Хмельницький національний університет
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення

# КВАЛІФІКАЦІЙНА РОБОТА

Мазура Дмитрія Ігоровича
Прізвище, ім`я, по батькові студента(ки)

на здобуття ступеня вищої освіти Бакалавра

Вебсистема для організації
Назва теми
та контролю виконання проєктних завдань

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

КвРІПЗ.2101078.01.07.ПЗ

Виконав студент IV курсу, група ІПЗ-21-1 _____ Дмитрій МАЗУР
Підпис Ім`я, ПРІЗВИЩЕ

Керівник канд. техн. наук, доцент _____ Юрій ФОРКУН
Науковий ступінь, звання Підпис Ім`я, ПРІЗВИЩЕ

Нормоконтролер канд. техн. наук, доцент _____ Оксана ЯШИНА
Підпис Ім`я, ПРІЗВИЩЕ

**До захисту допускаю:**
Завідувач кафедри інженерії
програмного забезпечення _____ Леонід Бедратюк
Підпис Ім`я, ПРІЗВИЩЕ

6 червня 2025 р.

Хмельницький 2025

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій
Кафедра Інженерії програмного забезпечення
Рівень вищої освіти Перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри ІПЗ
Л. П. Бедратюк
2.01.2025

# ЗАВДАННЯ
# НА КВАЛІФІКАЦІЙНУ РОБОТУ

Мазуру Дмитрію Ігоровичу
Прізвище, ім'я, по батькові студента

1. Тема кваліфікаційної роботи Вебсистема для організації та контролю виконання проєктних завдань
Керівник кваліфікаційної роботи Форкун Юрій Вікторович, канд. техн. наук, доцент
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання
Затверджена наказом ректора університету від 08.01.2024 р. № 6
2. Строк подання студентом роботи на кафедру 01.06.2025 р.
3. Вихідні дані до роботи Матеріали переддипломної практики
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
Дослідження предметної області та постановка задачі, проєктування вебсистеми. програмна реалізація та тестування
5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)
Презентаційні матеріали (слайди, 16 шт.), діаграма варіантів використання, діаграма архітектури патерну MVC, діаграма класів взаємодії модулів, ER-діаграма, блок-схема алгоритму створення завдання, діаграма послідовностей для процесу редагування завдання, діаграма послідовностей.
6. Консультанти розділів кваліфікаційної роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| Нормоконтроль | Яшина О. М., к.т.н., доцент | | |
| Антиплагіат | Форкун Ю.В., к.т.н., доцент | 12.05.2025 | 28.05.2025 |

7. Дата видачі завдання «2» 01 2025р.

# КАЛЕНДАРНИЙ ПЛАН

| Назва етапів (розділів) кваліфікаційної роботи | Строк виконання етапів роботи | Примітка |
|---|---|---|
| Ознайомлення з тематикою дипломного проєктування, визначення та узгодження індивідуальних тем кваліфікаційних робіт (КвР) | 01.12 – 31.12.2024 | |
| Збір матеріалу за темою КвР; дослідження предметної області, в якій планується використання програмного забезпечення (ПЗ), визначення задач та вимог, розробка технічного завдання | 01.01 – 20.02.2025 | |
| 3 Проєктування програмного забезпечення | 21.02 – 20.03.2025 | |
| Програмна реалізація з використанням відповідних засобів розробки. Тестування ПЗ | 21.03 – 30.04.2025 | |
| Написання вступу, загальних висновків, оформлення переліку джерел посилання та додатків. Оформлення пояснювальної записки КвР згідно вимог | 01.05 – 25.05.2025 | |
| 6 Попередній захист КвР | Травень 2025 | Згідно графіка |
| Перевірка КвР на плагіат, нормоконтроль, отримання відгуків, рецензій та інших супровідних документів. Брошурування (зшиття) пояснювальної записки. | 26.05 – 30.05.2025 | |
| Здача КвР на кафедру; підготовка КвР для розміщення у репозитарії ХНУ; підготовка до захисту та захист КвР | з 01.06.2025 | |

Студент _____ Мазур Д. І.
Підпис Ініціали, прізвище

Керівник роботи _____ Форкун Ю. В.
Підпис Ініціали, прізвище

# АНОТАЦІЯ

Тема кваліфікаційної роботи: «Вебсистема для організації та контролю виконання проєктних завдань».

Автор роботи: Мазур Дмитрій Ігорович.

Керівник роботи: Форкун Юрій Вікторович.

Пояснювальна записка: 125 с., 19 рис., 3 табл., 3 дод., 40 джерела.

Графічна частина: 16 слайдів.

ВЕБСИСТЕМА, УПРАВЛІННЯ ЗАВДАННЯМИ, REACT, EXPRESS.JS, NODE.JS, MONGODB, MVC

Метою роботи є розробка вебсистеми для організації та контролю виконання проєктних завдань, яка забезпечить зручне управління завданнями, прозорість робочих процесів і адаптивність для користувачів із різними доступами до завдань.

У кваліфікаційній роботі було проведено аналіз предметної області, досліджено сучасні підходи до управління проєктами, сформовано функціональні та нефункціональні вимоги, спроєктовано архітектуру системи на основі патерну MVC, обґрунтовано вибір стеку технологій (MERN), реалізовано основні модулі вебсистеми, зокрема авторизацію, управління завданнями та дошку Kanban.

Для реалізації клієнтської та серверної частини застосунку використано JavaScript-орієнтований MERN-стек. Для автентифікації користувачів застосовано JSON Web Token (JWT), а для інтерфейсу використано бібліотеку react-beautiful-dnd.

Запропонована вебсистема надає користувачам інтуїтивно зрозумілий інтерфейс, функціональність створення, редагування та перегляду завдань, можливість переміщення завдань між колонками для зміни статусу, фільтрацію за пріоритетом і статусом, а також розмежування доступу для користувачів до завдань.

Вебсистема може бути використана командами, які прагнуть ефективно організувати проєктну діяльність у єдиному цифровому середовищі.

28.05.2025
Дата

Підпис

# ВІДОМІСТЬ ДОКУМЕНТІВ

| № рядка | Формат | Позначення документа | Найменування документа | К-сть аркушів | № екз. | Примітка |
|---|---|---|---|---|---|---|
| | | | | | | |
| | | | Текстові документи | | | |
| | | | | | | |
| 1 | А4 | КвРІПЗ.2101078.01.07.ПЗ | Пояснювальна записка | 125 | | |
| 2 | А4 | | Завдання на кваліфікаційну роботу | 1 | | |
| 3 | А4 | | Анотація | 1 | | |
| | | | | | | |
| | | | Графічні документи | | | |
| | | | | | | |
| 4 | А4 | КвРІПЗ.2101078.01.07.Е8 | Логічна модель бази даних | 1 | | |
| 5 | А4 | КвРІПЗ.2101078.01.07.Е8 | Блок-схема алгоритму створення завдання | 1 | | |

| Змн. | Арк. | № докум. | Підпис | Дата | КвРІПЗ.2101078.01.07.ПЗ | | |
|---|---|---|---|---|---|---|---|
| Виконав | | Мазур Д. І. | | 28.05 | Вебсистема для організації та контролю виконання проєктних завдань<br>Пояснювальна записка | Літ. | Арк. 1 / Аркушів 1 |
| Керівник | | Форкун Ю. В. | | 28.05 | | | |
| Н. Контр. | | Яшина О. М. | | 28.05 | | ХНУ, ІПЗ-21-1 | |
| Зав. Каф. | | Бедратюк Л.П. | | 28.05 | | | |

## ЗМІСТ



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | КвРІПЗ.2101078.01.07.ПЗ | | |
| Змн. | Арк. | № докум. | Підпис | Дата | | | |
| Виконав | | Мазур Д. І. | | 28.05 | Вебсистема для організації та контролю виконання проєктних завдань | Літ. | Арк. 6 / Аркушів 72 |
| Керівник | | Форкун Ю. В. | | 28.05 | | | |
| Н. Контр. | | Яшина О. М. | | 28.05 | Пояснювальна записка | ХНУ, ІПЗ-21-1 | |
| Зав. Каф. | | Бедратюк Л.П. | | 28.05 | | | |

# ВСТУП

У сучасному світі, де технології дедалі швидше проникають у всі сфери людської діяльності, вебпростір відіграє ключову роль у формуванні нових підходів до організації роботи та взаємодії між людьми. Інтернет-технології не лише спрощують доступ до інформації, але й створюють умови для ефективного управління процесами, зокрема в рамках проєктної діяльності. У контексті глобалізації та цифровізації вебсистеми стають основою для інноваційних рішень, дозволяючи автоматизувати рутинні задачі, оптимізувати комунікацію та підвищувати продуктивність команд. Розвиток таких систем сприяє трансформації традиційних підходів до роботи, адаптуючи їх до потреб сучасного ринку, де швидкість і гнучкість є визначальними факторами успіху.

Організація та контроль проєктних завдань є однією з найактуальніших задач для компаній, які прагнуть досягти успіху в умовах динамічного ринкового середовища. Вебсистеми, призначені для управління проєктами, перетворюються на незамінний ресурс, що поєднує зручність використання, гнучкість налаштувань та можливість адаптації до специфічних потреб користувачів. Такі платформи дозволяють не лише систематизувати робочі процеси, але й забезпечити прозорість виконання завдань, що є важливим фактором для досягнення поставлених цілей. Зростання популярності віддаленої роботи та розподілених команд лише посилює потребу в надійних інструментах, які можуть ефективно координувати діяльність у реальному часі, незалежно від географічного розташування учасників.

Мета дипломної роботи полягає в розробці функціональної вебсистеми для організації та контролю виконання проєктних завдань, яка забезпечить зручний інтерфейс, безпечне управління доступом і адаптивність до потреб різних груп користувачів, таких як менеджери, виконавці та адміністратори. Для досягнення мети передбачається провести детальний аналіз вимог сучасних проєктних команд, спроєктувати архітектуру системи на основі моделі MVC, реалізувати програмне забезпечення з акцентом на інтуїтивність і продуктивність, а також

провести комплексне тестування для забезпечення стабільності та відповідності стандартам якості. Результатом роботи стане готове рішення, яке можна інтегрувати в реальні проєктні процеси для підвищення ефективності командної роботи.

Актуальність теми зумовлена зростанням попиту на автоматизовані рішення в управлінні проєктами, що особливо помітно в умовах переходу багатьох організацій до дистанційної роботи. Вебсистема, яка розробляється в рамках цього проєкту, має на меті не лише спростити координацію завдань, але й створити передумови для підвищення ефективності командної роботи. Дослідження охоплює як теоретичні аспекти створення вебсистем, так і практичні кроки з їх реалізації, що дозволяє поєднати фундаментальні знання з прикладними навичками в галузі веброзробки. Аналіз сучасних тенденцій показує, що такі системи стають невід'ємною частиною бізнес-процесів, сприяючи інноваціям і конкурентоспроможності організацій.

Таким чином, дипломна робота спрямована на створення функціонального програмного продукту, який відповідатиме потребам сучасних проєктних команд, а також на поглиблення розуміння процесів розробки вебсистем – від аналізу вимог до впровадження та вдосконалення. Отримані результати можуть бути застосовані в реальних умовах для підвищення ефективності управління проєктами в різних сферах діяльності, таких як ІТ, маркетинг, освіта та виробництво, сприяючи розвитку цифрової інфраструктури сучасних організацій.

# 1. ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

## 1.1. Змістовий аналіз предметної області, її структурних та функціональних особливостей

Сфера управління проєктними завданнями посідає одне з центральних місць у сучасному інформаційному просторі, де стрімкий розвиток технологій, глобалізація робочих процесів та зростання обсягів даних створюють нові виклики для організацій і команд. Ця предметна область, яка охоплює широкий спектр діяльності, пов’язаної з плануванням, координацією, розподілом ресурсів та відстеженням прогресу, набуває особливої актуальності в умовах, коли ефективність і швидкість виконання завдань стають визначальними факторами успіху. Вебсистеми, що використовуються для управління проєктними завданнями, представляють собою складний і багатогранний сегмент інформаційних технологій, який інтегрує в собі як технічні аспекти, так і організаційні принципи, адаптовані до потреб сучасного робочого середовища. Аналіз цієї предметної області є важливим кроком для розуміння її глибини, складності та потенціалу, що дозволяє сформувати теоретичну базу для подальших досліджень і розробок у цій сфері. У цьому підрозділі буде проведено детальний розгляд змістових аспектів, структурних компонентів та функціональних особливостей, які характеризують управління проєктними завданнями як окрему дисципліну в контексті сучасних інформаційних систем.

Сучасний світ відзначається високою динамікою змін, що впливає на всі аспекти людської діяльності, зокрема на організацію праці. У таких умовах традиційні методи управління проєктами, які базувалися на паперовій документації та особистих зустрічах, поступово втрачають свою актуальність, поступаючись місцем цифровим рішенням. Предметна область управління проєктними завданнями відображає цю трансформацію, стаючи ареною, де технології та управлінські підходи перетинаються, створюючи нові можливості

для підвищення продуктивності. Вона охоплює не лише технічні процеси, пов’язані з автоматизацією завдань, але й соціальні аспекти, такі як комунікація між учасниками, мотивація та розподіл відповідальності. Ця багатогранність робить аналіз предметної області надзвичайно важливим, адже він дозволяє виявити ключові тенденції, які формують сучасні підходи до управління, а також зрозуміти, як ці тенденції можуть еволюціонувати в майбутньому. Усе це сприяє формуванню цілісного уявлення про те, як інформаційні технології можуть слугувати інструментом для вирішення складних організаційних завдань.

Структурно предметна область управління проєктними завданнями складається з кількох взаємопов’язаних елементів, які разом створюють основу для функціонування будь-якої системи, орієнтованої на координацію робочих процесів. Одним із таких елементів є управління користувачами, яке передбачає створення системи ідентифікації та авторизації, де кожен учасник проєкту отримує унікальний профіль із визначеними правами доступу. Цей аспект є фундаментом для забезпечення безпеки даних і чіткого розподілу обов’язків, що особливо актуально в умовах роботи з великими командами, де кількість учасників може сягати десятків чи навіть сотень осіб. Іншим важливим структурним компонентом є організація проєктів, яка включає створення робочих просторів, де можна структурувати завдання, визначати їхні пріоритети та встановлювати взаємозв’язки між різними етапами роботи. Ця складова дозволяє формувати гнучкі рамки для реалізації ініціатив, адаптуючи їх до специфіки конкретного проєкту чи галузі. Розподіл завдань є ще одним ключовим елементом, який передбачає деталізацію робочих обов’язків, визначення виконавців і встановлення термінів виконання, що сприяє раціональному використанню ресурсів і уникненню дублювання зусиль. Нарешті, контроль виконання завдань завершує цю структуру, забезпечуючи механізми для моніторингу прогресу, оцінки результатів і своєчасного реагування на можливі відхилення від плану. Кожен із цих компонентів взаємодіє з іншими утворюючи складну, але гармонійну систему яка відображає всю глибину предметної області.

