	✓	✓	✓	✓	✓
Вбудований чат	✓	✗	✗	✗	✓

Рисунок 1.1 – Таблиця порівняння аналогічних програмних продуктів

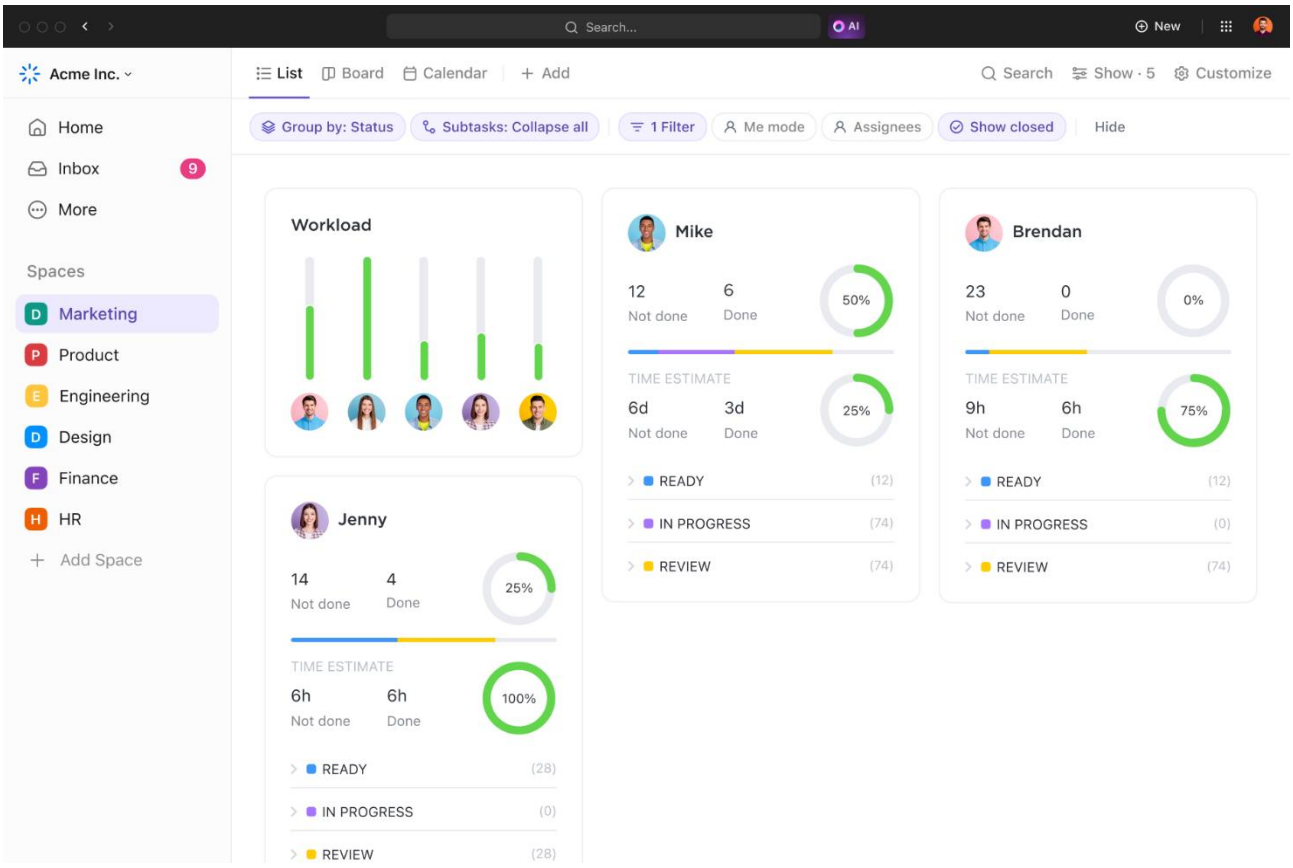


Рисунок 1.2 – Інтерфейс «ClickUP»

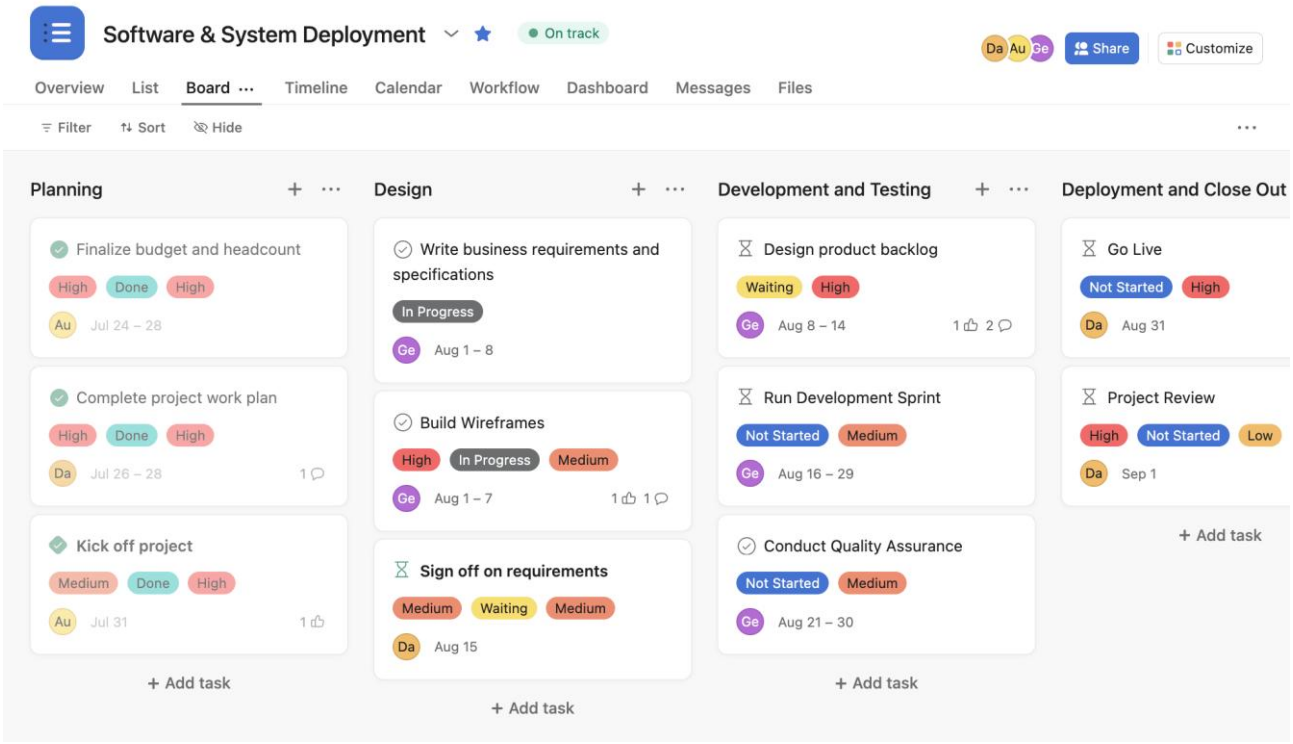


Рисунок 1.3 – Інтерфейс «Asana»

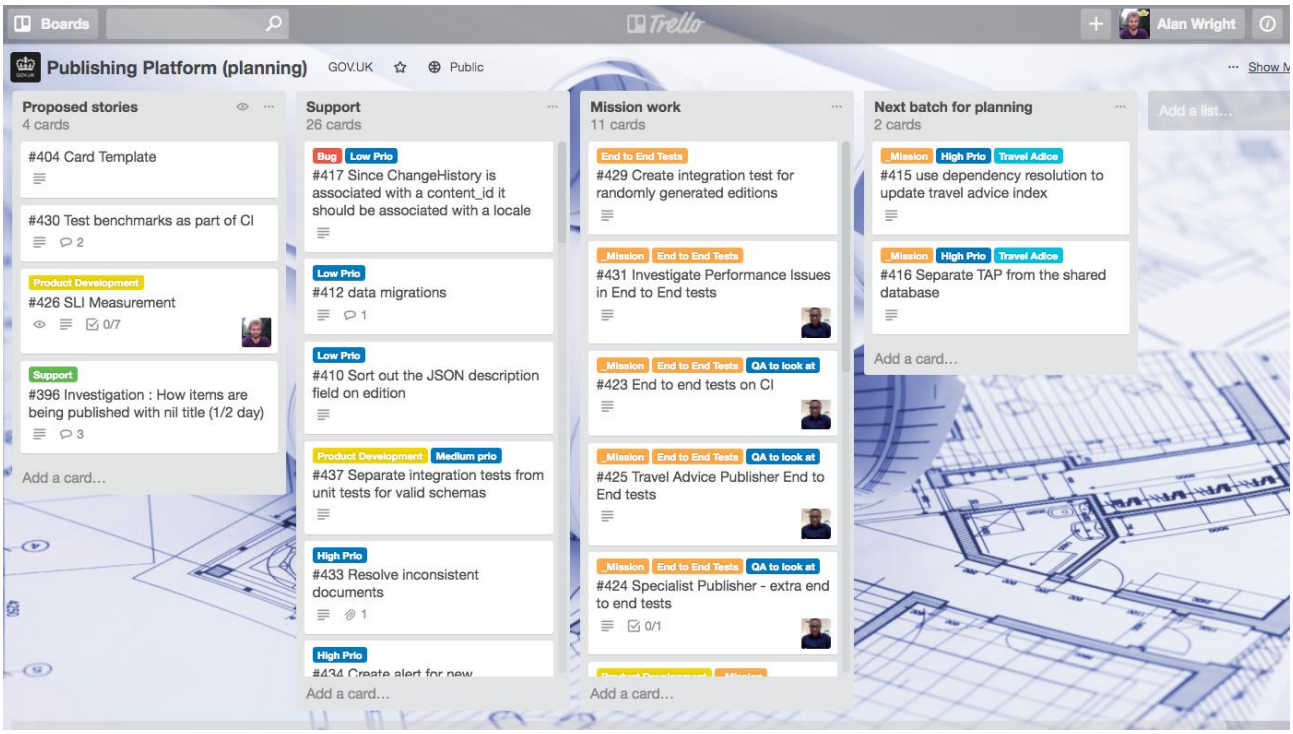


Рисунок 1.4 – Інтерфейс «Trello»

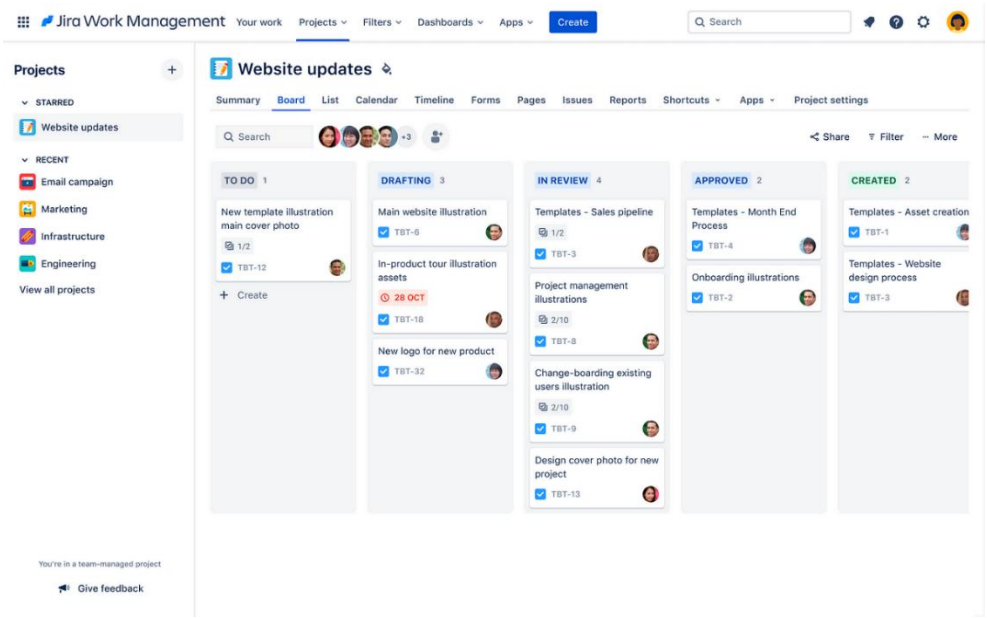


Рисунок 1.5 – Інтерфейс «Jira»

РОЗДІЛ 2. РОЗРОБКА ВЕБ-ДОДАТКУ ДЛЯ УПРАВЛІННЯ ПРОЕКТАМИ НА ОСНОВІ NODE.JS

2.1. Постановка задачі, призначення та вимоги до веб-додатку для управління проектами

З розвитком інформаційних технологій усе більше сфер діяльності переходить у цифрове середовище. Особливо це помітно в управлінні проектами, де від ефективної взаємодії між учасниками залежить успіх реалізації завдань. У таких умовах стає необхідним використання інструментів, які дозволяють централізовано організувати роботу над проектами, контролювати терміни виконання, розподіляти обов’язки та оперативно реагувати на зміни.