Функціональні особливості предметної області управління проєктними завданнями визначаються потребами тих, хто задіяний у робочому процесі – від менеджерів, які планують і координують діяльність, до виконавців, які безпосередньо реалізують поставлені завдання. Однією з центральних функцій є планування, яке включає визначення цілей, розбиття їх на окремі завдання та встановлення чітких дедлайнів. Цей процес вимагає не лише технічної реалізації, але й глибокого розуміння специфіки проєкту, що дозволяє адаптувати план до реальних умов і ресурсів. Інший важливий аспект – координація, яка забезпечує безперервний зв'язок між учасниками, дозволяючи обмінюватися інформацією, уточнювати деталі та вирішувати конфлікти в режимі реального часу. Така функція стає особливо важливою в умовах розподілених команд, де фізична відстань може ускладнювати спілкування.

Розподіл ресурсів є ще одним ключовим елементом, який передбачає оптимальне використання людського капіталу, обладнання та часу, що дозволяє мінімізувати витрати та підвищувати ефективність. Моніторинг прогресу, у свою чергу, дає змогу відстежувати стан завдань, аналізувати виконану роботу та вносити корективи, якщо виникають затримки чи інші проблеми. Візуалізація робочих процесів, наприклад, через дошки чи графіки, полегшує сприйняття інформації, роблячи її доступною навіть для тих, хто не має глибоких технічних знань. Генерація звітів є ще однією важливою функцією, яка дозволяє оцінити продуктивність команди, визначити сильні та слабкі сторони та підготувати аналітичні дані для подальшого планування. Нарешті, підтримка комунікації та інтеграція з іншими інструментами, такими як електронна пошта чи календарі, забезпечує гнучкість і зручність у використанні, адаптуючи систему до різних робочих стилів.

Предметна область управління проєктними завданнями також характеризується своєю еволюцією, що зумовлена постійним розвитком технологій і зміною потреб користувачів. У міру того, як компанії переходять до гібридних чи повністю дистанційних форматів роботи, зростає потреба в

системах, які можуть забезпечити не лише технічну підтримку, але й психологічний комфорт для учасників. Це включає створення інтуїтивно зрозумілих інтерфейсів, підтримку багатомовності та адаптацію до різних пристроїв – від настільних комп'ютерів до смартфонів. Крім того, усе більшої ваги набувають аспекти безпеки, адже конфіденційність даних стає пріоритетом у сучасному цифровому світі. Аналіз цих тенденцій показує, що предметна область є динамічною, що вимагає від дослідників і розробників постійного оновлення своїх знань і підходів. Ця динаміка також відображається в зростанні популярності методологій, таких як Agile чи Kanban, які акцентують увагу на гнучкості та ітеративному підході до управління, що робить їх невід'ємною частиною сучасної практики.

Таким чином, змістовий аналіз предметної області управління проєктними завданнями розкриває її як складну і багатошарову дисципліну, яка поєднує в собі технічні, організаційні та соціальні аспекти. Структурні компоненти, такі як управління користувачами, організація проєктів, розподіл завдань і контроль виконання, формують основу для будь-якої системи, орієнтованої на координацію робочих процесів. Функціональні особливості, включаючи планування, координацію, моніторинг і візуалізацію, відображають потреби користувачів і еволюцію підходів до управління. Цей аналіз не лише поглиблює теоретичне розуміння предметної області, але й закладає фундамент для подальшого дослідження її потенціалу в контексті сучасних інформаційних технологій, що є важливим кроком на шляху до створення ефективних і адаптивних рішень для організацій усього світу. Особливу увагу приділено інтеграції інноваційних інструментів, таких як дошки Kanban, які підвищують прозорість і продуктивність команд, а також адаптації систем до віддаленої роботи, що стає дедалі актуальнішим у глобальному масштабі. Крім того, аналіз підкреслює важливість автоматизації рутинних процесів, таких як сповіщення про дедлайни, що значно полегшує щоденну роботу команд. цей підхід відкриває можливості для інтеграції з хмарними технологіями.

## 1.2. Аналіз наявного програмно-технічного забезпечення предметної області

У сучасному світі інформаційних технологій управління проєктними завданнями стало невід'ємною частиною діяльності багатьох організацій, незалежно від їхнього розміру чи сфери діяльності. Для забезпечення ефективної координації, планування та контролю виконання завдань розроблено численні програмні продукти, які пропонують різноманітні підходи до вирішення цих задач. Аналіз наявного програмно-технічного забезпечення є важливим етапом дослідження, адже він дозволяє не лише ознайомитися з досвідом провідних розробників, але й визначити ключові тенденції, які формують сучасний ринок програмних рішень. Такий підхід допомагає уникнути типових помилок, врахувати передові практики та сформувати чітке уявлення про те, які аспекти варто врахувати при створенні нового програмного продукту. У цьому підрозділі буде проведено детальний огляд трьох популярних інструментів для управління проєктними завданнями – Trello, Asana та Jira – із акцентом на їхнє призначення, інтерфейс, функціонал, а також сильні та слабкі сторони. Кожен із цих продуктів має свої особливості, які роблять їх популярними серед користувачів, але водночас вони мають і певні обмеження, що можуть впливати на їхню ефективність у різних сценаріях використання.

Trello – одним із найпоширеніших інструментів для управління проєктами, який базується на концепції Kanban-дошок. Його основне призначення полягає в організації завдань через візуалізацію робочих процесів, що дозволяє командам легко планувати, розподіляти та відстежувати прогрес. Розробником Trello є компанія Atlassian, яка відома своїми рішеннями для командної співпраці, такими як Jira та Confluence. Trello (Рисунок 1.1) орієнтований на широку аудиторію – від невеликих команд і стартапів до індивідуальних користувачів, які використовують його для особистих цілей, наприклад, планування щоденних справ чи створення контент-планів.

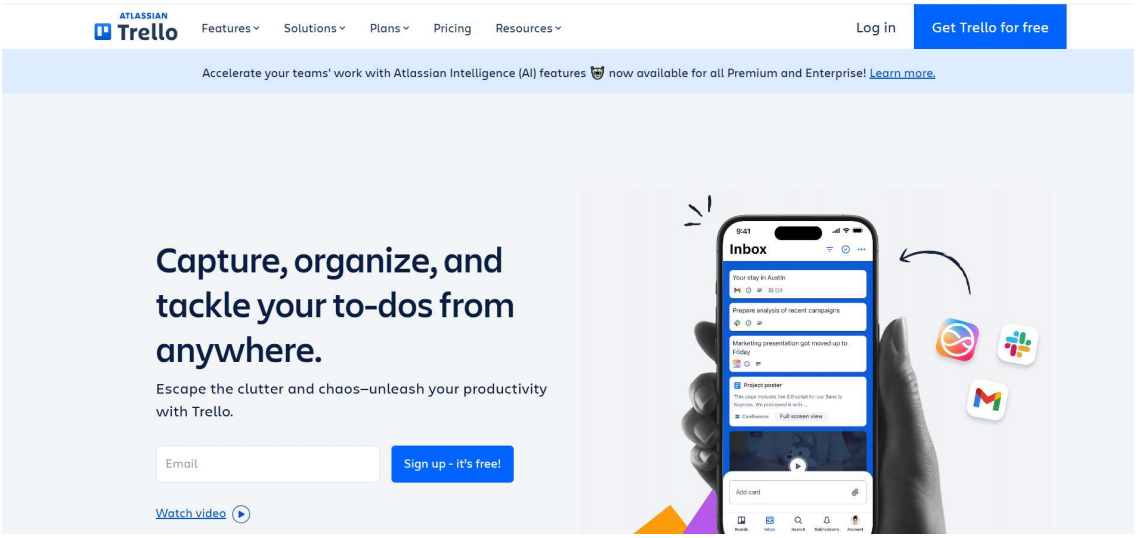



Рисунок 1.1 – Дизайн основних сторінок вебсистеми «Trello»

Дизайн головної сторінки Trello виконаний у сучасному та привабливому стилі, що одразу привертає увагу користувачів. На верхній частині сторінки розташований логотип Trello та кнопки для входу або реєстрації, які виділені контрастними кольорами – синім і білим – для зручності навігації. Основний фон сторінки має світло-сірий відтінок із додаванням яскравих ілюстрацій, які демонструють дошки Trello з картками та колонками. Центральна частина сторінки містить заклик до дії – «Trello допомагає командам рухати роботу вперед» – із кнопкою «Зареєструватися – це безкоштовно!». Нижче представлені блоки з описом можливостей, таких як візуалізація проєктів, інтеграція з іншими інструментами та автоматизація, які супроводжуються зображеннями інтерфейсу. Колірна палітра включає м’які відтінки синього, зеленого та помаранчевого, що створює відчуття легкості та дружелюбності. У нижній частині сторінки є інформація про компанії, які використовують Trello, а також посилання на блог, підтримку та політику конфіденційності, що додає довіри до продукту.

Функціонал Trello включає базові можливості для управління завданнями: створення карток із описами, дедлайнами, вкладеннями та коментарями, призначення виконавців, а також інтеграцію з популярними сервісами, такими як

Google Drive, Slack і Dropbox. Користувачі можуть додавати чек-листи до карток, що дозволяє розбивати завдання на підзавдання, а також використовувати розширення (Power-Ups), наприклад, для інтеграції з календарем чи автоматизації певних дій. Безкоштовна версія Trello пропонує необмежену кількість карток і до 10 дошок на робочий простір, але має обмеження, такі як максимальний розмір вкладень у 10 МБ і можливість підключення лише одного Power-Up.

Asana – це інструмент для комплексного управління проєктами, який підходить як для невеликих команд, так і для великих організацій із складними робочими процесами. Його призначення полягає в забезпеченні планування, постановки цілей, відстеження прогресу та координації командної роботи. Розробником є компанія Asana, Inc., яка зосереджена на створенні рішень для підвищення продуктивності. Asana (Рисунок 1.2) популярна серед маркетологів, розробників і менеджерів проєктів завдяки своїй гнучкості та широкому функціоналу.

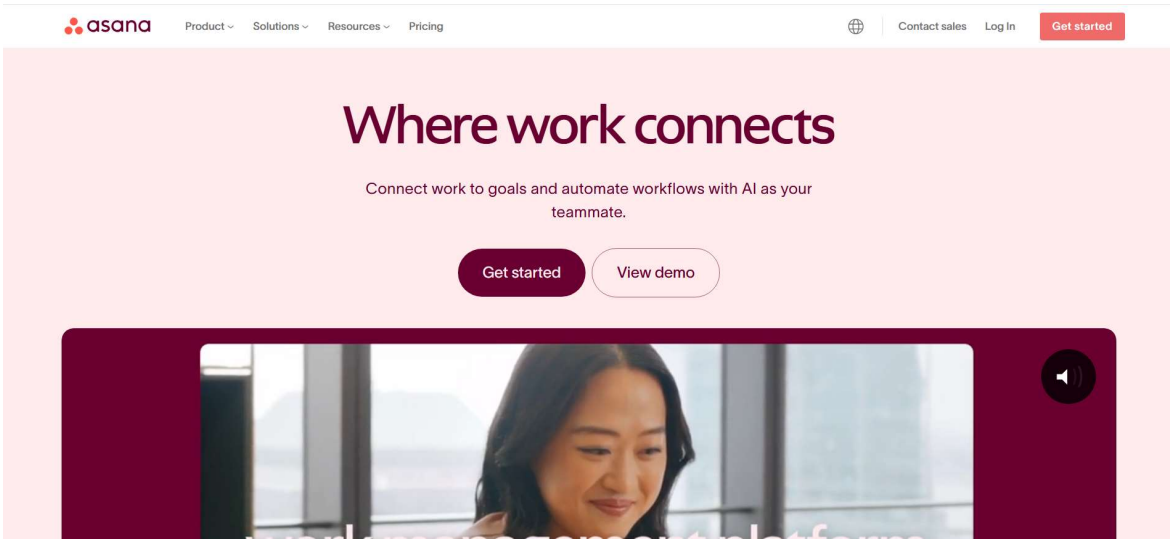



Рисунок 1.2 – Дизайн основних сторінок вебсистеми «Asana»

Інтерфейс Asana виглядає більш насиченим у порівнянні з Trello, але залишається зручним для користувачів. На головній сторінці відображається

робочий простір із завданнями, які можна переглядати у форматі списків, дощок або таймлайнів. Ліворуч розташована бічна панель із навігацією, де користувач може обрати проєкт, переглянути календар, звіти чи вхідні повідомлення. У верхній частині екрана є пошуковий рядок і кнопки для швидкого створення завдань. Кожне завдання відкривається у вигляді детальної картки, де можна додати опис, теги, дедлайни, виконавців і коментарі. Дизайн виконаний у світлих тонах із акцентами на кольорових іконках, що допомагає швидко орієнтуватися в інтерфейсі. Asana також підтримує кастомізацію, дозволяючи користувачам налаштовувати вигляд проєктів відповідно до їхніх потреб.

Функціонал Asana є значно ширшим, ніж у Trello. Користувачі можуть створювати завдання з підзавданнями, встановлювати залежності між ними, використовувати таймлайни для планування та відстежувати прогрес у реальному часі. Asana підтримує інтеграцію з такими сервісами, як Zoom, Microsoft Teams і Google Calendar, що робить її зручною для команд, які використовують різні інструменти. Також доступні шаблони проєктів, які спрощують початок роботи, і розширені звіти для аналізу продуктивності. Безкоштовна версія дозволяє працювати командам до 15 осіб, але для доступу до таймлайнів чи розширеної аналітики потрібна платна підписка. Крім того, Asana пропонує функцію автоматичних нагадувань, що допомагає уникати пропуску дедлайнів, а також гнучкі налаштування прав доступу для різних ролей у команді. Система також підтримує мобільний додаток, що забезпечує зручний доступ до проєктів у будь-який час, а її інтуїтивний інтерфейс сприяє швидкому освоєнню навіть для новачків. Завдяки цим можливостям Asana стає універсальним рішенням для команд різного масштабу та напрямку діяльності.

Jira – це спеціалізований інструмент для управління проєктами, орієнтований переважно на команди розробників програмного забезпечення. Його призначення полягає у відстеженні завдань, багів, спринтів і підтримці агілевих методологій, таких як Scrum і Kanban. Розробником є Atlassian, та сама компанія, що створила Trello, але Jira орієнтована на більш технічну аудиторію.

Вона широко використовується в IT-компаніях для управління розробкою, тестуванням і розгортанням програмного забезпечення, що робить її популярною серед програмістів і тестувальників. Крім того, Jira пропонує розширені налаштування робочих процесів, що дозволяють адаптувати систему під специфічні потреби проєктів, а також інтеграцію з інструментами розробки, такими як Bitbucket і GitHub. Завдяки детальним фільтрам і панелям управління, вона забезпечує глибокий аналіз продуктивності команд, а платна версія розширює можливості для великих організацій, включаючи підтримку кількох проєктів одночасно. Ці характеристики роблять Jira (Рисунок 1.3) незамінним інструментом для складних IT-проєктів із високими вимогами до структуризації.