Метою цієї роботи є розробка веб-додатку для управління проектами, який забезпечуватиме базовий набір функцій для планування, моніторингу та координації дій команди. Рішення має бути простим у використанні, логічно структурованим та здатним працювати у реальному часі.

Додаток повинен надавати користувачеві можливість:

- створювати облікові записи та входити до системи через авторизацію;
- ініціювати нові проекти, задавати їм назву, опис, учасників;
- формувати завдання в межах проекту з параметрами: назва, виконавець, термін виконання, статус;
- редагувати та видаляти як проекти, так і окремі завдання;
- переглядати список проектів і завдань з фільтрацією за статусами;
- відстежувати прогрес виконання проекту в загальному вигляді.

Для реалізації цього рішення планується використання середовища виконання Node.js, що дозволяє створювати швидкі й масштабовані серверні застосунки. У парі з Express.js ця технологія забезпечить обробку запитів та зручну побудову маршрутів. Зберігання даних відбуватиметься у базі даних MongoDB, а інтерфейс буде реалізований за допомогою стандартних веб-технологій — HTML, CSS та JavaScript.

Розроблений веб-додаток дозволить автоматизувати ключові процеси управління проектами та слугуватиме практичним інструментом для командної роботи, особливо в умовах віддаленого доступу чи гнучкого графіку.

2.2. Вибір моделі розробки архітектури та структури веб-додатку

Під час розробки веб-додатку для управління проектами було прийнято рішення реалізувати архітектуру за принципом поділу на клієнтську (frontend) та серверну (backend) частини. Такий підхід дозволяє досягти гнучкості, масштабованості та зручності супроводу системи.

Серверна частина побудована з використанням Node.js та Express.js — легкої, мінімалістичної платформи для створення RESTful API. Express.js забезпечує необхідний інструментарій для обробки запитів, маршрутизації, взаємодії з базою даних та реалізації логіки авторизації й аутентифікації.

Клієнтська частина реалізована з використанням бібліотеки React. Завдяки компонентному підходу React забезпечує високу модульність, повторне використання елементів інтерфейсу, а також ефективне оновлення даних за допомогою віртуального DOM. Такий підхід особливо ефективний у проектах, де користувацький інтерфейс повинен бути динамічним та швидко реагувати на зміни стану.

У якості архітектурної моделі було обрано підхід REST-клієнт/сервер. Клієнт звертається до API, отримує та надсилає дані у форматі JSON. Взаємодія з базою даних (MongoDB) реалізується через ORM-бібліотеку Mongoose, яка забезпечує зручне моделювання даних і валідацію на рівні схеми.

Вся система умовно поділена на такі логічні модулі:

- клієнтська частина (React): авторизація/реєстрація, перегляд проектів, створення задач, редагування статусів, взаємодія з командою;
- серверна частина (Node.js/Express): обробка HTTP-запитів, логіка управління проектами та задачами, захист доступу;
- база даних (MongoDB): зберігання інформації про користувачів, проєкти, задачі, коментарі;

- служби взаємодії: система сповіщень, логування подій, middleware-функції для обробки помилок.

Такий підхід дозволяє масштабувати систему, легко впроваджувати нові функції, забезпечувати безпеку й продуктивність, а також підтримувати зручну взаємодію між клієнтською частиною та сервером.

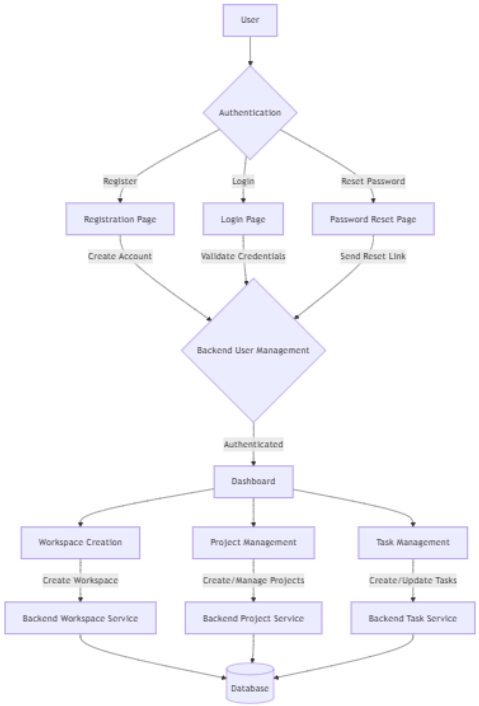


Рисунок 2.1 - Схема проєкту

2.3. Особливості програмної реалізації

Для забезпечення контролю доступу до функціональності веб-додатку реалізовано розділення маршрутів на відкриті та захищені. Перегляд більшості сторінок можливий лише після успішної авторизації, тоді як деякі маршрути доступні всім користувачам.

Логіка маршрутизації зосереджена в компоненті AppRoutes який визначає, які сторінки має бачити користувач залежно від його статусу (авторизований або ні). В межах цієї логіки застосовано два окремі компоненти-контейнери:

- PublicRoute використовується для маршрутів, відкритих лише для неавторизованих користувачів — таких як реєстрація, вхід у систему,

відновлення пароля. У разі, якщо користувач уже залогінений, його автоматично перенаправляє до головної сторінки кабінету (/dashboard);

- PrivateRoute відповідає за маршрути, доступні лише після входу в систему — наприклад, перегляд проектів або робочих просторів. Якщо користувач не авторизований, при спробі доступу до таких сторінок його буде перенаправлено до форми входу.

Маршрут /logout призначений для завершення сесії користувача: при переході на цю адресу викликається функція logout, яка очищує дані авторизації, після чого користувач автоматично перенаправляється на сторінку входу /login.

Наявна система маршрутизації чітко розмежовує рівні доступу до різних розділів веб-додатку, забезпечуючи комфортну навігацію між сторінками та контроль за використанням ключових функцій — зокрема управління проєктами й завданнями. Такий підхід дозволяє запобігти несанкціонованому доступу до критично важливої інформації та підтримувати логіку взаємодії відповідно до статусу користувача.

З метою захисту API і контролю за автентичністю запитів у додатку реалізовано механізм JWT (JSON Web Token). Це рішення забезпечує надійний рівень безпеки, дозволяючи сервісу обслуговувати лише авторизовані запити. Основні функціональні елементи цієї реалізації включають:

- генерація токена реалізується функцією generateToken, яка перевіряє правильність секретного ключа (secret_key) і створює унікальний токен із вбудованими даними. Токен має обмежений термін дії та підписується приватним ключем, що підвищує стійкість до злому;

- перевірка токена здійснюється через middleware-функцію authenticateToken, яка перевіряє присутність і валідність токена в заголовках запиту. У разі відсутності або помилки автентифікації доступ до ресурсу блокується. Якщо ж токен підтверджений, інформація про користувача додається до запиту для подальшої обробки [9];

Для початку роботи з веб-додатком користувач має пройти процедуру реєстрації. Цей процес передбачає введення особистих даних, створення пароля

та його підтвердження. Інтерфейс реєстраційної форми реалізований на стороні клієнта за допомогою компонента Register (рис. 2.3), що відповідає за відображення форми (рис. 2.2) та обробку введених даних.

Стан компонента управляється за допомогою React-хука useState, що дозволяє зберігати введені значення (електронна пошта, ім’я, прізвище, нікнейм, пароль) та повідомлення про успіх чи помилки під час заповнення форми.

Форма включає поля для введення основної інформації користувача, а також функціонал для перевірки правильності введених даних. На етапі валідації здійснюється базова перевірка — зокрема, відповідність формату електронної пошти, збіг паролів і перевірка обов’язкових полів. У разі виявлення помилок система відображає відповідне повідомлення.

Після проходження валідації дані надсилаються на сервер у вигляді POST-запиту. У заголовках передається авторизаційний токен, якщо це необхідно. У разі успішної реєстрації користувач автоматично авторизується в системі, а через короткий проміжок часу перенаправляється на сторінку входу. У випадку помилки користувачу виводиться повідомлення про причину збою.

Такий підхід до реалізації реєстраційного функціоналу дозволяє забезпечити зручну взаємодію з додатком та гарантує базовий рівень безпеки при створенні облікового запису.

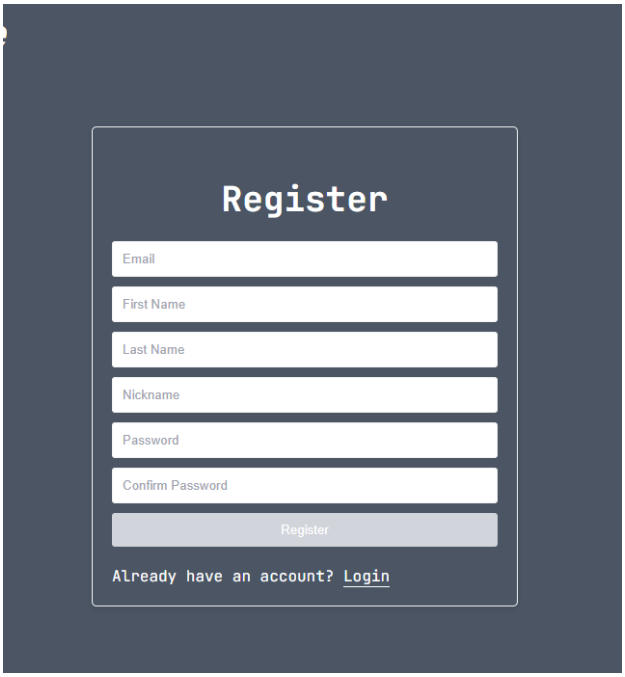


Рисунок 2.2 - Форма реєстрації

```
Register.jsx
3 usages  new *
const Register = () => {
    const { login } = useContext(AuthContext);
    const [email : string , setEmail] = useState( initialState: '');
    const [firstName : string , setFirstName] = useState( initialState: '');
    const [lastName : string , setLastName] = useState( initialState: '');
    const [nickname : string , setNickname] = useState( initialState: '');
    const [password : string , setPassword] = useState( initialState: '');
    const [confirmPassword : string , setConfirmPassword] = useState( initialState: '');
    const [error : string , setError] = useState( initialState: '');
    const [success : string , setSuccess] = useState( initialState: '');
    const navigate : NavigateFunction  = useNavigate();

    1 usage  new *
    const handleSubmit = async (e) :Promise<void>  => {
        e.preventDefault();
        setError( value: '');
        setSuccess( value: '');

        const apiUrl : string  = `${process.env.REACT_APP_BACKEND_BASE_URL}/auth/register`;

        try {
            const token : any | undefined  = await generateToken();

            const response : AxiosResponse<any>  = await axios.post(apiUrl,  data: {
                email,
                firstName,
                lastName,
                nickname,
                password,
                confirmPassword,
            },  config: {
                headers: {
                    Authorization: `Bearer ${token}`
                }
            });

            if (response.status === 201) {
                const userData :{…}  = {
                    email,
                    token,
                    message: response.data.message
                };

                login(userData);
                setSuccess( value: 'Registration successful! Redirecting to login...');
                setTimeout( callback: () :void  => {
                    navigate('/login');
                },  ms: 2000);
            } else {
                setError( value: 'Registration failed. Please try again.');
            }
        } catch (err) {
            setError( value: err.response?.data.message || 'Something went wrong!');
        }
    };
```

Рисунок 2.3 - Функціональна частина компоненту Register

На бекенді створено модель User для роботи з даними користувачів. Вона містить такі основні поля:

- email – унікальна електронна адреса;
- password – хешований пароль;
- firstName / lastName – ім’я та прізвище;
- nickname – опціональний нікнейм;
- resetCode / resetCodeExpiration – поля для відновлення пароля.
- isAdmin – булеве значення, що визначає, чи користувач є адміністратором (за замовчуванням false);
- hourlyRate – погодинна ставка користувача (числове значення, за замовчуванням 0);

- profileImage – шлях або URL до зображення профілю користувача (рядок, за замовчуванням порожній);

- penalty – штраф у доларах США (числове значення, за замовчуванням 0).