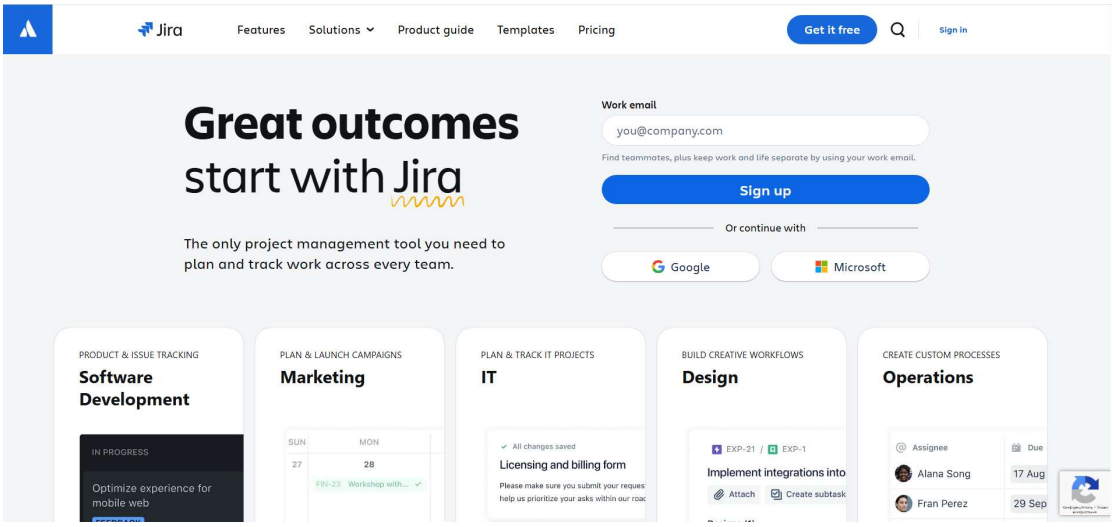



Рисунок 1.3 – Дизайн основних сторінок вебсистеми «Jira»

Дизайн головної сторінки Jira виглядає більш строгим і функціональним, що відображає її орієнтацію на технічних користувачів. У верхній частині сторінки розташований логотип Atlassian і навігаційне меню з пунктами, такими як «Продукти», «Рішення», «Ресурси» та «Ціни». Основний фон виконаний у білому кольорі з додаванням синьо-зелених акцентів, які є фірмовими для Atlassian. Центральний блок містить заголовок «Jira Software – інструмент управління проєктами №1 для гнучких команд» із кнопкою «Спробувати

безкоштовно». Нижче представлені розділи з описом можливостей, таких як планування спринтів, відстеження багів і створення звітів, які супроводжуються зображеннями інтерфейсу – дощок, беклогів і діаграм. Колірна схема включає відтінки синього, зеленого та сірого, що створює відчуття професіоналізму та надійності. У нижній частині сторінки є інформація про клієнтів, таких як Spotify і Cisco, а також посилання на документацію, підтримку та спільноту, що підкреслює орієнтацію на технічну аудиторію.

Функціонал Jira є одним із найпотужніших серед інструментів управління проєктами. Вона підтримує створення детальних робочих процесів, які можна налаштувати під потреби команди, відстеження багів, управління беклогами та генерацію звітів, таких як діаграми вигорання. Jira інтегрується з іншими продуктами Atlassian, такими як Confluence, а також із зовнішніми сервісами, наприклад, GitHub і Slack. Користувачі можуть створювати кастомні поля, автоматизувати повторювані дії та налаштовувати сповіщення. Інструмент підтримує методології Agile і Scrum, дозволяючи командам працювати зі спринтами, беклогами та дошками. Однак безкоштовна версія обмежена, і для повноцінного використання потрібна платна підписка, яка залежить від розміру команди.

Таблиця 1.1 – Порівняння функцій застосунків

| Програмне забезпечення | Переваги | Недоліки |
|---|---|---|
| Trello | Простота та інтуїтивність дизайну, безкоштовна версія з базовими функціями, підтримка Kanban-дошок, інтеграція з популярними сервісами, привабливий і дружелюбний дизайн головної сторінки. | Обмежений функціонал для складних проєктів, максимальний розмір вкладень 10 МБ у безкоштовній версії, слабка аналітика, лише одне розширення (Power-Up) у безкоштовному тарифі. |

Продовження таблиці 1.1 – Порівняння функцій застосунків

| | | |
|---|---|---|
| Asana | Гнучкість налаштувань, підтримка різних режимів перегляду (списки, дошки, таймлайни), інтеграція з багатьма сервісами, розширені звіти, сучасний і енергійний дизайн головної сторінки. | Висока вартість преміум-версій, складність освоєння для новачків через велику кількість функцій, обмеження до 15 осіб у безкоштовній версії, можлива повільність при великих обсягах даних. |
| Jira | Потужний функціонал для технічних команд, підтримка Agile/Scrum, гнучкі налаштування робочих процесів, інтеграція з розробницькими інструментами, професійний і строгий дизайн головної сторінки. | Складність для нетехнічних користувачів, висока ціна для великих команд, потреба в навчанні для ефективного використання, менш привабливий дизайн, обмежена безкоштовна версія. |

Проведений аналіз наявного програмно-технічного забезпечення у сфері управління проєктними завданнями показує, що кожен із розглянутих інструментів має свої унікальні особливості, які роблять його придатним для певних сценаріїв використання.

## 1.3. Визначення функціональних та нефункціональних вимог до вебсистеми

Визначення вимог до вебсистеми для організації та контролю виконання проєктних завдань є важливим етапом розробки, який дозволяє чітко окреслити її можливості та якісні характеристики. Функціональні вимоги визначають конкретні можливості системи, тобто описують, які задачі вона має виконувати, щоб відповідати потребам користувачів – команд, які працюють над проєктами,

та менеджерів, які контролюють їх виконання. Система повинна надавати зручний інтерфейс для створення структурованого робочого простору, де користувачі зможуть ефективно взаємодіяти, обмінюватися інформацією та відстежувати прогрес у реальному часі.

Функціональних вимог до системи:

– реєстрація користувачів: система має забезпечувати створення нового облікового запису шляхом введення електронної пошти, пароля та базової інформації (наприклад, ім’я, роль у команді), із відправленням листа для підтвердження на електронну пошту;

– авторизація користувачів: система повинна надавати безпечний вхід за допомогою електронної пошти та пароля, із повідомленням про помилку у разі неправильного введення даних і тимчасовим блокуванням після кількох невдалих спроб для захисту від несанкціонованого доступу;

– перегляд завдань: користувач має мати можливість переглядати список усіх завдань у межах проєкту у вигляді дошки з колонками (наприклад, «Backlog», «To Do», «Doing», «Done») для швидкої оцінки їхнього статусу;

– додавання завдання: система повинна дозволяти створювати нове завдання з вказівкою назви, опису, дедлайну, пріоритету, категорії та вкладень (файлів, зображень);

– редагування завдання: користувач має мати змогу змінювати параметри завдання (назву, опис, дедлайн, пріоритет, статус, вкладення);

– призначення виконавця: система повинна дозволяти призначати виконавців для завдань із урахуванням їхньої ролі та завантаженості, із можливістю вибору зі списку членів команди;

– контроль статусу завдань: користувач має мати можливість відстежувати статус завдань у реальному часі, переглядати етапи виконання та історію змін статусу;

– оновлення статусу виконання: система повинна дозволяти змінювати статус завдання шляхом перетягування картки між колонками або через форму;

– генерація звітів по завданнях: система має надавати можливість створювати звіти про кількість виконаних завдань, витрачений час, продуктивність команди та затримки, із можливістю експорту в PDF або Excel;

– отримання сповіщень: система повинна надсилати сповіщення про важливі події (призначення на завдання, зміна статусу, наближення дедлайну) через інтерфейс або електронну пошту.

Нефункціональні вимоги визначають якісні характеристики системи, які впливають на її продуктивність, безпеку, зручність використання та інші аспекти, що не стосуються безпосередньо функціоналу. Вони є важливими для забезпечення комфортної роботи користувачів і стабільності системи. Перелік основних нефункціональних вимог наведено нижче:

− продуктивність: час завантаження сторінки не має перевищувати 2 секунд за стабільного інтернет-з’єднання, а обробка запитів (наприклад, додавання завдання) – 1-2 секунди;

− масштабованість: система має підтримувати одночасну роботу щонайменше 1000 користувачів без зниження продуктивності, із можливістю масштабування шляхом додавання серверних ресурсів;

− безпека: усі дані користувачів (особиста інформація, паролі, файли) мають бути захищені шифруванням (HTTPS для передачі даних, хешування паролів), із захистом від атак, таких як SQL-ін’єкції та XSS;

− зручність використання: інтерфейс має бути інтуїтивно зрозумілим, із чіткою навігацією та мінімальною кількістю кроків для виконання дій, відповідаючи стандартам UX/UI, із контрастними кольорами та зручним розташуванням елементів;

− доступність: система має бути зручною для користувачів із різним рівнем технічної підготовки та враховувати потреби людей із обмеженими можливостями (наприклад, підтримка читання з екрану);

− сумісність: система повинна коректно працювати у всіх популярних браузерах останніх версій;

− надійність: рівень доступності має бути не нижче 99,9%, із автоматичним збереженням даних у разі збоїв для уникнення їх втрати;

− локалізація: система має підтримувати щонайменше дві мови (українську та англійську) із можливістю перемикання без перезавантаження сторінки, із перекладом усіх текстових елементів;

− продуктивність при великих обсягах даних: система повинна відображати до 500 завдань на одній дошці без затримок і забезпечувати швидкий пошук за ключовими словами;

− технічна підтримка: система має включати розділ із документацією, FAQ і можливістю звернення до технічної підтримки через форму зворотного зв’язку або чат у реальному часі.

Для кращого розуміння взаємодії користувачів із системою та структурування функціональних вимог використовуються діаграми UML (Unified Modeling Language), які є стандартизованою мовою моделювання для візуалізації архітектури програмного забезпечення. Одним із найпоширеніших типів діаграм є діаграма використання (Use Case Diagram), яка допомагає ідентифікувати основні сценарії використання системи, визначити ролі користувачів і їх взаємодію з функціоналом, дозволяючи розробникам і замовникам узгодити очікування щодо можливостей системи. Діаграма також включає розширені сценарії, наприклад, авторизацію та генерацію сповіщень, що полегшує розуміння логіки роботи системи. Завдяки цьому інструменту стало можливим виявити потенційні проблеми на ранніх етапах розробки та оптимізувати взаємодію між компонентами. Крім того, діаграма (Рисунок 1.4) слугує основою для подальшого створення технічної документації та тестування, забезпечуючи єдине бачення функціоналу серед усіх учасників проєкту. Це також допомагає уникнути непорозумінь між розробниками та замовниками.

Реалізація такої вебсистеми сприятиме підвищенню продуктивності команд, спрощенню контролю виконання завдань і відповідатиме сучасним вимогам до інструментів управління проєктами.


Рисунок 1.4 – Діаграма варіантів використання

Вона включає актора «Користувач», який представляє всіх осіб, що взаємодіють із системою – менеджерів, членів команди, адміністраторів. Усі варіанти використання згруповані в межах прямокутника «Вебсистема для організації та контролю виконання проєктних завдань», що позначає межі системи. Варианти використання поділені на кілька категорій: вхід у систему (реєстрація та авторизація), базова робота із завданнями (перегляд, додавання, редагування завдань), розширені можливості (призначення виконавця, контроль статусу, оновлення статусу, звіти) і додаткові функції (отримання сповіщень).

Більшість функцій, таких як перегляд, додавання, редагування завдань, призначення виконавців, контроль і оновлення статусу, а також генерація звітів, вимагають авторизації, що позначено відношенням «include». Відношення «extend» показує, що контроль статусу завдань може розширюватися до генерації звітів, а призначення виконавця та оновлення статусу – до надсилання сповіщень, якщо ці дії активують відповідні події. Ця діаграма є важливим інструментом для аналізу функціональних вимог, оскільки вона чітко визначає дії користувача, залежності між функціями та допомагає уникнути непорозумінь на етапі проектування, закладаючи міцний фундамент для подальшої розробки системи, яка відповідатиме потребам користувачів і сучасним технологічним стандартам.

## 1.4. Висновки дослідження предметної області та постановки задачі

У першому розділі кваліфікаційної роботи було здійснено детальне дослідження предметної області управління проєктними завданнями, що підтвердило актуальність розробки вебсистеми для організації та контролю виконання робочих процесів. У ході аналізу було розглянуто сучасні інструменти, такі як Trello, Asana та Jira, які є популярними рішеннями для управління проєктами. Вони демонструють сильні сторони, зокрема інтуїтивний інтерфейс у Trello, розширені функції планування в Asana та підтримку технічних команд у Jira, але мають і слабкі місця, такі як обмеження безкоштовних версій, висока ціна преміум-функцій чи складність для новачків, що створює потребу в новому рішенні, яке б урівноважило простоту, функціональність і доступність.

На наступному етапі було сформульовано функціональні та нефункціональні вимоги до вебсистеми, що дозволило визначити її основні можливості та якісні характеристики. Функціональні вимоги охоплюють створення, редагування та контроль завдань, призначення виконавців, генерацію звітів і сповіщення, забезпечуючи зручне управління проєктами. Нефункціональні вимоги зосереджені на забезпеченні високої продуктивності, безпеки, зручності інтерфейсу та адаптивності до різних пристроїв і браузерів. Для чіткого уявлення про взаємодію користувачів із системою було створено UML-діаграму використання, яка ілюструє ключові сценарії роботи та їх взаємозв’язки, сприяючи кращому розумінню функціоналу.

# 2. ПРОЄКТУВАННЯ ВЕБСИСТЕМИ

## 2.1. Аналіз та вибір архітектури вебсистеми

Проєктування вебсистеми для організації та контролю виконання проєктних завдань є важливим етапом, на якому визначаються способи реалізації вимог, сформульованих у попередньому розділі. У цьому підрозділі буде описано архітектуру системи, її структуру, а також обґрунтовано проєктні рішення, які забезпечують виконання технічного завдання, сумісність компонентів і можливість подальшого масштабування.

Для розробки архітектури системи було розглянуто кілька популярних архітектурних підходів, кожен із яких має свої особливості.