Модель використовується для взаємодії з базою даних MongoDB.

На стороні сервера реалізовано функціонал, який забезпечує обробку реєстраційних запитів та безпечне зберігання облікових даних користувачів. З цією метою було створено низку функцій:

- hashPassword — відповідає за хешування пароля перед його збереженням у базі даних. Використання криптографічних хеш-функцій дозволяє запобігти компрометації паролів у разі витоку даних;

- comparePassword — виконує порівняння введеного користувачем пароля з відповідним хешем, збереженим у базі даних, що є необхідним під час автентифікації;

- register (рис. 2.4) — основна функція обробки реєстраційного запиту. Вона здійснює перевірку коректності вхідних даних, зокрема унікальності електронної адреси, хешує пароль користувача, створює новий обліковий запис та зберігає його у базі даних.

Такий підхід до реалізації реєстраційного процесу дозволяє забезпечити надійний рівень безпеки та цілісність збережених даних.

```
// User registration function
1 usage  new *
const register = async (req, res) : Promise<...>  => {
    const { email, password, confirmPassword, firstName, lastName, nickname } = req.body;

    // Validate required fields
    if (!email || !password || !confirmPassword || !firstName || !lastName) {
        return res.status(400).json({ message: 'All fields are required' });
    }

    // Validate password confirmation
    if (password !== confirmPassword) {
        return res.status(400).json({ message: 'Passwords do not match' });
    }

    // Validate password length
    if (password.length < 8) {
        return res.status(400).json({ message: 'Password must be at least 8 characters long' });
    }

    try {
        // Check if the user already exists
        const existingUser = await User.findOne({ email });
        if (existingUser) {
            return res.status(400).json({ message: 'User already exists' });
        }

        // Hash the password and save the new user
        const hashedPassword = await hashPassword(password);
        const newUser : HydratedDocument<InferSchemaType  = new User( doc: {
            email,
            password: hashedPassword,
            firstName,
            lastName,
            nickname,
        });
        await newUser.save();

        return res.status(201).json({ message: 'User created successfully' });
    } catch (error) {
        console.error('Registration error:', error);
        return res.status(500).json({ message: 'Internal server error', error: error.message });
    }
};
```

Рисунок 2.4 - Функція register

У разі наявності облікового запису користувач може здійснити вхід до системи за допомогою форми авторизації (рис. 2.5).

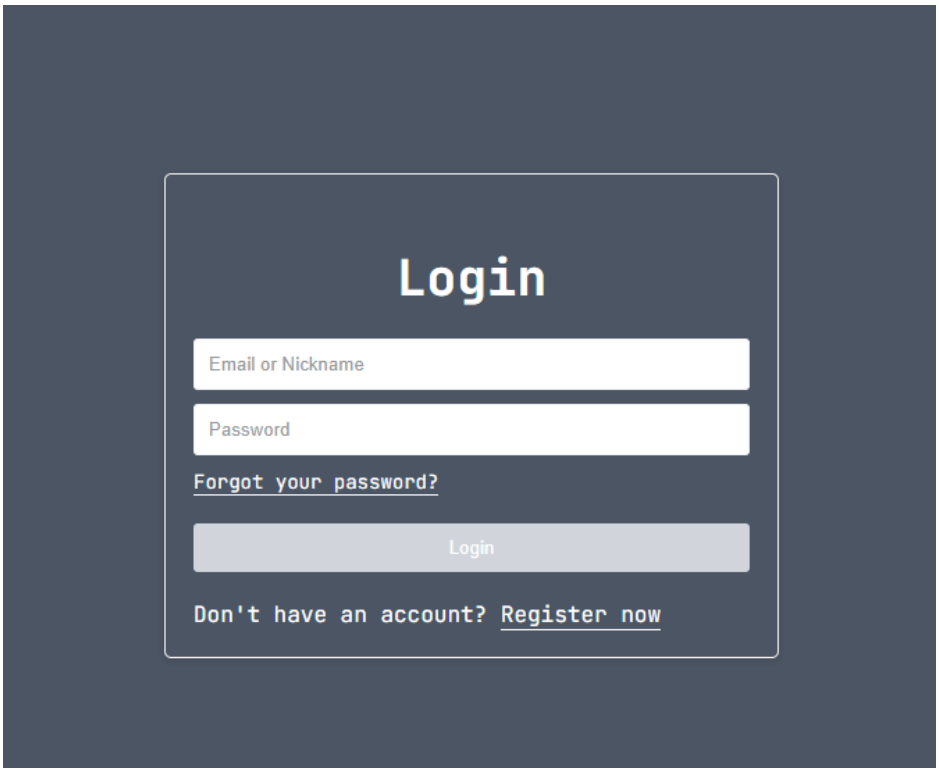


Рисунок 2.5 - Форма авторизації користувача

На стороні клієнта за її відображення та функціональність відповідає компонент Login (рис. 2.6), що реалізує наступну логіку:

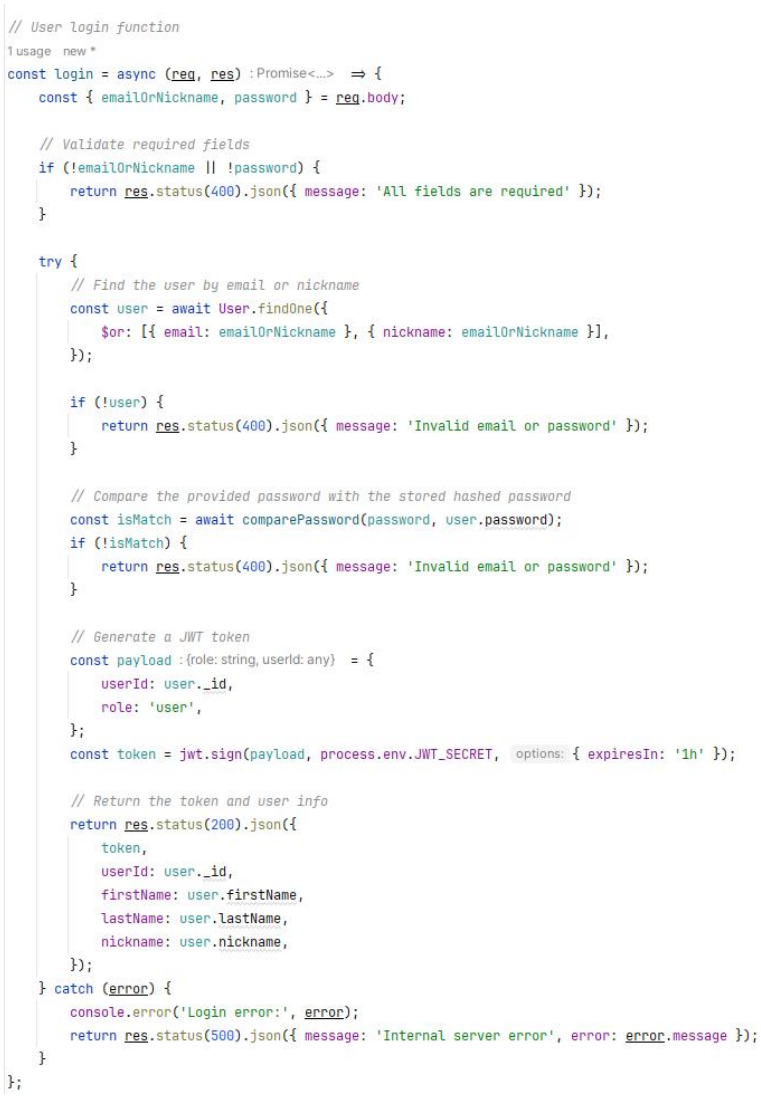
- надсилає запит до серверної частини для перевірки наданих облікових даних (електронної пошти або нікнейму та пароля);
- у разі успішної автентифікації зберігає ідентифікатор користувача та його нікнейм у контексті авторизації для подальшого використання;
- у разі помилки в облікових даних відображає відповідне повідомлення для користувача;
- після успішного входу здійснює автоматичне перенаправлення на головну сторінку додатку.

```
3 usages  new *
const Login = () => {
    const { login } = useContext(AuthContext);
    const [emailOrNickname : string , setEmailOrNickname] = useState( initialState: '');
    const [password : string , setPassword] = useState( initialState: '');
    const [error : string , setError] = useState( initialState: '');
    const [success : string , setSuccess] = useState( initialState: '');
    const navigate : NavigateFunction  = useNavigate();

    1 usage  new *
    const handleSubmit = async (e) : Promise<void>  => {
        e.preventDefault();
        setError( value: '');
        setSuccess( value: '');

        const apiUrl : string  = `${process.env.REACT_APP_BACKEND_BASE_URL}/auth/login`;

        try {
            const token : any | undefined  = await generateToken();
            const response : AxiosResponse<any>  = await axios.post(apiUrl,  data: {
                emailOrNickname,
                password,
            },  config: {
                headers: {
                    Authorization: `Bearer ${token}`
                }
            });

            if (response.status === 200) {
                const userData = response.data.user || response.data;

                // Save user ID and nickname in local storage
                if (userData.id) {
                    localStorage.setItem('userId', userData.id);
                    localStorage.setItem('nickname', userData.nickname || emailOrNickname);
                }

                login(userData); // Update context with user data
                setSuccess( value: 'Login successful! Redirecting to dashboard...');
                setTimeout( callback: () : void  => {
                    navigate('/dashboard');
                },  ms: 2000);
            }
        } catch (err) {
            setError( value: err.response?.data.message || 'Invalid credentials');
        }
    };
```

Рисунок 2.6 - Функціональна частина компоненту Login

На серверній стороні процес авторизації реалізовано за допомогою функції login (рис. 2.7), яка відповідає за обробку запитів на вхід до системи. Її основна логіка включає такі етапи:

- перевірка наявності обов’язкових полів emailOrNickname та password у тілі запиту;
- пошук користувача у базі даних за введеним email або нікнеймом;
- перевірка правильності введеного пароля шляхом порівняння збереженого хешу за допомогою функції comparePassword;
- у разі невідповідності або відсутності користувача повертається помилка з кодом 400;
- при виникненні внутрішніх помилок функція генерує відповідь зі статусом 500 та повідомленням про збій.



```
// User login function
1 usage  new *
const login = async (req, res) : Promise<...>  => {
    const { emailOrNickname, password } = req.body;

    // Validate required fields
    if (!emailOrNickname || !password) {
        return res.status(400).json({ message: 'All fields are required' });
    }

    try {
        // Find the user by email or nickname
        const user = await User.findOne({
            $or: [{ email: emailOrNickname }, { nickname: emailOrNickname }],
        });

        if (!user) {
            return res.status(400).json({ message: 'Invalid email or password' });
        }

        // Compare the provided password with the stored hashed password
        const isMatch = await comparePassword(password, user.password);
        if (!isMatch) {
            return res.status(400).json({ message: 'Invalid email or password' });
        }

        // Generate a JWT token
        const payload : {role: string, userId: any}  = {
            userId: user._id,
            role: 'user',
        };
        const token = jwt.sign(payload, process.env.JWT_SECRET,  options: { expiresIn: '1h' });

        // Return the token and user info
        return res.status(200).json({
            token,
            userId: user._id,
            firstName: user.firstName,
            lastName: user.lastName,
            nickname: user.nickname,
        });
    } catch (error) {
        console.error('Login error:', error);
        return res.status(500).json({ message: 'Internal server error', error: error.message });
    }
};
```

Рисунок 2.7 - Функція login

Після успішної авторизації користувач перенаправляється на головну сторінку — Dashboard, яка виступає стартовою точкою для подальшої роботи в системі.

Одним із перших елементів, що зустрічає користувача, є динамічне привітання. Його зміст змінюється залежно від поточного часу доби, у який здійснено вхід, що реалізовано через функцію getGreeting. Такий підхід створює персоналізований досвід взаємодії з додатком.

Крім того, на цій сторінці розміщений окремий блок із підказками та цікавими фактами на IT-тематику. Контент у цьому блоці обирається випадковим чином з попередньо сформованого масиву даних, що дозволяє забезпечити

різноманітність. Оновлення відбувається автоматично кожні 60 хвилин, завдяки чому користувачі регулярно отримують нову й корисну інформацію.

У хедері сайту розташоване навігаційне меню, за допомогою якого користувач може перейти на сторінку Workspaces (рис. 2.8).

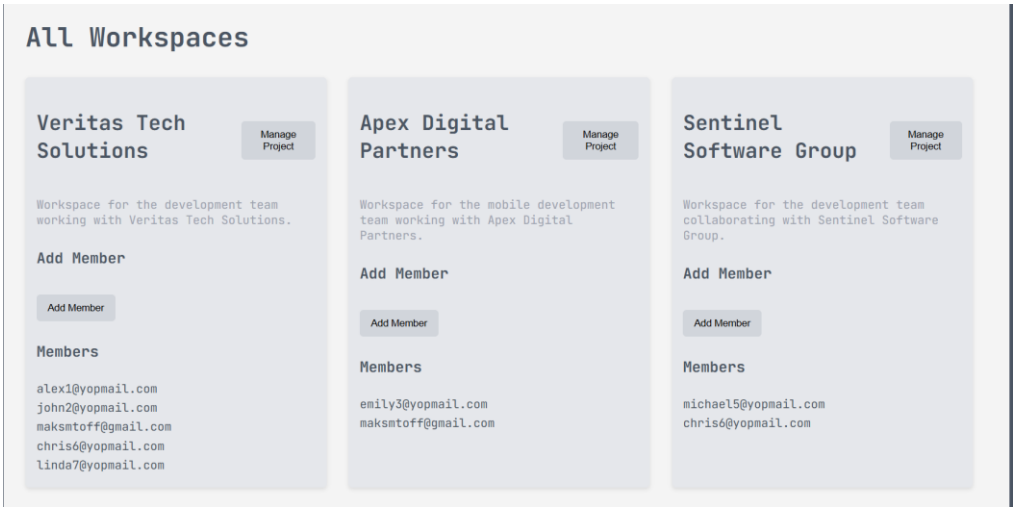


Рисунок 2.8 – Сторінка Workspaces

На цій сторінці реалізовано можливість створення нового робочого простору. Після натискання на кнопку CreateWorkspace відкривається модальне вікно, що містить форму для введення назви та опису воркспейсу.

У вікні також передбачено кнопки для підтвердження створення та закриття модального вікна. Після заповнення форми та її надсилання виконується функція створення нового воркспейсу, яка:

- надсилає POST-запит до сервера з введеними даними (назвою та описом) у тілі запиту та токеном авторизації в заголовку;
- у разі успіху додає новий воркспейс до поточного списку користувача;
- виводить повідомлення про успішне створення;
- у випадку помилки – відображає відповідне повідомлення з поясненням.

На серверній стороні за створення нового робочого простору відповідає функція createWorkspace (рис. 2.9). Її логіка реалізована в кілька основних етапів:

- отримання даних: з тіла запиту зчитуються значення назви та опису, введені користувачем у формі;

- створення об'єкта: за допомогою моделі Workspace створюється новий екземпляр, у якому фіксуються передані назва та опис;

- встановлення власника: ID авторизованого користувача, що зберігається у req.user.userId, призначається як власник воркспейсу;

- збереження в базі: за допомогою методу save() новий об’єкт воркспейсу записується до бази даних;

- відповідь клієнту: у разі успішного створення сервер надсилає відповідь зі статусом 201 та даними новоствореного воркспейсу;

- обробка винятків: якщо під час виконання виникає помилка, сервер повертає статус 500 та відповідне повідомлення про помилку.

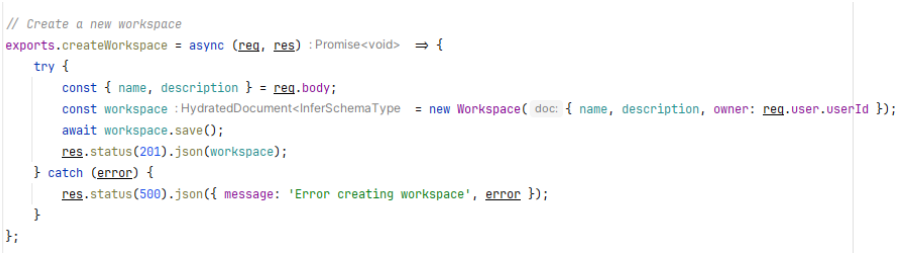


Рисунок 2.9 - Функція createWorkspace на бекенді

Після створення новий воркспейс одразу додається до загального списку доступних користувачу. Кожен елемент списку містить назву та опис відповідного воркспейсу, кнопку для запрошення нових учасників, а також перелік вже доданих членів команди.

Для додавання нового учасника до воркспейсу користувач натискає кнопку «Add member». У результаті відкривається модальне вікно, де необхідно ввести електронну адресу користувача, якого потрібно запросити. Після введення система перевіряє наявність такого користувача в базі. Якщо обліковий запис знайдено, з’являється можливість підтвердити додавання цієї особи до поточного воркспейсу.

Після того як користувач підтверджує додавання нового учасника, активується функція handleAddMember (рис. 2.10), яка виконує низку важливих дій. На першому етапі вона перевіряє, чи була обрана електронна адреса

користувача для додавання. Якщо адреса відсутня, користувач одразу бачить повідомлення про помилку, і подальше виконання функції припиняється.

Якщо ж усі дані вказані правильно, функція формує POST-запит до сервера. У тілі запиту передається унікальний ідентифікатор поточного воркспейсу (workspaceId) і електронна адреса нового учасника. Запит супроводжується заголовком із токеном автентифікації, що гарантує безпечний доступ до ресурсів.

Після отримання успішної відповіді від сервера дані нового учасника (зазвичай це електронна пошта або об’єкт користувача) додаються до масиву поточних мемберів. Це призводить до миттєвого оновлення списку учасників у інтерфейсі користувача. Крім того, система відображає повідомлення про успішне завершення операції.

У разі виникнення будь-якої помилки — наприклад, користувач із введеною адресою не знайдений, або сервер недоступний — на екрані з’являється відповідне повідомлення про помилку, щоб користувач міг вжити необхідних заходів.

```
1 usage  new *
const handleAddMember = async (workspaceId) : Promise<void>  => {
    const userEmail = selectedUserEmail[workspaceId];
    if (!userEmail) {
        setError( value: 'No user selected');
        return;
    }

    try {
        const token : any | undefined  = await generateToken();
        const response : AxiosResponse<any>  = await axios.post( url: `${process.env.REACT_APP_BACKEND_BASE_URL}/workspaces/${workspaceId}/add-member`,  data: {
            email: userEmail
        },  config: {
            headers: {
                Authorization: `Bearer ${token}`
            }
        });

        setMembers( value: (prev : {} ) => {
            const currentMembers = prev[workspaceId] || [];
            const newMember : {email: any}  = { email: userEmail };
            return { ...prev, [workspaceId]: [...currentMembers, newMember] };
        });

        setAddMemberSuccess( value: (prev : {} ) => ({ ...prev, [workspaceId]: 'User added successfully!' }));
        setSelectedUserEmail( value: (prev : {} ) => ({ ...prev, [workspaceId]: '' }));
        setError( value: '');
        setShowAddMemberPopup( value: (prev : {} ) => ({ ...prev, [workspaceId]: false }));
    } catch (err) {
        setError( value: 'Error adding member');
    }
};
```

Рисунок 2.10 - Функція handleAddMember

На серверній частині пошук учасників воркспейсу реалізовано у функції searchWorkspaceMembers. Ця функція виконує послідовний набір дій для обробки запиту.

На початку вона отримує необхідні параметри: workspaceId витягується з параметрів маршруту (req.params), а пошуковий запит q — з рядка запиту (req.query). Після цього система звертається до бази даних, щоб знайти воркспейс із відповідним ID. Під час пошуку також виконується попереднє заповнення (populate) списку учасників, що дає змогу одразу отримати додаткову інформацію про кожного користувача, зокрема електронну пошту та нікнейм.

Якщо воркспейс за заданим ID не існує, функція повертає помилку з кодом 404 та відповідне повідомлення. У випадку, коли воркспейс знайдено, відбувається фільтрація його учасників. Для цього функція перевіряє, чи починається електронна адреса або нікнейм користувача з введеного запиту, причому пошук виконується незалежно від регістру (метод .toLowerCase()).

Після відбору релевантних результатів кожного знайденого користувача функція трансформує у зручний для фронтенду формат: включає ID користувача, ID члена воркспейсу, нікнейм та електронну адресу. Якщо пошук пройшов успішно, сервер повертає цей список зі статусом 200 (OK).

У разі виникнення будь-якої помилки в процесі пошуку сервер повертає статус 500 та повідомлення про внутрішню помилку.

На бекенді функцію додавання нового учасника до воркспейсу реалізує метод addMemberToWorkspace (рис. 2.11). Уся логіка побудована навколо послідовної перевірки, пошуку і модифікації даних.

На початку обробляється запит, з якого витягується електронна адреса користувача через req.body.email. За цією адресою виконується пошук користувача в базі даних за допомогою User.findOne({ email }). Якщо користувача не знайдено, функція одразу завершує роботу з повідомленням про помилку (статус 404).

Далі відбувається пошук воркспейсу, до якого необхідно додати учасника. Якщо такого воркспейсу не існує, також повертається помилка зі статусом 404.

Після підтвердження існування користувача і воркспейсу функція перевіряє, чи вже є цей користувач учасником. Це реалізовано за допомогою методу some(), що проходить по масиву members і порівнює userId. Якщо користувач вже є в списку, повертається помилка зі статусом 400.

У разі, якщо користувача ще не додано, він включається до списку членів воркспейсу шляхом додавання нового об’єкта { userId, email } до масиву members. Після цього зміни зберігаються в базі даних через await workspace.save().

Якщо всі кроки проходять успішно, сервер надсилає відповідь зі статусом 200 та оновленим об'єктом воркспейсу. У випадку помилки будь-якого типу на будь-якому етапі (наприклад, база даних недоступна або некоректний запит) клієнт отримує статус 500 із відповідним повідомленням про помилку.

```
// Adding a user to the workspace
exports.addMemberToWorkspace = async (req, res) :Promise<...>  => {
    const { email } = req.body;

    try {
        const user = await User.findOne({ email });
        if (!user) {
            return res.status(404).json({ message: 'User not found' });
        }

        const workspace = await Workspace.findById(req.params.workspaceId);
        if (!workspace) {
            return res.status(404).json({ message: 'Workspace not found' });
        }

        const isMember = workspace.members.some(member => member.userId.equals(user._id));
        if (isMember) {
            return res.status(400).json({ message: 'User is already a member of the workspace' });
        }

        workspace.members.push({ userId: user._id, email: user.email });
        await workspace.save();

        res.status(200).json({ message: 'User added to workspace successfully', workspace });
    } catch (error) {
        console.error('Error adding user to workspace:', error);
        res.status(500).json({ message: 'Error adding user to workspace', error: error.message });
    }
};
```

Рисунок 2.11 - Функція addMemberToWorkspace

Після переходу по кнопці ManageProjects в елементі воркспейсу, користувач потрапляє на сторінку проєктів вибраного воркспейсу (рис. 2.12). На цій сторінці відображаються:

- кнопка створення проєкту – дозволяє користувачеві створювати новий проєкт у межах вибраного воркспейсу;

- список проєктів – містить перелік усіх проєктів, що належать цьому воркспейсу, з можливістю перегляду і керування ними.