− клієнт-серверна архітектура передбачає чіткий розподіл між клієнтською частиною, яка відповідає за інтерфейс користувача, і серверною частиною, що обробляє дані та бізнес-логіку. Такий підхід добре підходить для вебсистем, оскільки дозволяє легко масштабувати серверну частину при зростанні кількості користувачів, але може ускладнити розробку через необхідність управління мережевими взаємодіями, а також потребує ретельного забезпечення безпеки передачі даних;

− монолітна архітектура, у свою чергу, об’єднує всі компоненти системи в єдине ціле, що спрощує розробку на початкових етапах, адже всі модулі тісно пов’язані і працюють у межах одного додатка. Однак із часом монолітна система стає складною для масштабування та підтримки, особливо якщо потрібно вносити зміни в окремі частини без впливу на всю систему;

− мікросервісна архітектура пропонує інший підхід, розділяючи систему на невеликі незалежні сервіси, які взаємодіють через API. Це забезпечує високу гнучкість, адже кожен сервіс можна розробляти, розгортати та масштабувати окремо, але водночас ускладнює управління системою через необхідність координації між сервісами, а також може призводити до затримок у зв’язку між ними;

− шарова архітектура розділяє систему на кілька рівнів, таких як представлення, бізнес-логіка та доступ до даних, що сприяє модульності та спрощує тестування, оскільки кожен рівень має чітко визначені обов’язки. Проте надмірна кількість шарів може ускладнити систему, а передача даних між шарами іноді знижує продуктивність;

− подієво-орієнтована архітектура, компоненти системи реагують на події, що генеруються іншими частинами. Такий підхід добре працює для асинхронних систем, наприклад, для обробки сповіщень, але може бути складним у реалізації через необхідність управління потоками подій і забезпечення їхньої послідовності.

Для вебсистеми управління проєктними завданнями було обрано архітектурний патерн MVC (Model-View-Controller), який є оптимальним вибором для створення вебдодатків. У цьому патерні модель відповідає за обробку даних і бізнес-логіку, представлення забезпечує відображення даних користувачу через інтерфейс, а контролер обробляє запити користувача та координує взаємодію між моделлю та представленням. MVC дозволяє досягти чіткого розподілу обов’язків між компонентами, що полегшує розробку та тестування, адже зміни в інтерфейсі не впливають на бізнес-логіку, а оновлення моделі не зачіпають представлення. Це також сприяє масштабованості системи, оскільки нові функції можна додавати, не змінюючи основну структуру. Крім того, MVC спрощує підтримку коду, адже кожен компонент ізольований і має чітко визначену роль, що особливо важливо для командної розробки. Однак у цього підходу є й певні складнощі: для новачків MVC може здаватися складним через необхідність розуміння взаємодії між трьома компонентами, а в невеликих проєктах його використання може виглядати надмірним, додаючи зайву складність там, де простіший підхід був би ефективнішим. Також іноді виникають труднощі з управлінням великою кількістю контролерів у складних системах, що може призводити до дублювання коду, якщо не дотримуватися чітких правил організації.

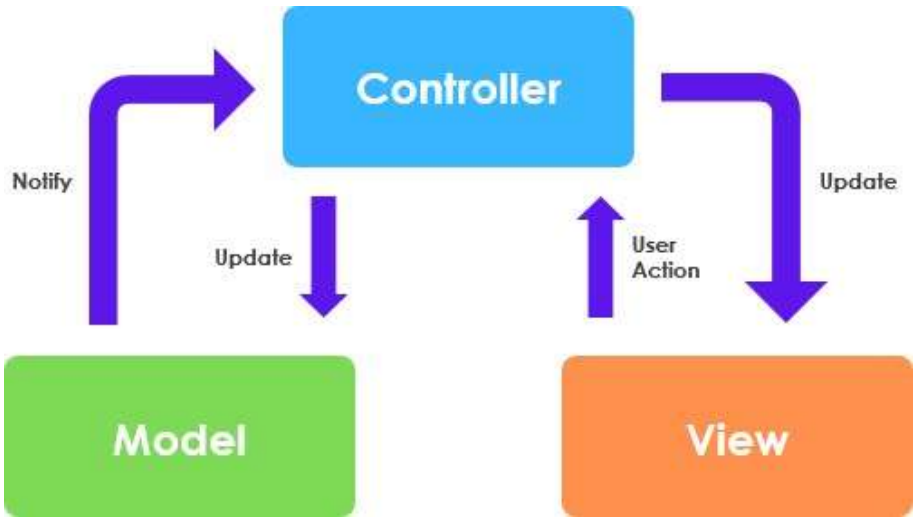



Рисунок 2.1 – Діаграма архітектурного патерну MVC

Щоб деталізувати архітектуру, було розроблено діаграму архітектурного патерну MVC (Рисунок 2.1). На цій діаграмі показано, як модель (MongoDB із Mongoose) забезпечує збереження та обробку даних, представлення (React Client із UI Components і Drag-and-Drop Logic) відповідає за відображення інтерфейсу, а контролер (Node.js Server із Express Routes і JWT Authentication) обробляє запити та забезпечує взаємодію між моделлю та представленням. Діаграма ілюструє основний потік даних: користувач взаємодіє з інтерфейсом, React Client надсилає запити до Node.js Server через REST API (Axios), сервер обробляє запити, звертається до бази даних через Mongoose і повертає результат, який відображається користувачу. Цей процес забезпечує швидку обробку запитів, адже сервер використовує асинхронні методи для взаємодії з MongoDB, що зменшує час очікування. Крім того, використання REST API дозволяє легко інтегрувати систему з іншими сервісами в майбутньому, наприклад, для аналітики чи сповіщень.

Архітектура системи базується на трирівневій моделі, що включає рівень представлення, рівень бізнес-логіки та рівень доступу до даних. Рівень представлення відповідає за взаємодію з користувачем через адаптивний вебінтерфейс, який коректно відображається на різних пристроях, включаючи настільні комп’ютери та мобільні телефони. Рівень бізнес-логіки обробляє

запити, виконує операції з даними, такі як створення завдань, призначення виконавців і оновлення статусів, а також забезпечує логіку сповіщень для інформування користувачів про зміни. Рівень доступу до даних відповідає за збереження та обробку даних у базі даних MongoDB, гарантуючи їхню цілісність, безпеку та швидкий доступ через Mongoose. Крім того, трирівнева модель забезпечує гнучкість при масштабуванні, дозволяючи окремо оптимізувати кожен рівень залежно від навантаження. Це також полегшує тестування, оскільки кожен рівень можна перевіряти незалежно від інших. Завдяки такому підходу система залишається стійкою до змін у вимогах, що є важливим для майбутнього розвитку. Чіткий розподіл сприяє командній роботі, дозволяючи розробникам спеціалізуватися на конкретних рівнях без конфліктів у коді.

Підсистема управління користувачами, яка включає модулі:

− модуль реєстрації (створення облікових записів із валідацією email і пароля);

− модуль авторизації (обробка входу через JWT-токени);

− модуль профілів (редагування особистих даних користувача).

Підсистема управління завданнями, яка складається з модулів:

− модуль створення завдань (додавання нових завдань із полями: назва, дедлайн, пріоритет, виконавці);

− модуль редагування завдань (оновлення статусу та деталей);

− модуль перегляду завдань (фільтрація та сортування за статусом і пріоритетом);

− модуль призначення виконавців (прив’язка користувачів до завдань);

− модуль контролю статусу (переміщення завдань між колонками через drag-and-drop).

Підсистема сповіщень, яка включає модулі:

− модуль надсилання сповіщень (генерація повідомлень про нові завдання або зміни статусів);

− модуль управління сповіщеннями.

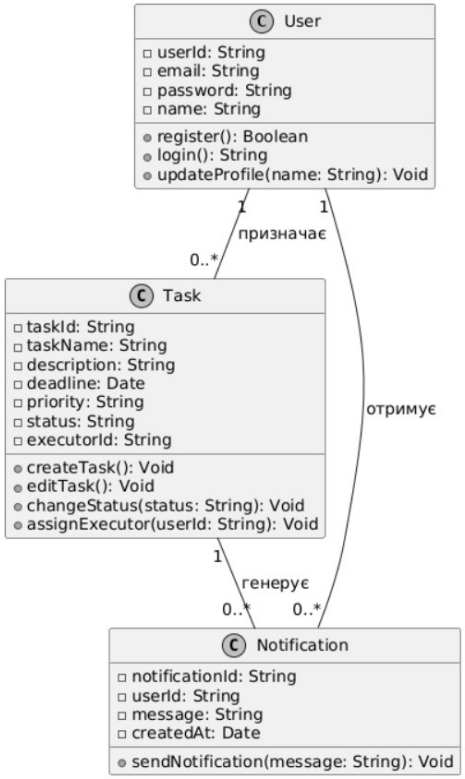



Рисунок 2.2 – Діаграма класів взаємодії модулів.

Структура системи формуватиметься шляхом декомпозиції підсистем на модулі, що дозволить організувати її у вигляді дерева модулів із чітким розподілом обов’язків, видно на діаграмі класів (Рисунок 2.2). Підсистема управління користувачами включатиме модулі для реєстрації, авторизації та управління профілями, які відповідатимуть за створення облікових записів, безпечний вхід і редагування даних користувачів. Підсистема управління завданнями складатиметься з модулів для створення, редагування, перегляду завдань, призначення виконавців і контролю статусу, забезпечуючи основний функціонал системи. Підсистема сповіщень включатиме модулі для надсилання та управління сповіщеннями, щоб користувачі отримували повідомлення про важливі події. Взаємозв’язки між модулями планується організувати через чіткі інтерфейси, наприклад, модуль створення завдань взаємодіятиме з модулем сповіщень для автоматичного повідомлення виконавців, а модуль звітів отримуватиме дані від модуля перегляду завдань.

Декомпозиція системи реалізована за модульним типом, який передбачає поділ на автономні функціональні блоки, такі як авторизація, управління завданнями та дошка Kanban. Окрім модульної, також застосована ієрархічна декомпозиція, де підсистеми розбиті на менші модулі з чіткою субординацією: наприклад, модуль авторизації є базовим, від якого залежить доступ до інших модулів. Такий підхід сприяє підвищенню зрозумілості коду, спрощує тестування окремих компонентів і дозволяє ефективно масштабувати систему при збільшенні функціоналу.

Щодо міжпроцесних залежностей, вони визначені з урахуванням логічного порядку виконання операцій. Модуль авторизації перевіряє права доступу перед тим, як користувач може взаємодіяти з модулем управління завданнями, який у свою чергу передає оновлені дані про статуси до модуля дошки Kanban для відображення змін у реальному часі. Обмін даними між модулями здійснюється через централізовану базу даних MongoDB, що забезпечує узгодженість інформації та швидкий доступ. Наприклад, після створення завдання через модуль створення завдань дані зберігаються в MongoDB, після чого модуль сповіщень автоматично генерує повідомлення для виконавця, а модуль перегляду завдань оновлює список доступних завдань для користувача. Така організація дозволяє уникнути дублювання даних і мінімізує ризик помилок при паралельній роботі кількох користувачів. Крім того, міжпроцесні залежності включають асинхронну взаємодію: наприклад, модуль сповіщень може працювати незалежно від модуля створення завдань, але активізується лише після завершення операції збереження. Це підвищує ефективність системи та її здатність обробляти навантаження до 100 одночасних користувачів, як зазначено у вимогах.

Для деталізації алгоритмів роботи системи наведено блок-схему алгоритму створення завдання (Рисунок 2.3). Цей алгоритм детально описує процес створення завдання: користувач вводить дані (назва, опис, дедлайн, пріоритет, виконавець), React Client формує об’єкт taskData і надсилає POST-запит через

Axios до Node.js Server. Сервер валідує дані (перевірка наявності назви, формату дедлайну, виконавця), зберігає завдання в MongoDB через Mongoose, надсилає сповіщення виконавцю і повертає результат. У разі некоректних даних користувач отримує повідомлення про помилку. Блок-схема чітко ілюструє цей процес із програмними деталями, такими як HTTP-статуси (201 для успіху, 400 для помилки) і взаємодія між клієнтом і сервером.

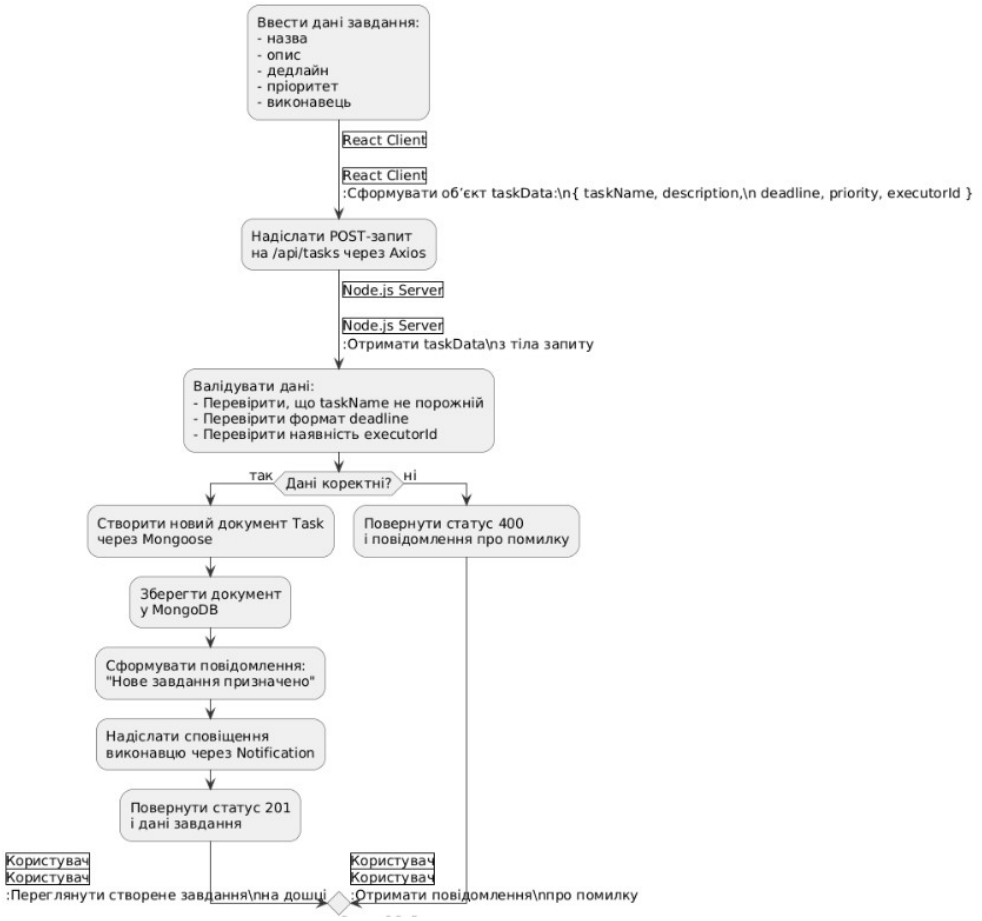



Рисунок 2.3 – Блок-схема алгоритму створення завдання.

Прийняті проєктні рішення закладають основу для подальшої розробки системи. Використання MVC і трирівневої архітектури забезпечує модульність і масштабованість, а декомпозиція на підсистеми та модулі дозволяє чітко розподілити функціональні обов'язки. Такий підхід також сприяє підвищенню безпеки системи, адже кожен рівень ізольовано обробляє свої завдання.

## 2.2. Проєктування структури даних та моделі бази даних

Проєктування логічної моделі бази даних для вебсистеми управління проєктними завданнями зосереджується на створенні структури, яка забезпечить ефективне зберігання та обробку даних, необхідних для реалізації функціональних вимог. На цьому етапі формується модель, що відображає основні сутності, їхні атрибути та зв'язки, щоб забезпечити логіку роботи системи. Логічна модель бази даних розробляється за принципом «сутність-зв'язок», де сутності представляють ключові об'єкти системи, такі як користувачі, проєкти чи завдання, а зв'язки визначають їхню взаємодію. Сутності мають атрибути, які описують їхні характеристики, наприклад, користувач може мати ідентифікатор, ім'я та роль, а завдання – назву, статус і дедлайн. Зв'язки між сутностями показують, як ці об'єкти пов'язані: наприклад, один користувач може створювати кілька завдань, а одне завдання може належати до одного проєкту. У системі управління проєктними завданнями основними сутностями є користувачі, проєкти, завдання, теги та сповіщення, які разом забезпечують функціонал, такий як управління завданнями, відстеження прогресу та надсилання повідомлень. Ця модель дозволяє легко додавати нові атрибути чи зв'язки, що забезпечує гнучкість системи при розширенні. Крім того, логічна структура сприяє ефективному плануванню фізичної реалізації бази даних, зокрема в MongoDB, де документи можуть відображати складні зв'язки. Нижче представлено діаграму «сутність-зв'язок», яка ілюструє логічну модель бази даних (Рисунок 2.4). Діаграма допомагає візуалізувати складні взаємозв'язки між сутностями, що полегшує розуміння структури бази розробниками та замовниками. Вона також слугує основою для подальшого створення фізичної моделі, адаптованої до особливостей MongoDB. Крім того, логічна модель сприяє оптимізації запитів до бази даних, забезпечуючи швидкий доступ до даних у реальному часі. Такий підхід також забезпечує узгодженість даних , що є критично важливим для коректного функціонування системи.

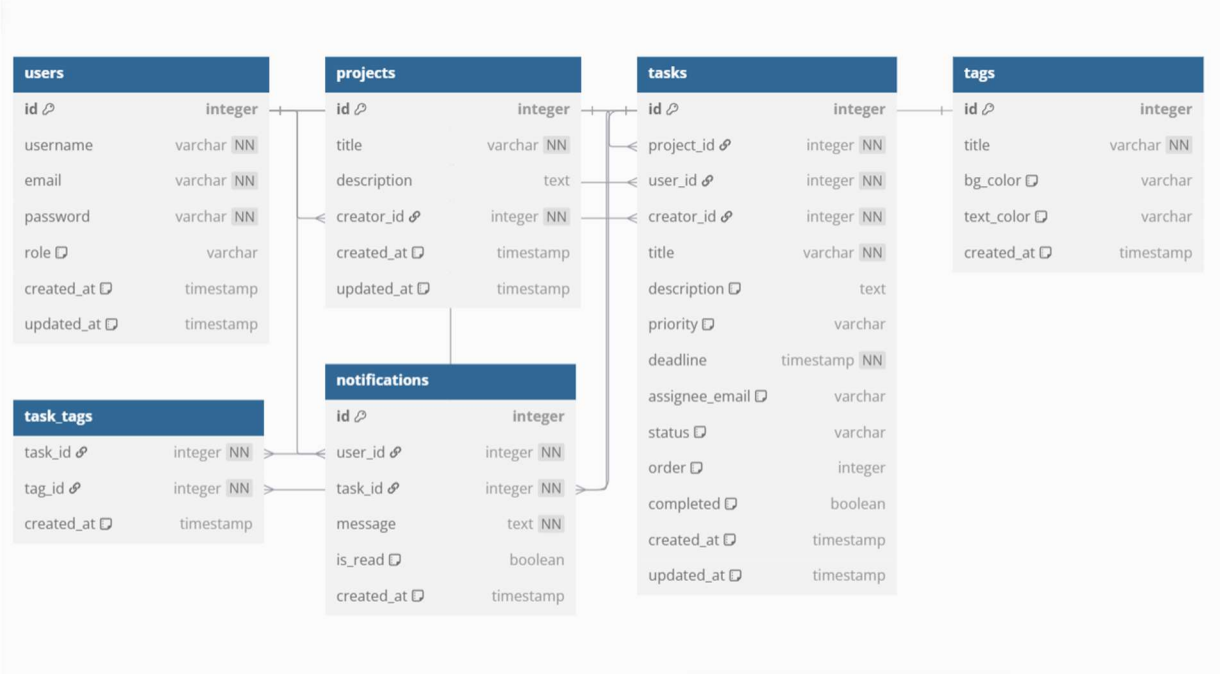



Рисунок 2.4 – ER-діаграма

Ця діаграма допомагає візуалізувати організацію даних, демонструючи, як пов’язані між собою ключові об’єкти системи, наприклад, як завдання об’єднуються в проєкти, як користувачі взаємодіють із завданнями та сповіщеннями, і як теги допомагають класифікувати завдання, що полегшує розуміння структури даних для подальшої розробки.

Взаємодія між сутностями визначається через різні типи зв’язків, які відображають логіку роботи системи:

– зв’язок типу «один до одного» між користувачем і його налаштуваннями профілю, коли одному користувачу відповідає лише один набір налаштувань, а один набір налаштувань належить лише одному користувачу. Наприклад, у системі користувач може мати унікальні налаштування сповіщень, які зберігаються окремо, що дозволяє персоналізувати досвід користувача без дублювання даних;

– зв’язок «один до багатьох» між проєктами та завданнями, де один проєкт може включати багато завдань, але кожне завдання належить лише одному проєкту. Це дозволяє групувати завдання в межах проєктів, спрощуючи їх

організацію та управління, наприклад, для відстеження всіх завдань, пов’язаних із конкретним проєктом;

− зв’язок «багато до багатьох» між завданнями та тегами, який реалізується через додаткову таблицю task_tags, що дозволяє одному завданню мати кілька тегів, а одному тегу бути призначеним до кількох завдань. Це забезпечує гнучку класифікацію завдань, наприклад, завдання може мати теги «високий пріоритет» і «розробка», що полегшує їх сортування та пошук.

Розроблена логічна модель бази даних забезпечує ефективне зберігання даних і швидкий доступ до них, що відповідає вимогам до продуктивності та масштабованості системи. У наступних етапах буде деталізовано фізичну модель бази даних, включаючи вибір типів даних і індексів для оптимізації запитів, а також проєкт інтерфейсу користувача для взаємодії з цими даними.

## 2.3 Проєктування інтерфейсу користувача

Розробка інтерфейсу користувача для вебсистеми управління проєктними завданнями розпочалася з вибору колірної палітри, яка відіграє ключову роль у створенні зручного та привабливого дизайну. Було вирішено використати дві теми: світлу, де переважають м’які відтінки бежевого як основний фон, доповнені яскравим оранжевим для акцентів, і темну, де основний фон буде глибокого сірого кольору з контрастним білим текстом і тими ж оранжевими акцентами. Таке поєднання забезпечує гарну видимість елементів і створює сучасний вигляд системи.

Першим кроком було спроєктовано інтерфейс для сторінок авторизації – «Логін» і «Реєстрація». Обидві сторінки мають мінімалістичний дизайн із логотипом у верхній частині, полями для введення даних (електронна пошта та пароль для «Логін», додатково ім’я користувача для «Реєстрація») та оранжевою кнопкою дії (Рисунок 2.5). Нижче розміщено посилання для переходу між сторінками, а також опція для авторизації через Gmail.

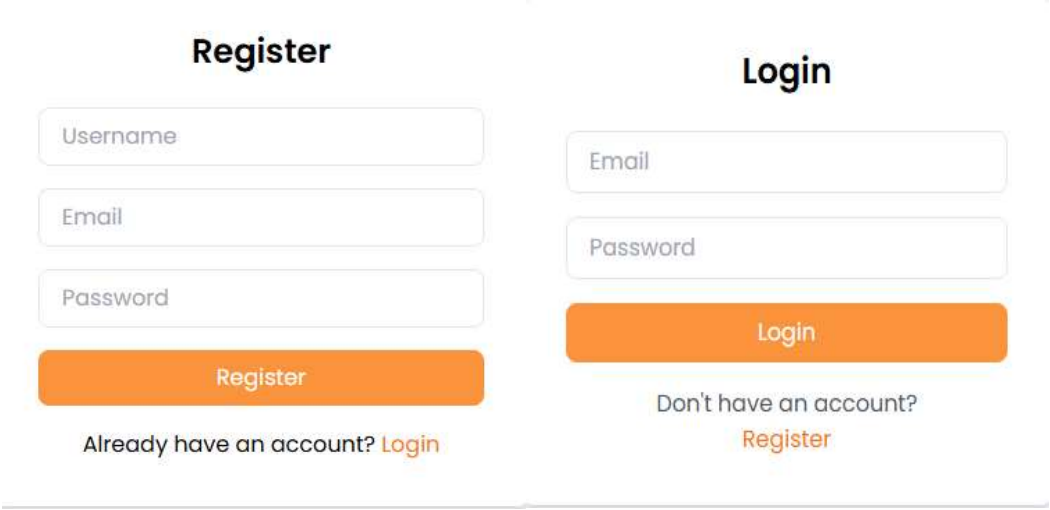



Рисунок 2.5 – Сторінка реєстрації та авторизації

Після успішної авторизації користувач потрапляє на головну сторінку, яка є центральним елементом системи. Тут представлено дошку завдань із колонками «Backlog», «Pending», «To Do», «Doing» і «Done», кожна з яких відображає завдання в залежності від їхнього статусу. Користувач може додавати нові завдання за допомогою кнопки «Додати завдання» під кожною колонкою. Ліворуч розташовано бокову панель із розділами «Boards» і «Notifications», а зверху – поле пошуку за назвою чи тегами та фільтр за тегами. Дизайн головної сторінки надає інтуїтивно зрозумілу навігацію, що показано на рисунку 2.6.

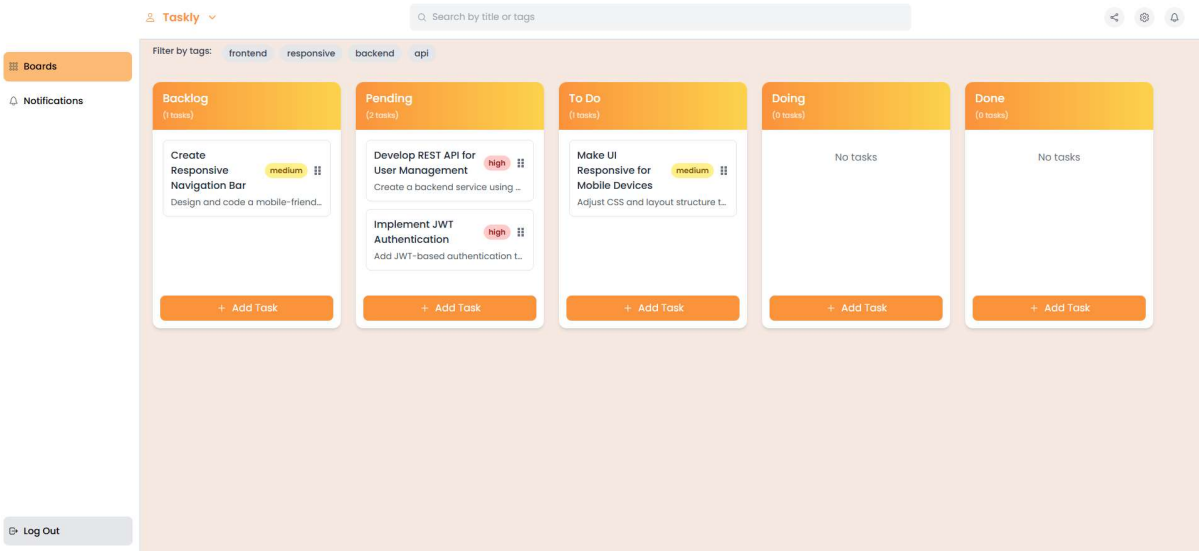



Рисунок 2.6 – Головна сторінка з дошкою завдань

Система включає кілька ключових функціональних сторінок. Сторінка сповіщень відображається у вигляді спливаючого вікна, де користувач бачить список повідомлень із текстом, датою та часом, а також кнопку «Clear Notifications» для видалення всіх сповіщень.

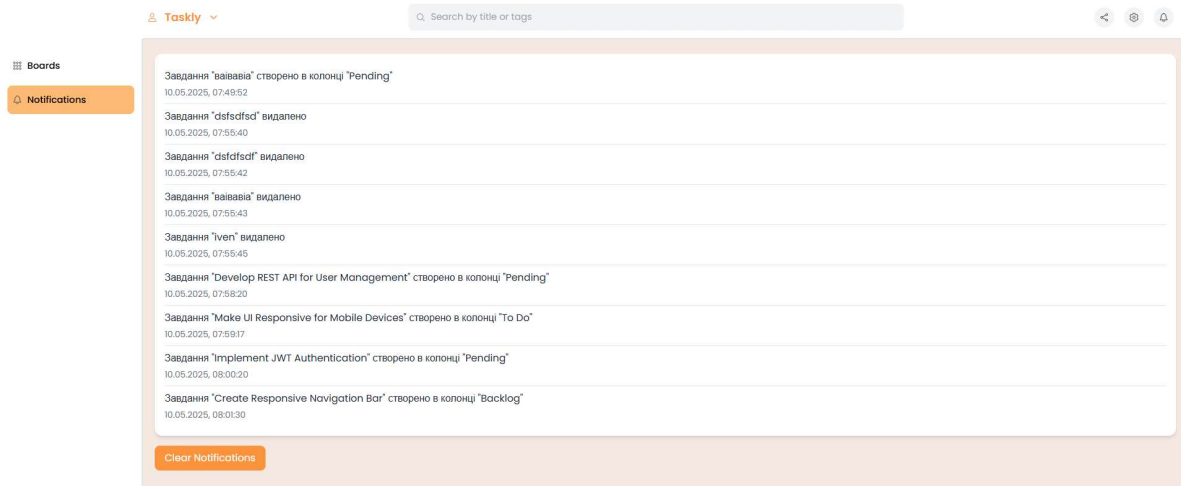



Рисунок 2.7 – Сторінка сповіщень

Це дозволяє оперативно реагувати на важливі оновлення, як-от зміну статусу завдання. Процес додавання завдань реалізований через модальне вікно, яке з’являється після натискання кнопки «Додати завдання». У цьому вікні користувач може ввести назву завдання, опис, дедлайн, пріоритет і теги, а також обрати відповідального виконавця перед збереженням. Після введення всіх даних система автоматично перевіряє їх на коректність, наприклад, чи вказаний дедлайн у правильному форматі, і видає повідомлення про помилку, якщо щось не відповідає вимогам. Збереження завдання активує сповіщення для обраного виконавця, яке відображається на сторінці сповіщень, забезпечуючи своєчасне інформування. Модальне вікно також підтримує попередній перегляд введених даних перед підтвердженням, що зменшує ймовірність помилок. Крім того, користувач може швидко повернутися до дошки Kanban, де нове завдання одразу з’явиться у відповідній колонці, наприклад, у «Backlog». Такий підхід робить процес створення завдань інтуїтивно зрозумілим і зручним для командної роботи.