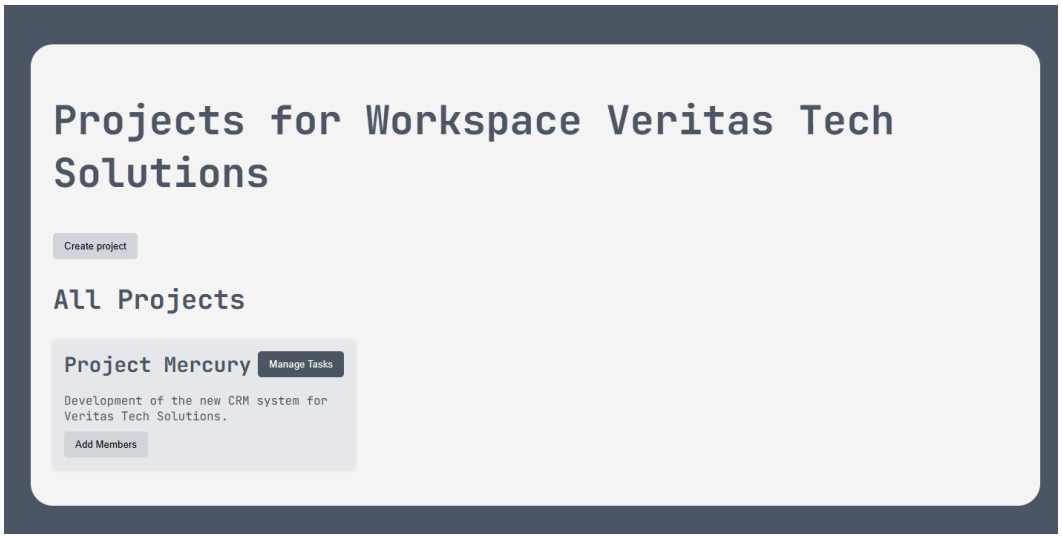


Рисунок 2.12 - Сторінка Projects

Після натискання на створити користувач побачить вікно з можливістю ввести назву та опис проєкту та створити його.

Для створення проєкту на фронтенді створена функція, яка збирає дані з форми у вікні та відправляє їх на бекенд.

На бекенді було реалізовано функцію createProject (рис. 2.13), яка збирає дані з фронтенду, валідує їх та створює проєкт у разі успішної валідації. Після успішного створення, проєкт з’являється у списку на сайті.

```
// Create Project
exports.createProject = async (req, res) : Promise<...>  => {
    const { name, description } = req.body;
    const { workspaceId } = req.params;
    try {
        if (!name || !description) {
            return res.status(400).json({ error: 'Name and description are required' });
        }

        const project : 0 extends (1 & module:mongoose...  = await Project.create({ name, description, workspaceId });
        res.status(201).json(project);
    } catch (error) {
        res.status(400).json({ error: error.message });
    }
};
```

Рисунок 2.13 – Функція createProject

У проєкті можна натиснути AddMember і побачити вікно додавання учасників в проект. Пошук відбувається по учасниках воркспейсу в якому створений проєкт.

При переході по ManageTasks, користувач попаде на сторінку задач проекту (рис. 2.14). На сторінці можна побачити кнопку Add New Task та 5 колонок, в якій фільтруються задачі по статусах.

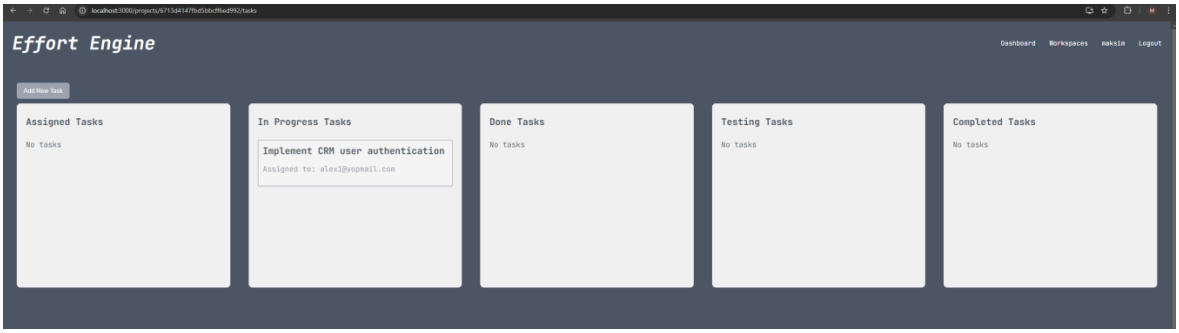


Рисунок 2.14 - Сторінка задач проєкту

При натисканні кнопки "Add new task" відкривається спливаюче вікно (рис. 2.15), яке дозволяє швидко і зручно створити нове завдання. У вікні потрібно вказати:

- назву завдання, яка стисло передає суть роботи;
- опис, що містить деталі та додаткові інструкції;
- виконавця, відповідального за виконання;
- дедлайн, до якого завдання має бути завершене.

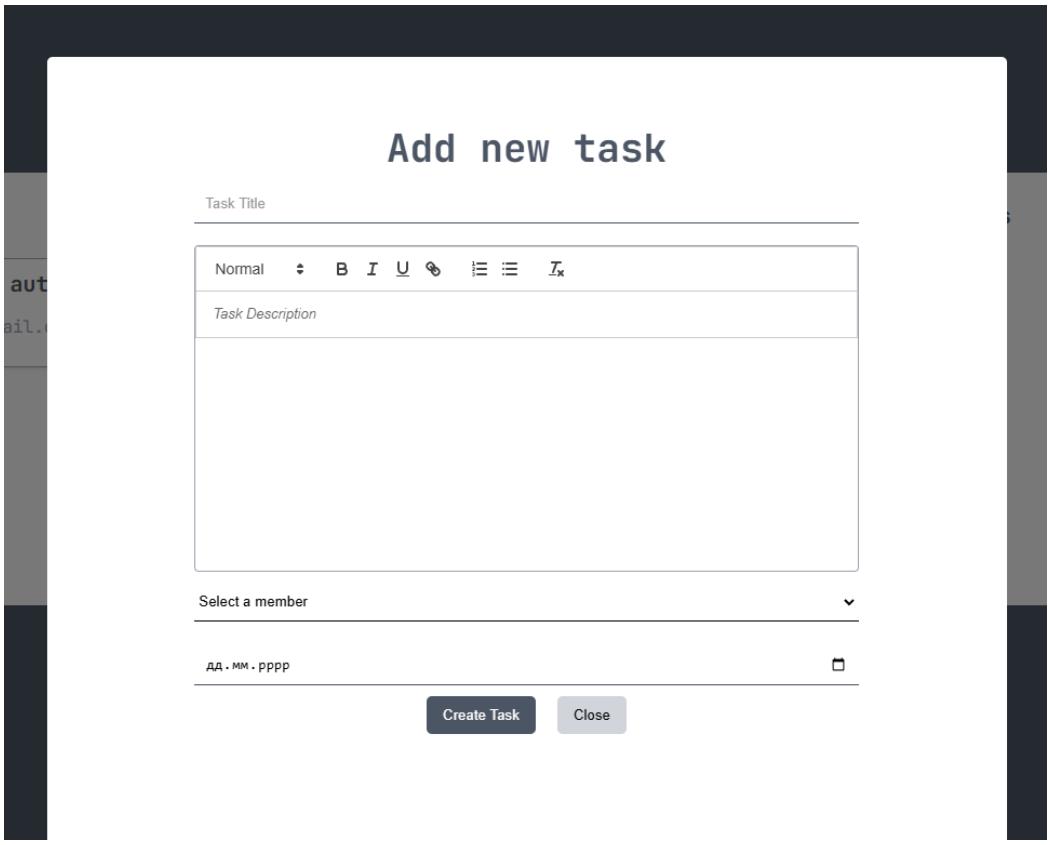


Рисунок 2.15 - Вікно створення задачі

Під час відправлення форми на фронтенді викликається функція handleSubmit. Вона збирає введені користувачем дані та надсилає запит до бекенду для створення задачі на їх основі.

На бекенді реалізована функція createTask (рис. 2.16), яка відповідає за створення нових задач. Вона:

- перевіряє вхідні дані: усі обов'язкові поля повинні бути передані. У разі відсутності одного з них функція повертає помилку зі статусом 400 і відповідним повідомленням;

- створює нову задачу: на основі отриманих даних створює новий екземпляр моделі Task. із базовим статусом assigned, якщо статус не зазначено, і зберігає його у базі даних.

```
1 usage  new *
const createTask = async (req, res) :Promise<...>  => {
    try {
        const { projectId, title, description, assignedTo, dueDate, status } = req.body;

        // Ensure all required fields are provided
        if (!projectId || !title || !description || !assignedTo || !dueDate) {
            return res.status(400).json({ message: 'All fields are required: projectId, title, description, assignedTo, and dueDate.' });
        }

        // Optional: Validate the status if included
        if (status && !['assigned', 'in progress', 'done', 'testing', 'completed'].includes(status)) {
            return res.status(400).json({ message: 'Status must be one of: assigned, in progress, done, testing, completed.' });
        }

        const newTask :HydratedDocument<InferSchemaType  = new Task( doc: {
            projectId,
            title,
            description,
            assignedTo,
            dueDate,
            status: status || 'assigned' });
        await newTask.save();
        res.status(201).json(newTask);
    } catch (error) {
        console.error(error);
        res.status(500).json({ message: 'Server error' });
    }
};
```

Рисунок 2.16 - Функція createTask

Після успішного виконання функція повертає створену задачу зі статусом 201 або повідомляє помилку, якщо щось пішло не так.

На фронтенді задачі після успішного створення автоматично відображаються у відповідній колонці зі статусом Assigned. Інтерфейс управління списком задач реалізований у форматі drag-and-drop (рис. 2.17), що забезпечує зручне та інтуїтивне пересування задач між колонками статусів.

Drag-and-drop функціонал забезпечується компонентом <DragDropContext>, який відстежує дії користувача при переміщенні задач. Кожна колонка представлена компонентом TaskColumn, який отримує:

- droppableId – унікальний ідентифікатор колонки для визначення статусу задач;
- tasks – список задач, пов’язаних із цим статусом;
- title – назву колонки для візуального відображення;
- projectId – ідентифікатор поточного проєкту.

```
: (
  <DragDropContext onDragEnd={onDragEnd}>
      <div className="tasks-columns">
          <TaskColumn droppableId="assigned" tasks={tasks.assigned} title="Assigned Tasks" projectId={projectId} />
          <TaskColumn droppableId="inProgress" tasks={tasks.inProgress} title="In Progress Tasks" projectId={projectId} />
          <TaskColumn droppableId="done" tasks={tasks.done} title="Done Tasks" projectId={projectId} />
          <TaskColumn droppableId="testing" tasks={tasks.testing} title="Testing Tasks" projectId={projectId} />
          <TaskColumn droppableId="completed" tasks={tasks.completed} title="Completed Tasks" projectId={projectId} />
      </div>
  </DragDropContext>
```

Рисунок 2.17 - Реалізація drag-and-drop на фронтенді

При зміні розташування задачі між колонками викликається функція onDragEnd, яка:

- визначає, з якої колонки і в яку колонку переміщено задачу;
- автоматично відправляє запит на бекенд для оновлення статусу задачі;
- оновлює локальний стан задач у реальному часі, забезпечуючи плавний користувацький досвід.

Цей підхід дозволяє ефективно керувати статусами задач у межах проєкту, мінімізуючи затримки і забезпечуючи інтерактивний та зручний інтерфейс.

При натисканні на задачу користувач побачить модальне вікно(рис. 2.44), яке відображає детальну інформацію про вибрану задачу. У цьому вікні доступні наступні функції:

- детальний опис задачі - відображається розширений опис задачі, який містить всю необхідну інформацію про її зміст, цілі та умови виконання. Це дозволяє користувачу чітко зрозуміти, що потрібно зробити;
- виконавець задачі - у модальному вікні зазначається, хто є відповідальним за виконання задачі. Якщо потрібно, користувач має можливість змінити виконавця за допомогою функції "Reassigntask". Це забезпечує гнучкість у розподілі роботи між учасниками команди;
- коментарі до задачі - реалізовано функціонал для додавання коментарів. Користувачі можуть залишати свої думки, уточнення або обговорення щодо задачі. Це створює зручний канал комунікації між членами команди, пов’язаними з виконанням задачі.

Модальне вікно (рис. 2.18) є інтуїтивно зрозумілим і функціональним, що дозволяє користувачам ефективно управляти задачами та спілкуватися в процесі їх виконання.

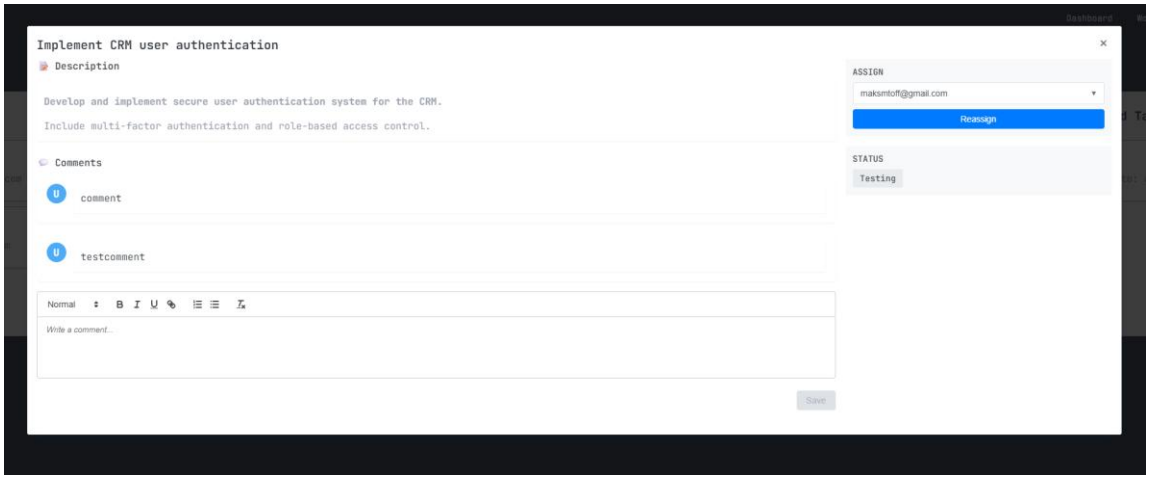
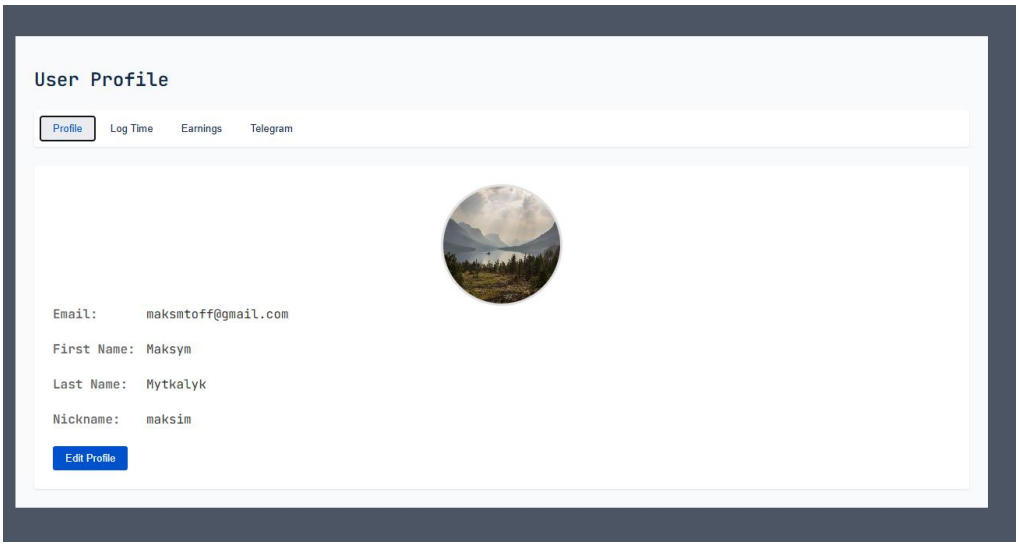
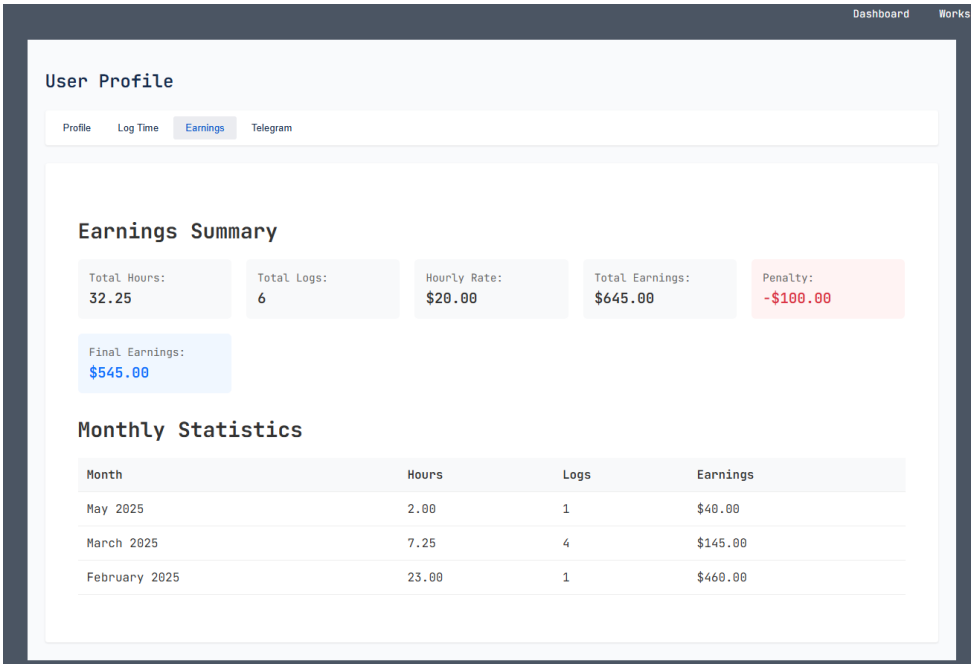


Рисунок 2.18 - Модальне вікно з детальною інформацією про задачу

Також через навігаційне меню у хедері можна потрапити до профілю користувача (рис. 2.19). У веб-додатку він відіграє важливу роль у взаємодії з проектами та командою. Окрім базових налаштувань, таких як особисті дані, користувачі мають можливість ефективно відстежувати свій робочий час, працюючи над конкретними задачами, а також побачити кількість зароблених коштів, що надає прозорість у фінансовій частині їхньої діяльності.





Рисунки 2.19 - Вигляд профілю користувача

Крім того, для покращення комунікації та моніторингу проєктів, користувач може приєднати свій профіль до бота (рис. 2.20), що дозволяє отримувати актуальні оновлення та взаємодіяти з командою через месенджер.

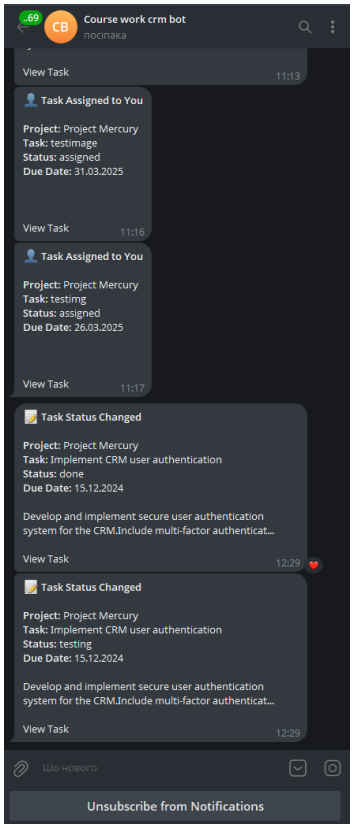


Рисунок 2.20 - Вигляд бота

Для керування всім аспектами системи була реалізована за допомогою Admin.js панель адміністратора(рис. 2.21). Вона забезпечує адміністраторам можливість безперешкодно управляти користувачами, проектами, задачами та іншими важливими елементами, що дозволяє ефективно контролювати роботу додатку.

Однією з основних функцій панелі є управління користувачами. Адміністратор має можливість переглядати, редагувати та видаляти профілі користувачів, а також додавати нових учасників до проектів. Це дозволяє централізовано керувати ролями та доступами в системі, а також ефективно організовувати робочі процеси.

Управління проектами та задачами є іншою важливою складовою частиною панелі адміністратора. Завдяки цьому інструменту адміністратор може переглядати та редагувати деталі проектів, додавати нові задачі, призначати відповідальних виконавців та встановлювати статуси задач. Це дає змогу ефективно керувати проєктами в реальному часі, а також підтримувати гнучкість у процесі роботи над завданнями.

Адміністратори також мають доступ до фінансової інформації, що дозволяє відстежувати зароблені кошти та витрати, пов’язані з конкретними проектами. Це важливо для контролю бюджету і забезпечення фінансової ефективності проектів. Інтерфейс панелі адміністратора зручний для використання, що дозволяє легко працювати з фінансовими даними без необхідності в складних налаштуваннях..

Завдяки простому інтерфейсу та широким можливостям для адміністрування, реалізованим через Admin.js, панель адміністратора дозволяє користувачам, навіть без технічних знань, ефективно управляти проектами та задачами. Це забезпечує високу продуктивність і спрощує організацію робочих процесів у межах веб-додатку.

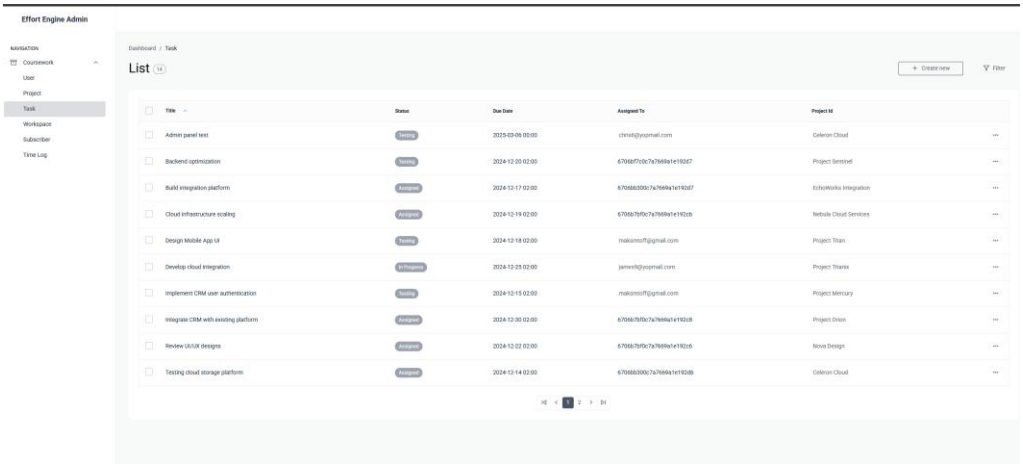


Рисунок 2.21 - Вигляд адміністративної панелі

2.4. Тестування та налагодження веб додатку для управління проектами на основі Node.js

Після завершення розробки веб-додатку для управління проектами на платформі Node.js важливим етапом є тестування та налагодження системи. Це допомагає виявити помилки та забезпечити правильну роботу компонентів додатку.

Юніт-тести використовуються для перевірки окремих функцій та методів серверної частини додатку. Вони дозволяють перевірити логіку на рівні окремих функцій. Для цього застосовуються Mocha, Chai та Sinon.

Інтеграційні тести перевіряють взаємодію між компонентами системи, наприклад, перевірку бізнес-логіки додатку чи взаємодію з базою даних. Вони допомагають перевірити, чи додається новий користувач до проекту чи відображаються всі завдання для конкретного проекту.

Тести на продуктивність оцінюють, як додаток справляється з навантаженням. Використовується Artillery для моделювання запитів від багатьох користувачів одночасно.

Кінцеві тести (End-to-End**)** проводяться для перевірки роботи всього додатку, від авторизації користувача до виконання задач. Для цього використовуються інструменти, такі як Cypress.

Логування помилок є важливим етапом у процесі налагодження. Використовуються Winston для логування помилок та повідомлень, а також Morgan для запису HTTP запитів.

Налагодження через консоль допомагає відслідковувати виконання коду. Використовуються стандартні методи для виведення змінних, а також можливості Node.js debugger та VSCode debugger для покрокового налагодження.