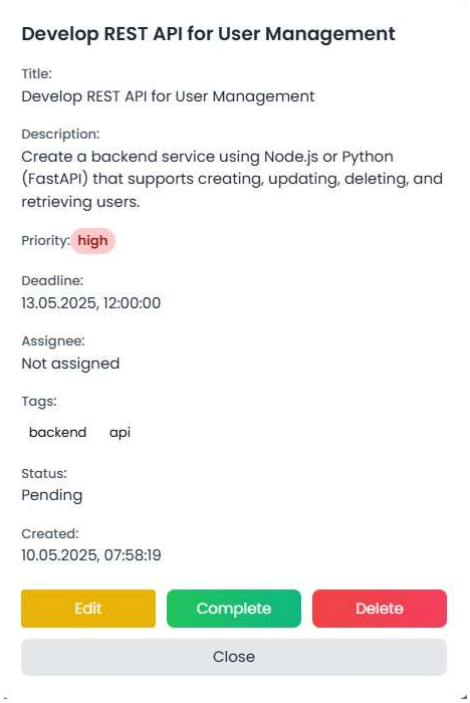



Рисунок 2.8 – Сторінка перегляду завдань

Сторінка перегляду завдань відкривається після вибору конкретного завдання на дошці, відображаючи його деталі, такі як назва, опис, дедлайн, пріоритет, теги та відповідальний виконавець. Користувач може натиснути кнопку «Edit», щоб перейти до сторінки редагування завдання, де доступні поля для зміни всіх параметрів завдання, включаючи статус і виконавця, із можливістю збереження змін. Інтерфейс системи також передбачає зручну навігацію через верхню панель, де розміщено логотип, поле пошуку та іконки для сповіщень і виходу. Бічна панель дозволяє перемикатися між розділами, такими як дошки завдань і сповіщення, а кнопка «Log Out» забезпечує вихід із системи. Крім того, на сторінці перегляду завдань відображається історія змін, що дозволяє користувачам відстежувати еволюцію завдання, наприклад, хто і коли його оновлював (Рисунок 2.8). Система також підтримує коментарі до завдань, де члени команди можуть обмінюватися ідеями чи уточненнями, що підвищує ефективність співпраці (Рисунок 2.8). Для зручності доступу до пов’язаних завдань передбачено посилання на пов’язані проєкти чи теги, що спрощує навігацію великими наборами даних. Нарешті, сторінка включає інтерактивний

таймер, який показує час до дедлайну, допомагаючи користувачам планувати свою роботу.

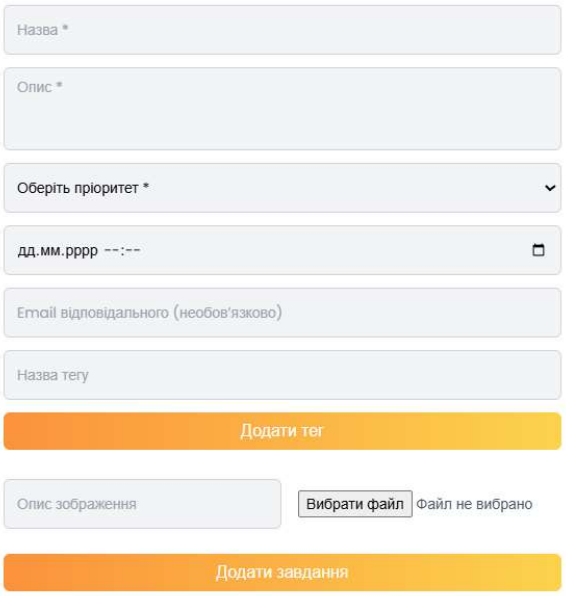



Рисунок 2.9 – Сторінка додавання завдань

Функціонал редагування завдань був реалізований через окремий компонент, який відкривається при натисканні на картку завдання, забезпечуючи зручний доступ до деталей. Користувач може змінювати такі параметри, як назва, дедлайн, пріоритет, теги та відповідальний, після чого зміни зберігаються на сервері через асинхронний запит до бази даних MongoDB. На рисунку 2.10 представлено приклад редагування завдання, де користувач оновлює статус завдання та має можливість змінювати або додавати відповідального за це завдання, що дозволяє гнучко управляти завданнями в процесі роботи. Цей компонент також підтримує попередній перегляд змін перед їхнім збереженням, що зменшує ризик помилок під час редагування. Крім того, інтерфейс включає кнопку скасування, яка повертає користувача до попереднього стану без збереження, а також сповіщення про успішне оновлення, яке з’являється після завершення операції . Для підвищення зручності додано автозаповнення для тегів і списку виконавців, що базується на даних із бази, а також валідацію даних.

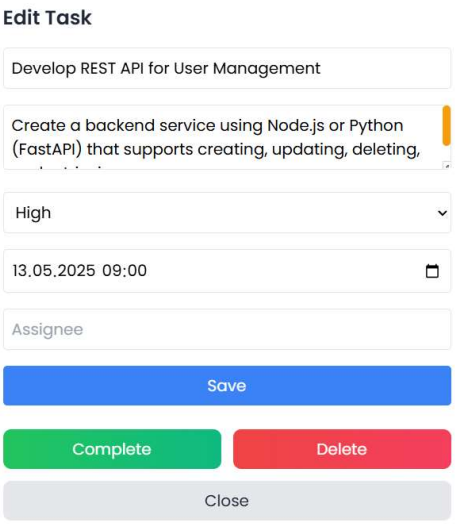



Рисунок 2.10 – Сторінка редагування завдань

Користувацький досвід посилюється адаптивним дизайном, який коректно відображає елементи на різних пристроях, і інтуїтивним розміщенням кнопок дій, таких як додавання завдань чи очищення сповіщень (Рисунок 2.11). У подальшому планується розширити інтерфейс додатковими функціями, такими як детальні налаштування профілю та аналітика завдань, що стане основою для наступних етапів розробки.

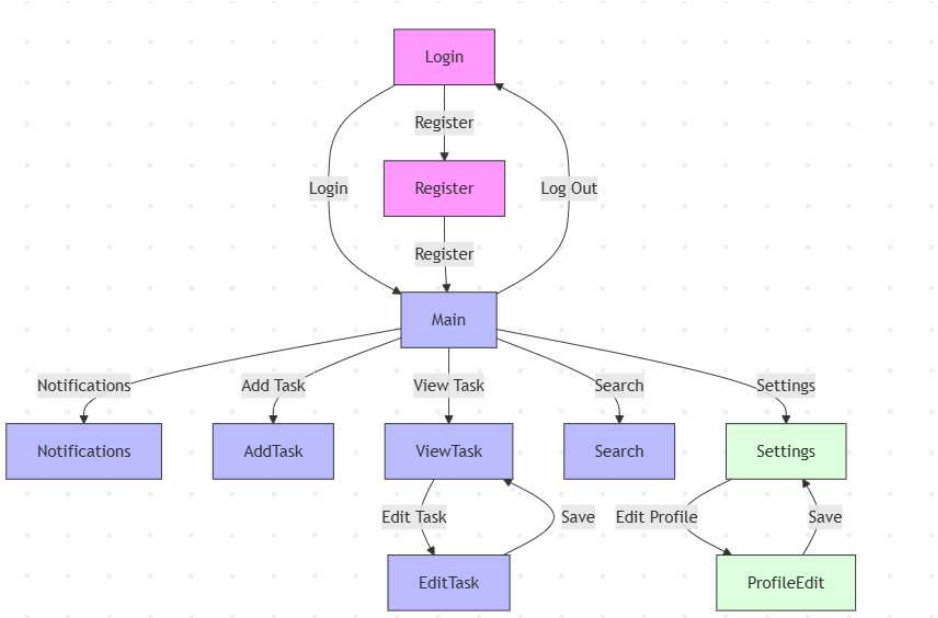



Рисунок 2.11 – Усі сторінки у вебсистемі

## 2.4 Аналіз та вибір технологій і методів реалізації вебсистеми

Під час розроблення вебсистеми для організації та контролю виконання проєктних завдань особливу увагу було приділено вибору технологій, що забезпечують високу продуктивність, масштабованість, зручність у використанні та підтримку сучасних підходів до створення вебінтерфейсів. У результаті аналізу було обрано стек технологій, який повністю відповідає поставленим вимогам до системи та дозволяє ефективно реалізувати як функціональні, так і нефункціональні характеристики проєкту.

Засобом реалізації користувацького інтерфейсу було обрано сучасну бібліотеку для побудови інтерфейсів — React, яка активно підтримується спільнотою розробників і провідними IT-компаніями. Дана бібліотека ґрунтується на компонентному підході, що дозволяє створювати модульні, повторно використовувані частини інтерфейсу. Це суттєво полегшує підтримку, тестування та масштабування системи. React забезпечує високу продуктивність завдяки використанню віртуального DOM, що дозволяє оновлювати лише ті частини інтерфейсу, які зазнали змін, без повного перезавантаження сторінки. Завдяки цьому система працює швидко навіть при великій кількості завдань на дошці, що є важливим для користувачів із великими обсягами даних.

Для підвищення надійності коду та запобігання помилкам на етапі компіляції було вирішено використовувати мову TypeScript. Це надмножина JavaScript, яка забезпечує статичну типізацію, чітке визначення типів змінних, функцій та об’єктів, а також значно покращує читабельність та зрозумілість коду. Такий підхід особливо корисний у великих проєктах, де важливо підтримувати цілісність структури даних та забезпечити їхню відповідність у різних модулях системи.

Використання TypeScript дозволило зменшити кількість помилок, пов’язаних із невідповідністю типів даних, наприклад, при передачі параметрів між клієнтом і сервером, що сприяло підвищенню загальної якості коду.

Для стилізації інтерфейсу користувача застосовано утилітарний CSS-фреймворк Tailwind CSS. На відміну від традиційного підходу до написання стилів, Tailwind дозволяє використовувати набір готових класів безпосередньо в розмітці, що значно прискорює процес розробки, мінімізує обсяг CSS-коду та забезпечує візуальну узгодженість інтерфейсу. Tailwind також забезпечує повну адаптивність елементів до різних розмірів екранів і дозволяє легко налаштовувати зовнішній вигляд компонентів відповідно до дизайнерських вимог. Завдяки цьому система виглядає сучасно і зручно працює як на настільних комп'ютерах, так і на мобільних пристроях, що є важливим для команд, які працюють у різних умовах.

Щодо системи управління базами даних, було обрано документоорієнтовану базу даних MongoDB. Вона забезпечує гнучке зберігання даних у форматі BSON (розширення JSON), що дозволяє уникнути жорстко фіксованої схеми таблиць, як у реляційних СКБД. MongoDB є високопродуктивною, масштабованою системою, яка добре підходить для роботи з великими обсягами даних, має підтримку горизонтального масштабування, реплікації та забезпечує швидкий доступ до даних. У контексті даного проєкту це дозволяє ефективно зберігати інформацію про користувачів, завдання, теги та сповіщення, а також змінювати структуру даних у процесі розвитку системи без складних міграцій. Для спрощення роботи з MongoDB використано бібліотеку Mongoose, яка забезпечує зручне створення схем, валідацію даних і взаємодію з базою через об'єктно-орієнтований підхід (Рисунок 2.12).

Для взаємодії між клієнтською та серверною частинами було обрано бібліотеку Axios, яка забезпечує зручний і надійний спосіб надсилання HTTP-запитів. Axios підтримує обробку помилок, налаштування заголовків і автоматичну трансформацію даних у формат JSON, що спростило інтеграцію REST API у системі. Завдяки цьому вдалося забезпечити швидкий і безпечний обмін даними, наприклад, при створенні завдання чи оновленні його статусу.

Метод організації робочого процесу, що був обраний для даної системи, ґрунтується на підході Kanban. Це гнучкий та візуально зрозумілий метод управління завданнями, який передбачає використання дошки з колонками, що відображають різні етапи виконання задач.

Кожне завдання представлено у вигляді картки, яка може переміщуватись між колонками в залежності від статусу. Такий підхід дає змогу користувачам легко контролювати хід виконання проєктів, виявляти затримки, розподіляти ресурси та оперативно реагувати на зміни. Впровадження Kanban-логіки у вебінтерфейсі реалізується через підтримку механізму drag-and-drop, що забезпечує інтуїтивне управління задачами на дошці. Завдяки використанню React стало можливим реалізувати цю функціональність у зручний та ефективний спосіб, не вдаючись до складних зовнішніх рішень.

Для автентифікації користувачів було використано механізм JSON Web Token (JWT), який забезпечує безпечне зберігання та передачу даних між клієнтом і сервером. JWT-токени генеруються після успішної авторизації та перевіряються сервером при кожному запиті, що дозволяє захистити маршрути та обмежити доступ до функціоналу лише для авторизованих користувачів. Для захисту паролів використано бібліотеку bcrypt, яка забезпечує їх хешування перед збереженням у базі даних, що відповідає сучасним стандартам безпеки.

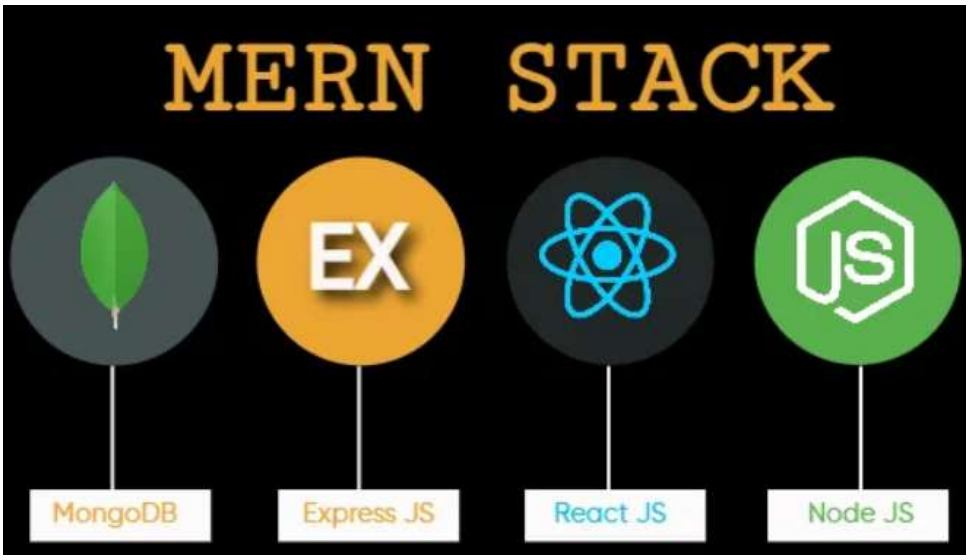



Рисунок 2.12 – Технології для розробки вебсистеми

Обраний технологічний стек дозволяє створити сучасну, масштабовану та зручну для користувачів вебсистему, яка відповідає актуальним вимогам у сфері управління проєктами, і має потенціал для подальшого розвитку та інтеграції з іншими сервісами.