Тестування API проводиться за допомогою інструментів, таких як Postman та Insomnia, для перевірки запитів та аналізу відповідей від сервера.

2.4. Рекомендації з використання та впровадження веб-додатку в робочі процеси

Для ефективного впровадження веб-додатку в робочі процеси компанії важливо забезпечити навчання користувачів, чітке визначення ролей та прав доступу, а також інтеграцію з іншими інструментами, що використовуються в організації. Потрібно провести навчання команди щодо основних функцій додатку, таких як створення проектів, додавання завдань, зміна статусів та управління виконавцями. Важливо також інтегрувати додаток з іншими інструментами, такими як електронна пошта, календар та системи зберігання файлів, щоб покращити зручність роботи.

Наступний крок — визначити ролі користувачів, забезпечивши доступ до функцій залежно від їхніх обов'язків. Адміністратори мають повний доступ, менеджери проектів можуть управляти завданнями та командою, а звичайні користувачі працюють з конкретними завданнями, які їм призначені.

Паралельно потрібно налаштувати робочі процеси в команді, враховуючи етапи проектів, моніторинг виконання завдань і звітність. Це допоможе забезпечити структуру в управлінні проектами. Не забувайте про безпеку даних, впроваджуючи шифрування та регулярні оновлення системи.

Для досягнення ефективності впровадження необхідно постійно моніторити ефективність використання додатку, отримувати відгуки користувачів і вдосконалювати систему на основі цих даних.

2.5. Загальна характеристика реалізованої системи

Загалом, можна зробити висновок, що при створенні розробленої системи управління проектами було свідомо зроблено спробу уникнути типових недоліків, характерних для громіздких програмних продуктів великих розробників. Насамперед мова йде про надмірну складність, дублювання функціональності, зайву бюрократичність і перевантаження інтерфейсу. Натомість у межах цієї роботи реалізовано концепцію цільового мінімалізму — залишено лише ті функції, які дійсно необхідні для ефективного управління завданнями, контролю термінів і участі команди.

Однією з ключових інновацій стала побудова авторського інтерфейсу монетизації вкладу користувача в режимі реального часу. Це дозволяє не лише прозоро відстежувати прогрес окремих учасників проєкту, а й відображати їхню мотивацію у фінансових показниках, що може бути корисним як у внутрішніх командах, так і в умовах фриланс- або аутсорсинг-проєктів. Такий підхід дозволяє забезпечити справедливу винагороду, засновану на фактичному внеску кожного, що вигідно вирізняє розроблену систему серед більшості існуючих аналогів.

Окрім базової функціональності керування завданнями, проєктом також враховано специфіку реального процесу розробки та супроводу IT-продуктів — зокрема, впроваджено механізм умовних штрафів за порушення термінів виконання задач. Це дозволяє з одного боку — дисциплінувати учасників команди, з іншого — об’єктивно оцінювати динаміку та ефективність роботи над проєктом.

У результаті реалізовано 7 з 10 функцій, характерних для типового програмного забезпечення для управління проєктами, при цьому значно знижено рівень надлишковості. Такий баланс між функціональністю та простотою дозволяє запропонованій системі бути придатною для швидкої інтеграції в робочі процеси команд будь-якого масштабу, зберігаючи при цьому адаптивність і зручність користування.

Функціонал	ClickUp	Asana	Trello	Jira	Zoho Projects	Effort Engine
Управління завданнями	✓	✓	✓	✓	✓	✓
Канбан-дошка	✓	✓	✓	✓	✓	✓
Тайм-трекінг	✓	✗	✗	✓	✓	✓
Календар	✓	✓	✓	✗	✓	✗
Спільна робота	✓	✓	✓	✓	✓	✓
Гант-діаграма	✓	✓	✗	✗	✓	✗
Інтеграції з іншими сервісами	✓	✓	✓	✓	✓	✓
Автоматизація процесів	✓	✓	✗	✓	✓	✓
API для розширень	✓	✓	✓	✓	✓	✓
Вбудований чат	✓	✗	✗	✗	✓	✗

Рисунок 2.22 - Співставлення функціоналу веб-додатку з альтернативними рішеннями

ВИСНОВКИ

Під час розробки вебдодатку для управління проєктами та задачами було використано зв’язку Node.js для серверної частини та React для клієнтського інтерфейсу. Така технологічна комбінація дозволила створити сучасний, продуктивний та зручний інструмент для командної роботи, що відповідає актуальним стандартам веброзробки й забезпечує стабільну взаємодію з користувачем.

Node.js забезпечує обробку запитів у неблокуючому режимі, що сприяє високій швидкодії бекенду, тоді як React надає змогу реалізувати гнучкий, динамічний інтерфейс із чіткою логікою представлення даних. Компонентна структура React значно спростила побудову UI і прискорила розробку.

Серед основних функціональних можливостей додатку — створення та редагування проєктів, керування задачами, призначення виконавців і контроль їхнього виконання. Впроваджено систему автентифікації та розмежування прав доступу, що дозволяє гнучко управляти ролями користувачів.

Обрана архітектура забезпечує гнучке масштабування, дозволяючи швидко впроваджувати нові можливості або інтеграції без суттєвого впливу на основну логіку. Такий підхід не лише підвищує ефективність роботи з великими обсягами даних, а й сприяє кращій адаптації системи до потреб користувачів, забезпечуючи інтуїтивне та комфортне середовище для керування проєктами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. HowtoBuild a TaskManagementApp [2024 Guide]. Freshcode: IT ConsultingandSoftwareDevelopmentCompany. URL: https://www.freshcodeit.com/blog/how-to-create-task-management-app-mvp (дата звернення: 23.01.2025).
2. Index | Node.js v23.3.0 Documentation. Node.js – RunJavaScriptEverywhere. URL: https://nodejs.org/en/docs/ (дата звернення: 24.01.2025).
3. JetBrains. WebStorm: TheSmartestJavaScript IDE, byJetBrains. *JetBrains*. URL: https://www.jetbrains.com/webstorm/ (дата звернення: 24.01.2025).
4. GettingStarted – React. *React*. URL: https://reactjs.org/docs/getting-started.html (дата звернення: 25.01.2025).
5. Fullstackopen. *Fullstackopen*. URL: https://fullstackopen.com/ (дата звернення: 03.03.2025).
6. PowerPoint Templates, GraphicsandThemes - PPT Slides | SketchBubble. *PowerPoint Templates, GraphicsandThemes - PPT Slides | SketchBubble*. URL: https://www.sketchbubble.com/ (дата звернення: 05.03.2025).
7. Team M. D. MongoDBDocumentation. MongoDB: TheDeveloperDataPlatform | MongoDB. URL: https://www.mongodb.com/docs/ (дата звернення: 14.10.2024).
8. Express - Node.js webapplicationframework. Express - Node.js webapplicationframework. URL: https://expressjs.com/ (дата звернення: 05.03.2025).
9. JWT.IO - JSON WebTokensIntroduction. JSON WebTokens - jwt.io. URL: https://jwt.io/introduction/ (дата звернення: 06.03.2025).
10. Jira | Issue & Project Tracking Software | Atlassian. Collaboration software for software, IT and business teams. URL: https://www.atlassian.com/software/jira (дата звернення: 01.02.2025).

11.Manage Your Team’s Projects From Anywhere | Trello. Manage Your Team’s Projects From Anywhere | Trello. URL: https://trello.com/ (дата звернення: 01.02.2025).
12. Manage your team’s work, projects, & tasks online • Asana • Asana. Asana. URL: https://asana.com/ (дата звернення: 01.02.2025).
13. monday.com | Your go-to work platform. monday.com. URL: https://monday.com/ (дата звернення: 01.02.2025).
14. ClickUp. ClickUpâ□¢ | The everything app for work. URL: https://clickup.com/ (дата звернення: 01.02.2025).

ДОДАТКИ
ДОДАТОК А
ТЕХНІЧНЕ ЗАВДАННЯ

Тематика

Розробка системи для управління проектами та завданнями в командній роботі, з можливістю реєстрації, авторизації користувачів та управління доступом до ресурсів. Система має на меті покращити організацію робочих процесів через інтегровані інструменти для керування проектами та завданнями, а також забезпечити зручний інтерфейс для ефективної роботи команд.

Цільова аудиторія:

- компанії та організації, що працюють у командному режимі;
- стартапи та малі бізнеси;
- фрілансери та команди, що працюють на віддалену роботу;
- організації, що потребують ефективного управління проектами.

Платформа:

- клієнтська частина: React;
- серверна частина: Node.js / Express;
- база даних: MongoDB;
- середовище розробки: WebStorm або іншої IDE.

Основні функції:

- реєстрація та автентифікація користувачів;
- створення та управління воркспейсами;
- управління проектами: створення, редагування та видалення проектів;
- управління завданнями в рамках проектів;
- можливість призначати завдання учасникам і змінювати їх статус;
- інтерфейс для зручного управління завданнями через drag-and-drop;
- налаштування доступу до проектів та завдань;
- перегляд списку проектів та завдань з можливістю фільтрації.

Вимоги до безпеки

- надійна система автентифікації користувачів;
- забезпечення конфіденційності та цілісності даних;
- захист доступу до воркспейсів та проектів;
- система журналювання всіх операцій для аудиту.

Особливі вимоги

- мінімалістичний дизайн інтерфейсу з використанням нейтральних кольорів;
- інтуїтивно зрозумілий інтерфейс для швидкої навігації;
- адаптивний дизайн для роботи на різних пристроях;
- підтримка багатомовності, включаючи українську мову;
- оптимізація системи для роботи з великими обсягами даних;
- інтеграція з іншими популярними інструментами для командної роботи (наприклад, Slack або Trello).

ДОДАТОК Б
КЕРІВНИЦТВО КОРИСТУВАЧУ

Це керівництво призначене для надання користувачам інструкцій щодо використання вебдодатку для управління проєктами та завданнями EffortEngine. Воно містить інформацію про основні функції додатку, поради щодо його ефективного використання та вирішення типових проблем.

Реєстрація та вхід у систему:

Для створення нового профілю на сайті EffortEngine перейдіть на сторінку /register. У формі, яку ви побачите на сторінці, введіть необхідні дані для реєстрації та натисніть Register(рис. Б.1). Після цього буде створено профіль.

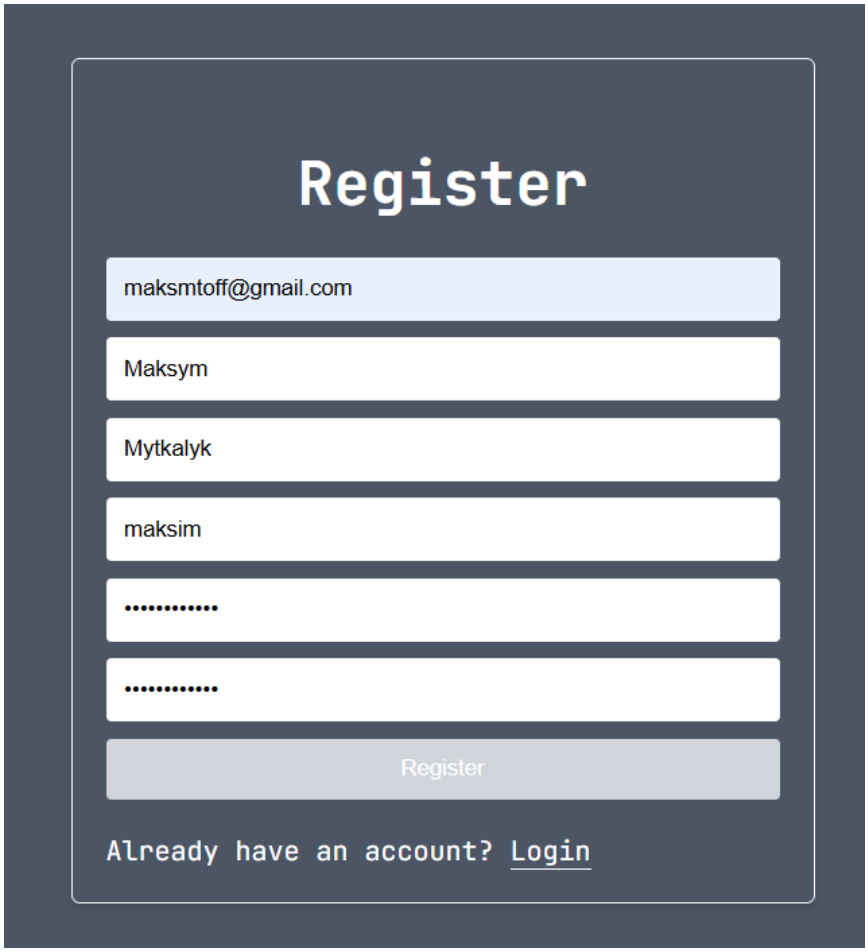


Рисунок Б.1- Процес реєстрації

.