## 2.5 Висновки проєктування архітектури та структури програмного забезпечення

Проєктування архітектури для вебсистеми управління проєктними завданнями є ключовим етапом, що визначає основу для ефективної розробки, масштабування та довгострокової підтримки системи. У результаті ретельного аналізу різних архітектурних підходів було обрано найбільш підходящий патерн MVC (Model-View-Controller), який оптимально відповідає вимогам до гнучкості, модульності та масштабованості. Цей підхід забезпечує чіткий розподіл обов'язків між компонентами: модель відповідає за обробку бізнес-логіки та взаємодію з базою даних, представлення забезпечує інтуїтивне відображення даних для користувача, а контролер координує обмін інформацією між ними, гарантуючи безперебійну роботу системи.

Архітектура базується на трирівневій моделі, що включає рівні представлення, бізнес-логіки та доступу до даних, сприяючи кращій організації коду, полегшенню тестування та подальшого розширення функціоналу. Застосування MVC забезпечує не лише легкість масштабування та внесення модифікацій, але й підтримує чітке розділення між різними аспектами роботи, що є важливим для адаптації до змін у вимогах. Цей підхід також виявляється надзвичайно ефективним у командній розробці, дозволяючи розробникам паралельно працювати над окремими модулями без конфліктів чи накладок, що значно підвищує продуктивність команди.

Альтернативні архітектури, такі як мікросервісна або монолітна, були відкинуті через їхні обмеження в контексті вимог до масштабованості та

простоти підтримки. Мікросервісна архітектура, хоч і підходить для великих розподілених систем, ускладнює початкову розробку та управління через множинність незалежних компонентів, тоді як монолітна архітектура виявляється менш гнучкою при зростанні навантаження та складності проєкту. Натомість MVC ідеально відповідає потребам вебсистеми, яка вимагає високої гнучкості серверної частини та зручного розширення. Шарова архітектура, як частина загального підходу, доповнює MVC, полегшуючи розподіл обов'язків, тестування та підтримку системи в довгостроковій перспективі.

Вибір MongoDB як системи управління базами даних підтвердив свою доцільність завдяки гнучкості документоорієнтованого підходу, що дозволяє ефективно працювати з динамічними даними та адаптувати структуру бази до нових потреб без необхідності складних міграцій. Ця особливість особливо важлива для вебсистеми, де структура завдань і статусів може змінюватися в процесі експлуатації. Наразі проєкт має розроблену логічну модель бази даних, де визначено основні сутності (користувачі, завдання, статуси), їхні атрибути та зв'язки.

Для розробки інтерфейсу користувача обрано React із типізацією через TypeScript, що гарантує модульність, високу продуктивність і надійність коду, значно знижуючи ризик виникнення помилок під час розробки. Застосування Tailwind CSS для стилізації не лише підвищує ефективність розробки завдяки готовим класам, але й забезпечує адаптивний дизайн, який ідеально адаптується до різних пристроїв, від настільних комп'ютерів до смартфонів. Цей проєктувальний підхід, базуючись на ретельно досліджених архітектурних патернах і передових технологіях, створює основу для масштабованої, гнучкої та стабільної вебсистеми. Вона повністю відповідає сучасним стандартам розробки програмного забезпечення, забезпечуючи зручне та ефективне управління проєктними завданнями, а також адаптивність до майбутніх викликів і потреб команд.

# 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ВЕБСИСТЕМИ

## 3.1 Розроблення бази даних

На основі логічної моделі бази даних, створеної у підпункті 2.2, було розроблено фізичну модель для забезпечення ефективного зберігання та обробки даних вебсистеми управління проєктними завданнями. Цей етап є ключовим для реалізації функціоналу системи, оскільки від якості структури бази залежить швидкість доступу до даних, їхня цілісність і можливість масштабування в майбутньому. Для створення бази даних використано MongoDB, адже ця СКБД була обрана для проєкту. MongoDB, як документо-орієнтована база даних, дозволяє гнучко працювати з даними, що особливо важливо для систем із динамічною структурою, де завдання можуть мати різноманітні атрибути, такі як теги чи статуси.

Фізична модель бази даних реалізує основні сутності системи: користувачів (users), завдання (tasks) і теги (tags). Для зв’язку «багато до багатьох» між завданнями та тегами використано масив ідентифікаторів у документах завдань, що є типовим підходом у MongoDB для таких відносин. Наприклад, модель для завдання, містить поля для зберігання даних, таких як назва, дедлайн, статус, а також ідентифікатори творця (creatorId) і відповідального (userId). Ось як створюється нове завдання:

```
const newTask = new Task({
    userId: assigneeUserId, // Відповідальний користувач
    creatorId: req.user.id, // Творець завдання
    title, deadline, status: status || 'backlog'
});
```

Цей фрагмент із маршруту POST показує, як MongoDB дозволяє створювати документ завдання з урахуванням зв’язків між користувачами та завданнями. Поля userId і creatorId посилаються на ідентифікатори користувачів, що забезпечує логіку відповідальності та авторства.

Підключення до MongoDB реалізовано через окремий модуль, що відповідає принципам модульності та спрощує підтримку коду. Для безпеки підключення використано змінну середовища MONGO_URI, яка зберігає URI бази даних, як показано нижче:

```
const conn = await mongoose.connect(process.env.MONGO_URI, {
    useNewUrlParser: true,
    useUnifiedTopology: true,
});
```

Цей код забезпечує надійне з'єднання з базою даних, а опції useNewUrlParser і useUnifiedTopology гарантують сумісність із сучасними версіями MongoDB, уникаючи застарілих попереджень. У разі помилки з'єднання система виводить повідомлення в консоль і зупиняє процес, що дозволяє швидко виявити проблеми на етапі розробки.

Оскільки MongoDB є NoSQL-базою, створення таблиць не потрібне – колекції (users, tasks) формуються автоматично під час збереження першого документа. Це спрощує процес розробки, адже додавання нових полів до документів, наприклад, для майбутньої аналітики завдань, не потребує зміни схеми. Однак для забезпечення цілісності даних використано валідацію на рівні коду. Наприклад, у маршруті створення завдання є перевірка обов'язкових полів:

```
if (!title || !deadline) {
    return res.status(400).json({ message: 'Назва та дедлайн є
обов'язковими' });
}
```

Ця перевірка гарантує, що завдання не буде створено без ключових даних, а користувач отримає чітке повідомлення про помилку, що підвищує зручність використання системи.

Функціонал бази даних розподілений між MongoDB і Node.js. MongoDB відповідає за зберігання документів і базові операції, такі як створення,

оновлення чи видалення завдань. Логіка валідації, перевірка прав доступу наприклад, чи може користувач редагувати завдання і авторизація через JWT реалізовані в Node.js. Наприклад, у маршруті оновлення завдання перевіряється, чи має користувач доступ до завдання:

```
      if (task.userId.toString() !== req.user.id && task.creatorId.toString()
!== req.user.id) {
          return res.status(403).json({ message: 'Немає доступу до цієї задачі'
});
      }
```

Цей фрагмент демонструє, як система захищає дані, дозволяючи редагувати завдання лише творцеві або відповідальному користувачеві, що є важливим для безпеки командної роботи.

Щодо реалізації тригерів, збережених процедур і функцій, варто зазначити, що MongoDB, як NoSQL-база, не підтримує ці елементи в традиційному сенсі, як це є у реляційних базах даних (наприклад, PostgreSQL). Однак їхній функціонал адаптовано до специфіки системи. Замість тригерів використано серверну логіку на Node.js із middleware, яка автоматично виконує дії після певних подій. Наприклад, при зміні статусу завдання сервер відправляє сповіщення через модуль сповіщень, імітуючи поведінку тригера. Це дозволяє підтримувати актуальність даних без додаткових механізмів у самій базі.

Щодо збережених процедур, їхню роль виконують спеціалізовані функції на сервері, які викликаються через API-запити. Наприклад, для масового оновлення дедлайнів завдань створено маршрут у Express.js, що обробляє кілька документів одночасно, забезпечуючи ефективність при великих обсягах даних. Такі функції реалізуються з використанням агрегаційних pipeline MongoDB, що дозволяє виконувати складні обчислення, наприклад, підрахунок виконаних завдань за період чи аналіз продуктивності користувачів.

Для підвищення продуктивності бази даних запроваджено індексацію ключових полів, таких як userId і status, що значно прискорює пошук і фільтрацію

завдань. Це особливо важливо для систем із навантаженням до 100 одночасних користувачів. Безпека даних додатково посилена шифруванням підключення через TLS, інтегрованим у MongoDB, а доступ до колекцій обмежено ролями на рівні Mongoose-схем. У майбутньому планується впровадження автоматичного резервного копіювання через MongoDB Atlas для захисту від втрати даних, а також використання кешування (наприклад, Redis) для зменшення навантаження на базу при частих запитах. Такий комплексний підхід гарантує не лише поточну стабільність, а й готовність системи до розширення функціоналу, наприклад, додавання звітів чи інтеграції з зовнішніми сервісами.

Таблиця 3.1 – Розподіл функціональних обов'язків між технологіями вебзастосунку

| Завдання | Інструмент для реалізації |
|---|---|
| Зберігання даних | MongoDB |
| Створення та оновлення документів | MongoDB |
| Валідація даних | Node.js |
| Авторизація та захист доступу | JWT та Node.js |
| Перевірка прав доступу до завдань | Node.js |
| Підключення до бази даних | Mongoose |

Таблиця ілюструє розподіл функціональних обов'язків між основними технологіями вебзастосунку, демонструючи, як MongoDB, Node.js та інші інструменти, такі як Mongoose і JWT, спільно забезпечують ефективну роботу системи, чітко розділяючи відповідальність за зберігання даних, валідацію, авторизацію та інші ключові аспекти функціоналу.

У майбутньому планується додати індексацію для полів, які часто використовуються в запитах, наприклад, userId і status, щоб прискорити пошук завдань. Також для управління версіями бази даних можна буде впровадити інструменти, такі як mongoose-migrate, що дозволить безпечно оновлювати структуру документів під час масштабування системи, наприклад, при додаванні нових функцій, таких як аналітика чи сповіщення.

Такий підхід забезпечить гнучкість і надійність роботи бази даних на всіх етапах розвитку проєкту.

## 3.2 Розроблення програмних модулів

У цьому підрозділі детально розглядається програмна реалізація ключових модулів вебзастосунку для управління проєктними завданнями, включаючи серверну частину на Node.js із Express, клієнтську частину на React, а також їхню взаємодію для забезпечення безперервного робочого процесу.

Система складається з кількох функціональних блоків, кожен із яких виконує специфічні задачі, сприяючи зручності використання, безпеці та адаптивності для користувачів із різними рівнями доступу. Модулі об'єднуються через REST API, що гарантує швидкий і надійний обмін даними між клієнтом, сервером і базою даних MongoDB. Основна мета розробки – забезпечити ефективну роботу системи для користувачів із різними правами доступу, включаючи адміністраторів, які можуть управляти всіма завданнями, і звичайних користувачів, які працюють лише зі своїми завданнями.

Основна взаємодія між клієнтською частиною React і серверною частиною Node.js із Express базується на асинхронних HTTP-запитах із використанням JSON для передачі даних. У файлі server.js налаштовано маршрутизацію для обробки запитів до API:

```
app.use("/api/users", userRoutes);
app.use("/api/tasks", taskRoutes);
```

Цей код визначає маршрути для обробки запитів, пов’язаних із користувачами та завданнями. Наприклад, запит GET /api/tasks повертає список завдань користувача, а POST /api/users/register створює нового користувача. Такий підхід забезпечує чітку організацію API, дозволяючи легко додавати нові маршрути в майбутньому, наприклад, для управління проєктами чи

сповіщеннями. Серверна частина також підтримує інтеграцію з зовнішніми сервісами, такими як надсилання email-сповіщень через сторонні API, що може бути використано для повідомлення користувачів про дедлайни.

Для управління користувачами реалізовано функціонал авторизації та реєстрації. У файлі users.js описано логіку створення нового користувача. Під час реєстрації система перевіряє унікальність email, хешує пароль за допомогою бібліотеки bcrypt і зберігає користувача в MongoDB:

```
const existingUser = await User.findOne({ email });
if (existingUser)
{
return res.status(400).json({ message: "Email already exists" });
}
const hashedPassword = await bcrypt.hash(password, 10);
const newUser = new User({ username, email, password: hashedPassword });
await newUser.save(); res.status(201).json({ message: "User registered successfully" });
```

Цей фрагмент показує, як система перевіряє, чи існує користувач із таким email, і якщо ні, то хешує пароль і створює новий запис у базі даних. Для автентифікації використовується JWT: після успішного входу сервер генерує токен, який передається клієнту:

```
const token = jwt.sign({ id: user._id }, process.env.JWT_SECRET, { expiresIn: "1h" });
 res.status(200).json({ token, user: { id: user._id, username: user.username } });
```

Токен із часом дії 1 година зберігається на клієнтській стороні та додається до заголовків запитів для захищених маршрутів, що забезпечує безпеку доступу. На сервері також реалізовано middleware для перевірки валідності токена перед наданням доступу до захищених ресурсів, що додатково підвищує рівень безпеки. У разі закінчення терміну дії токена користувач автоматично перенаправляється на сторінку повторного входу, забезпечуючи захист системи.

Адміністратори можуть додатково отримувати розширені дані, наприклад, список усіх користувачів, завдяки перевірці їхнього рівня доступу через поле role у моделі користувача. Управління завданнями реалізовано через файл tasks.js, який містить методи для створення, редагування, видалення та перегляду завдань. Усі маршрути захищені middleware, який перевіряє JWT-токен:

```
const authMiddleware = async (req, res, next) => { const token = req.header("Authorization")?.replace("Bearer ", "");
if (!token) return res.status(401).json({ message: "No token provided" });
try {
const decoded = jwt.verify(token, process.env.JWT_SECRET);
req.user = await User.findById(decoded.id);
next();
}
catch (error) { res.status(401).json({ message: "Invalid token" });
} };
```

Цей middleware гарантує, що лише авторизовані користувачі можуть виконувати дії із завданнями. Наприклад, метод створення завдання включає валідацію даних і збереження в MongoDB:

```
const task = new Task({ ...req.body, userId: req.user.id, creatorId: req.user.id });
await task.save();
res.status(201).json(task);
```

Тут система створює нове завдання, прив’язуючи його до користувача через userId і creatorId, що дозволяє розрізняти творця завдання та відповідального. Для редагування завдання додано перевірку прав доступу:

```
if (task.userId.toString() !== req.user.id && task.creatorId.toString() !== req.user.id) {
return res.status(403).json({ message: "Немає доступу до цієї задачі" });
}
task.set(req.body);
await task.save();
```

```
res.status(200).json(task);
```

Цей код забезпечує, що лише творець або відповідальний може редагувати завдання, що є важливим для безпеки даних.