Вхід у систему для існуючих користувачів:

- якщо у вас вже є обліковий запис, ви можете увійти в систему, відвідавши ж сторінку входу /login;
- введіть ваші дані для входу, включаючи електронну пошту(або нікнейм) та пароль, у відповідні поля;
- натисніть на кнопку "Login" для доступу до вашого облікового запису.

Відновлення пароля:

Якщо ви забули пароль ви можете легко відновити його скориставшись сторінкою /reset-password:

Спочатку введіть електронну пошту на яку маєте зареєстрований обліковий запис(рис. Б.2):

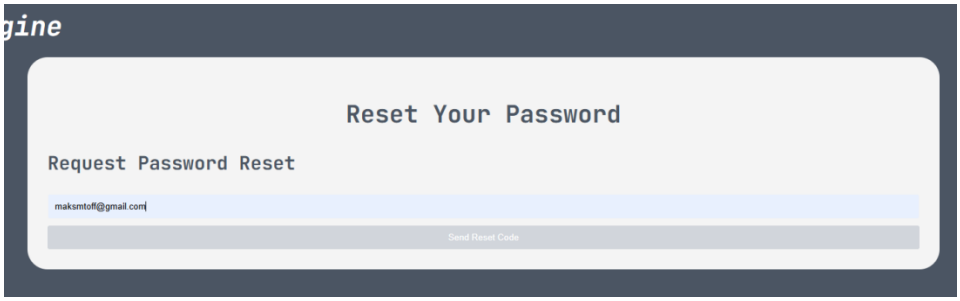


Рисунок Б.2- Перший крок відновлення пароля

Після натискання SendResetCode ви побачите другий крок, де потрібно буде ввести код, який прийде вам на електронну пошту:

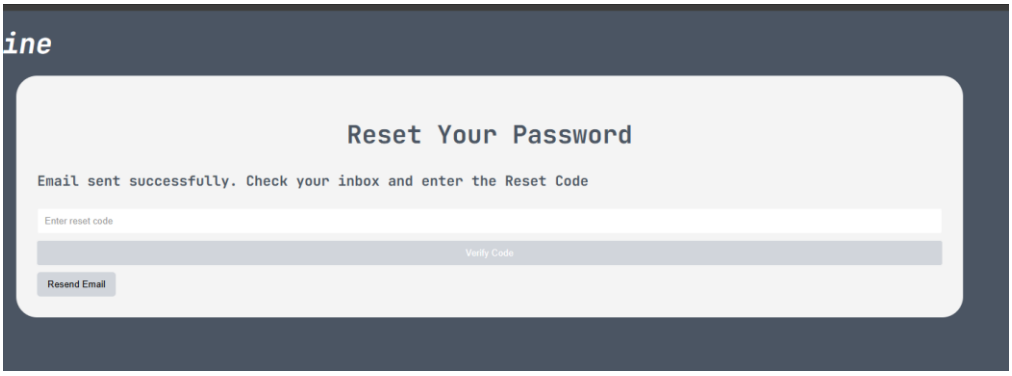


Рисунок Б.3- Введення коду з електронного листа

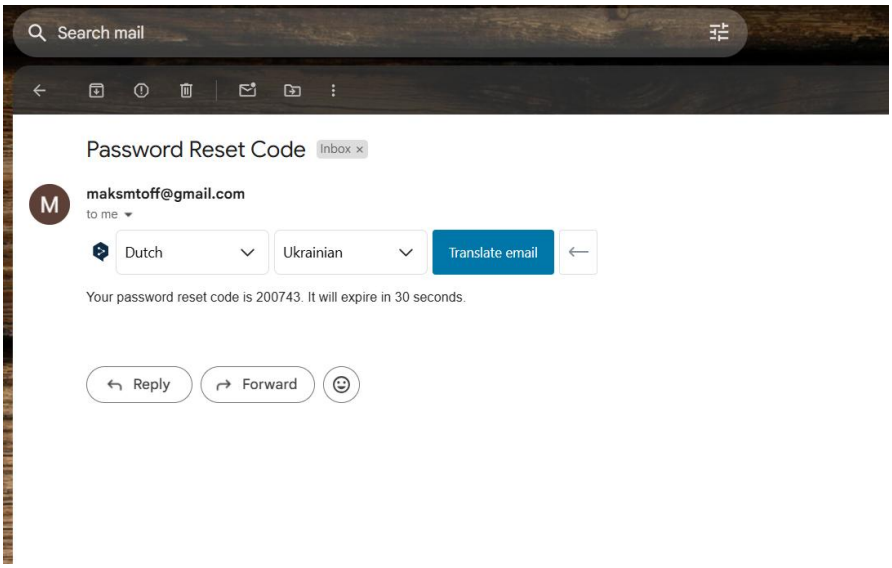


Рисунок Б.4- Електронний лист з кодом для скидання пароля

Після підтвердження електронної пошти за допомогою коду ви побачите форму для введення нового паролю(рис. Б.5):

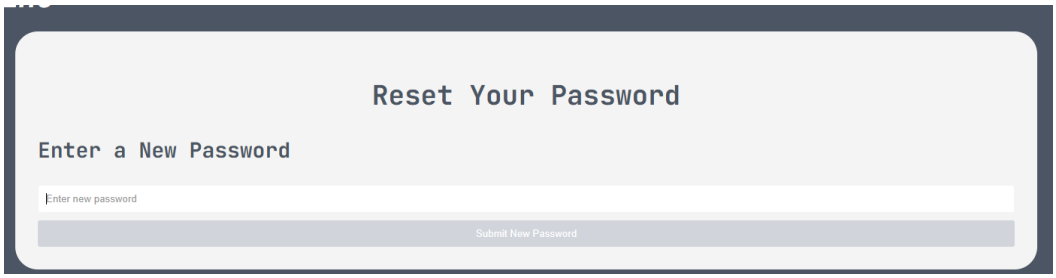


Рисунок Б.5- Форма встановлення нового пароля

Примітка:

Анонімні користувачі бачать лише сторінку реєстрації, логінації та скидання паролю. Для того, щоб отримати доступ до повного функціоналу сайту ви повинні бути авторизованим користувачем.

Керування воркспейсами, проєктами та задачами:

Перейдіть на сторінку /workspaces, де можете скористатиськнопкоюAddworkspaceта створити воркспейс(рис. Б.6) у вікні, яке відкриється по кліку.

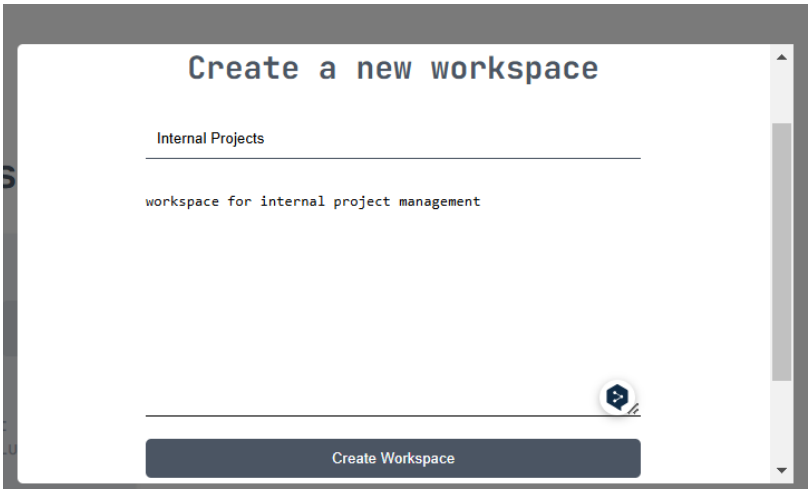


Рисунок Б.6-Процес створення воркспейсу

.

Щоб перейти до проєктів у потрібному воркспейсі, достатньо натиснути кнопку "ManageProjects". На сторінці проєктів ви можете не тільки створити новий проєкт, але й швидко перейти до вже існуючого. Кожен проєкт має свою дошку завдань, на якій ви зможете переглядати поточні завдання, призначені вам, або ж створити нову задачу, щоб доручити її іншому працівнику.

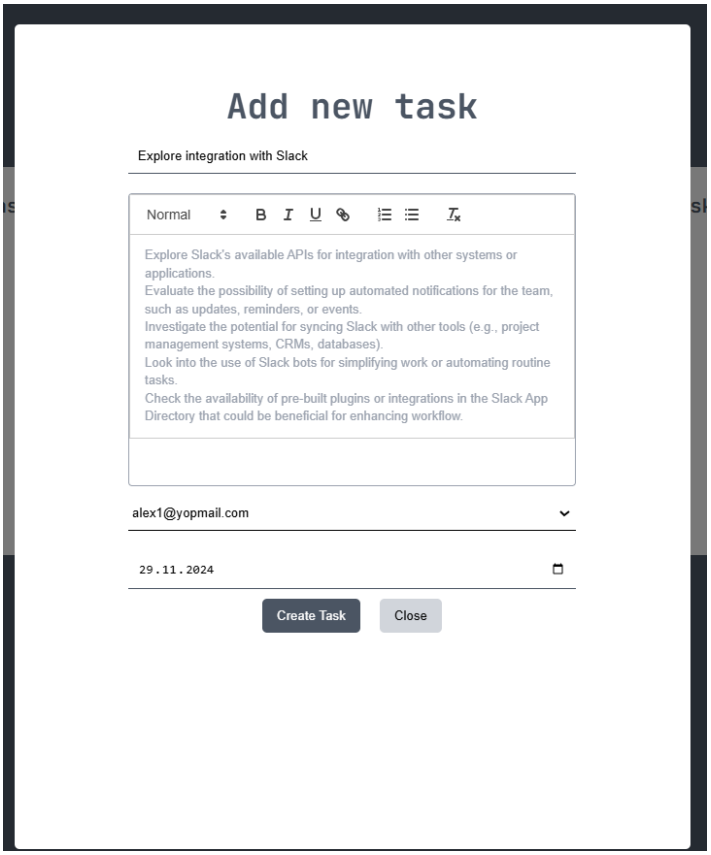


Рисунок Б.7– Процес створення задачі

Примітка:

Ви матимете доступ до перегляду воркспейсу та проєкту лише в тому випадку, якщо вони були створені вами або якщо ви є учасником цих проєктів.

АНОТАЦІЯ

Миткалик М.О. – **Розробка веб-додатку для управління проектами на Node.js**.

Кваліфікаційна робота за спеціальністю 122 Комп’ютерні науки. – Волинський національний університет імені Лесі Українки, Луцьк. – 2025р.

Робота присвячена розробці вебзастосунку для управління проєктами та задачами, який дозволяє ефективно організовувати командну роботу, координувати виконання завдань та забезпечувати прозору взаємодію між учасниками. Основна мета – створити сучасну платформу, що поєднує гнучкість, масштабованість і зручність у використанні. Серверна частина реалізована за допомогою Node.js, а клієнтський інтерфейс – на основі бібліотеки React. Серед основних функцій – реєстрація та автентифікація користувачів, створення проєктів, додавання задач, призначення виконавців, відстеження статусу завдань, а також інтеграція з ботом у месенджері. Для адміністрування системи використано бібліотеку AdminJS. Окрему увагу приділено користувацькому профілю, де реалізовано функціонал трекінгу робочого часу та підрахунку зароблених коштів. Робота охоплює повний цикл створення застосунку – від проєктування до тестування.

Ключові слова: вебзастосунок, управління проєктами, Node.js, React, бот, трекінг задач, авторизація, REST API, MongoDB, AdminJS.