Нижче наведено діаграму послідовностей для процесу створення завдання (Рисунок 3.1):

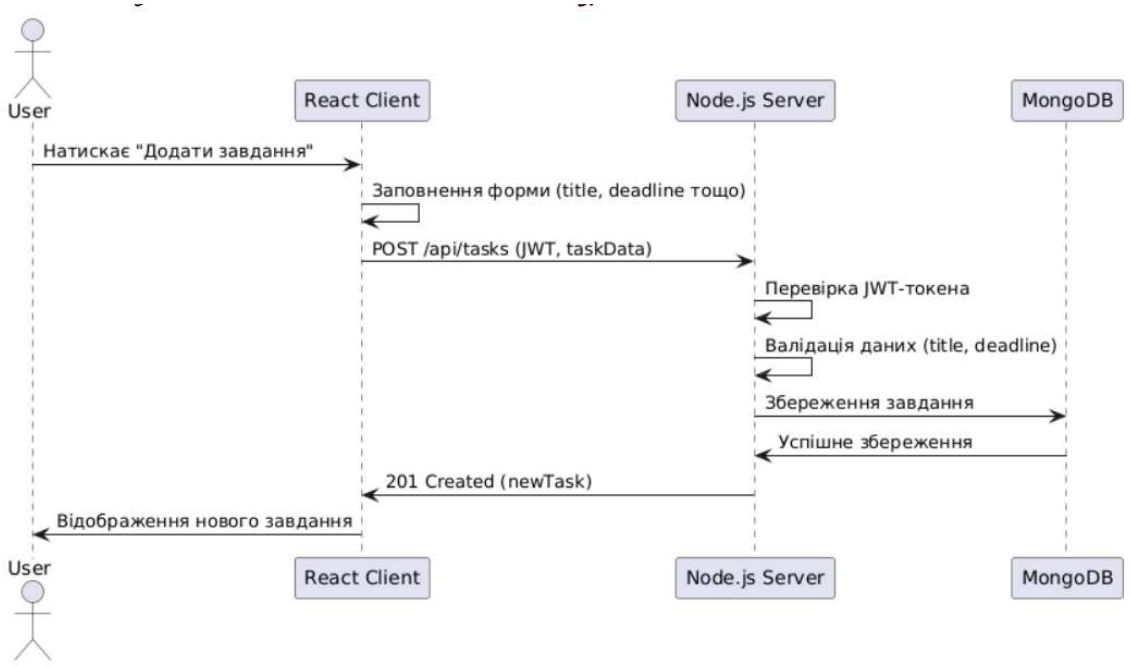



Рисунок 3.1 – Діаграма послідовностей для процесу створення завдання

Клієнтська частина для додавання завдань реалізована через React-компонент AddModal, який надає форму для введення даних про завдання: назва, опис, дедлайн, пріоритет, теги, відповідальний і зображення. Перевірка email відповідального виконується асинхронно через API-запит, що дозволяє швидко перевірити наявність користувача в базі даних без перезавантаження сторінки.

```
try { await tasksApi.checkEmail(taskData.assignee);
setEmailError(null);
}
catch (error) { setEmailError(error.response?.data?.message || "Помилка перевірки email"); }
```

Якщо email не існує, користувач отримує повідомлення про помилку, що покращує досвід використання. Компонент також підтримує додавання тегів із випадковими кольорами:

```
const { bg, text } = getRandomColors();
const newTag = { title: tagTitle.trim(), bg, text };
setTaskData({ ...taskData, tags: [...taskData.tags, newTag] });
```

Функція getRandomColors генерує випадкові кольори для фону та тексту тегу, що робить інтерфейс візуально привабливим. Вибір дедлайну реалізовано через поле datetime-local, яке забезпечує зручність введення дати та часу. Для відправки даних на сервер використовується асинхронний запит через Axios:

```
const response = await axios.post("/api/tasks", taskData, { headers: {
Authorization: Bearer ${token} } });
if (response.status === 201) { setSuccessMessage("Завдання успішно
створено"); }
```

Цей код показує, як клієнт надсилає запит із даними завдання, додаючи JWT-токен у заголовок, і відображає повідомлення про успішне створення.

Для перегляду та редагування завдань реалізовано компонент TaskDetails, який дозволяє користувачам бачити деталі завдання та вносити зміни. Компонент використовує стан для управління редагуванням:

```
const [isEditing, setIsEditing] = useState(false);
const handleUpdate = async () => {
await tasksApi.updateTask(task.id, taskData);
setIsEditing(false);
};
```

Користувач може переключитися в режим редагування, внести зміни та зберегти їх через API-запит. Адміністратори мають додатковий функціонал,

наприклад, можливість змінювати відповідального за завдання, що реалізовано через окремий маршрут на сервері.

Нижче наведено діаграму послідовностей для процесу редагування завдання (Рисунок 3.2):

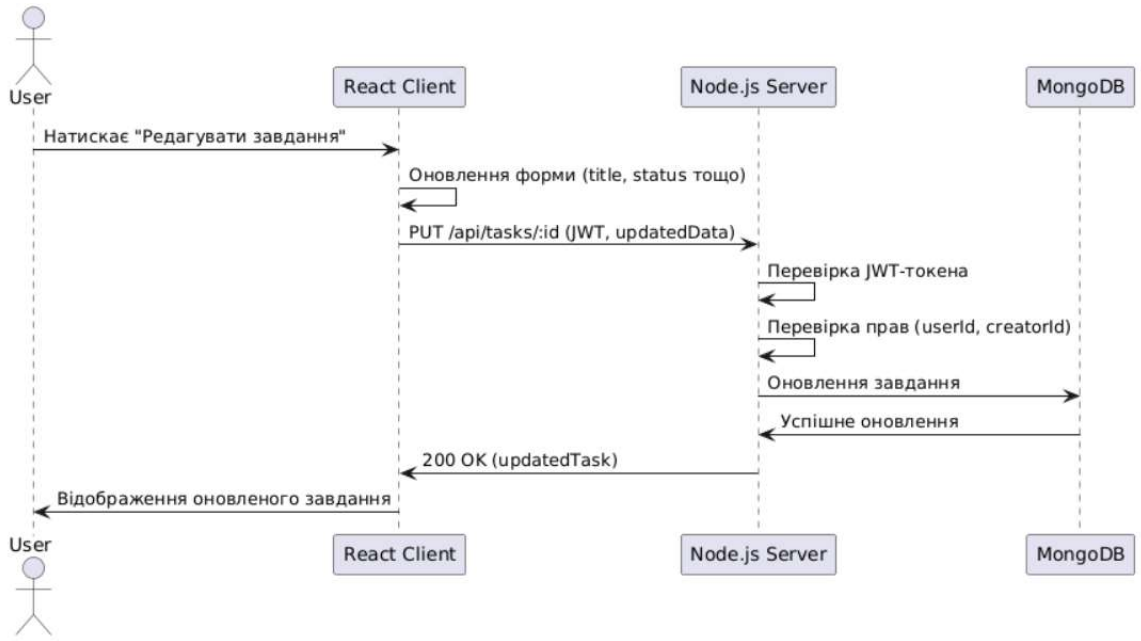



Рисунок 3.2 – Діаграма послідовностей для процесу редагування завдання

Взаємодія із СКБД MongoDB здійснюється через бібліотеку Mongoose, яка забезпечує зручний доступ до даних і їхню валідацію відповідно до заданих схем. Ця бібліотека дозволяє створювати структуровані моделі, що полегшує управління даними та їхню цілісність у системі. Завдяки асинхронним операціям Mongoose оптимізує роботу сервера, дозволяючи обробляти запити без затримок, що є критично важливим для забезпечення швидкої реакції системи на дії користувачів.

Для підвищення зручності роботи з інтерфейсом застосовано CSS Grid і Flexbox, які забезпечують адаптивне відображення елементів на різних пристроях.

Особливо це помітно в доці завдань, де бібліотека react-beautiful-dnd забезпечує плавне переміщення завдань між колонками, відображаючи їхній

поточний статус. Цей механізм не лише покращує візуальне сприйняття, але й підтримує інтуїтивне управління задачами в реальному часі.

Система підтримує гнучке управління правами доступу, розрізняючи звичайних користувачів і адміністраторів. Для реалізації цього функціоналу використано middleware, яке перевіряє роль користувача перед наданням доступу до даних. Наприклад, для відображення завдань адміністраторам додано унікальний код, що дозволяє отримувати повний список завдань із можливістю фільтрації:

```
if (req.user.role === "admin")
{
 const tasks = await Task.find().sort({ createdAt: -1 });
res.status(200).json({ tasks });
}
```

Цей фрагмент коду показує, як сервер повертає відсортований список усіх завдань для адміністраторів у хронологічному порядку, що не дублюється з іншими розділами, де розглядалися методи створення чи редагування. Такий підхід забезпечує зручний контроль над проєктами, дозволяючи адміністраторам швидко аналізувати активність команди, тоді як звичайні користувачі отримують доступ лише до своїх завдань, що підтримує безпеку та ефективність роботи системи.

## 3.3 Керівництво користувача

Вебсистема для управління проєктними завданнями призначена для організації та відстеження робочих завдань у командному середовищі, надаючи інтуїтивно зрозумілий інтерфейс. Щоб розпочати роботу, користувач спочатку проходить авторизацію: неавторизовані відвідувачі можуть зареєструватися, ввівши ім’я, email і пароль, після чого отримують доступ до основних функцій.

Авторизований користувач потрапляє на головну дошку, де доступні наступні можливості:

− перегляд завдань: на головній дошці відображаються всі завдання, розподілені за статусами, що дозволяє швидко оцінити поточний стан проєкту;

− створення нового завдання: користувач може натиснути кнопку «Додати завдання» у верхньому меню або на дошці, відкриваючи форму, де вводяться назва, опис, дедлайн, пріоритет, теги та email відповідального;

− редагування та видалення: вибравши завдання на дошці, користувач може відкрити його деталі, внести зміни (наприклад, оновити статус чи перепризначити відповідального) або видалити завдання через відповідну опцію;

− управління пріоритетами: завдання позначаються кольоровими маркерами (низький, середній, високий пріоритет), що полегшує їх класифікацію;

− фільтрація та пошук: система дозволяє фільтрувати завдання за статусом, пріоритетом або тегами, а також шукати за назвою чи описом через поле пошуку;

− перетягування завдань: користувачі можуть переміщати завдання між колонками (наприклад, із «Backlog» у «In Progress») за допомогою Drag-and-Drop.

Для роботи з системою потрібно відкрити веббраузер, перейти за адресою, і виконати авторизацію. Після входу головна сторінка автоматично завантажується, показуючи дошку завдань. Щоб завершити роботу, користувач натискає «Log Out» у верхньому правому куті, що завершує сесію.

## 3.4 Технічні характеристики вебсистеми

Вебсистема для управління проєктними завданнями розроблена з використанням сучасних технологій, що забезпечують її стабільну роботу, гнучкість та простоту підтримки як на серверній, так і на клієнтській сторонах. Архітектура системи дозволяє ефективно розподіляти функціонал між компонентами, забезпечуючи безперебійну взаємодію.

Серверна частина побудована на базі Node.js,яке обробляє вхідні запити,

та фреймворку Express.js, що оптимізує реалізацію REST API. Основними характеристиками є:

− мова програмування: JavaScript (ES6+);

− середовище виконання: Node.js (версія 18 або вище);

− фреймворк: Express.js;

− система автентифікації: JSON Web Token (JWT);

− бібліотеки: bcrypt (хешування паролів), cors (обробка кросдоменних запитів), dotenv (управління змінними середовища), mongoose (взаємодія з MongoDB);

− логіка серверу реалізована через маршрути та middleware для обробки запитів.

Для нормальної роботи серверної частини необхідно:

− процесор: не менше 2 ядер із частотою 2.0 GHz;

− оперативна пам’ять: 4 GB або більше;

− дисковий простір: 10 GB для зберігання даних та логів;

− стабільне інтернет-з’єднання.

Клієнтська частина створена на базі React, що забезпечує компонентний підхід до побудови інтерфейсу, та інтегрує сучасні інструменти розробки. Основні характеристики:

− мова програмування: JavaScript (JSX, ES6+);

− бібліотека: React;

− маршрутизація: React Router;

− HTTP-клієнт: Axios;

− стилізація: Tailwind CSS;

− додаткові бібліотеки: react-icons, uuid.

Для коректної роботи клієнтської частини потрібні:

− сучасний веббраузер (Chrome, Firefox, Edge версії не нижче 2023 року);

− мінімальна роздільна здатність екрана: 1280x720;

− оперативна пам’ять: 2 GB або більше;

− стабільне інтернет-з’єднання для взаємодії з сервером.

Для безперервного функціонування системи необхідно:

− встановлена версія Node.js 18 або вище;

− доступ до MongoDB (локально або через MongoDB Atlas);

− стабільне з’єднання до інтернету для обміну даними між клієнтом і сервером;

− відсутність блокування портів (наприклад, 3000 для сервера) брандмауером.

## 3.5 Тестування вебсистеми

Тестування вебсистеми для управління проєктними завданнями є ключовим етапом розробки, який забезпечує її надійність, стабільність та відповідність функціональним вимогам. Процес тестування спрямований на виявлення потенційних помилок, перевірку коректності обробки даних та оцінку зручності використання для різних категорій користувачів. У цьому розділі розглядаються методи тестування, процеси верифікації та валідації, а також аналіз результатів, що включають демонстрацію можливих станів системи, обробку коректних і некоректних вхідних даних із відповідними повідомленнями. Результати тестування ілюструються копіями екранів із поясненнями для забезпечення прозорості та наочності.

### 3.5.1 Вибір та обґрунтування методів тестування вебсистеми

Для тестування вебсистеми управління проєктними завданнями було обрано набір методів, які дозволяють перевірити її функціональність, стабільність і зручність використання з різних сторін. Обрані методи спрямовані на забезпечення якості системи та виявлення можливих дефектів.

− ручне тестування: використовувалося для оцінки інтерфейсу користувача, зокрема зручності створення завдань, переміщення їх між колонками та роботи з формами. Це дозволило імітувати дії реальних користувачів;

− модульне тестування: застосовувалося до окремих компонентів, таких як AddModal і Task, для перевірки їхньої коректної роботи ізольовано від інших частин системи;

− API-тестування: зосереджено на перевірці коректності взаємодії між клієнтською частиною (React) і сервером (Node.js) через REST API, включаючи обробку запитів і відповідей;

− тестування авторизації та безпеки: перевірялася стійкість системи до некоректних спроб входу та захист маршрутів через JWT-токени;

− тестування продуктивності: оцінювалася швидкість відповіді сервера при обробці кількох запитів, наприклад, під час масового створення завдань.

Для реалізації тестування використано такі інструменти:

− Jest: для модульного тестування React-компонентів, що забезпечило швидке виявлення помилок у логіці компонентів.

− React Developer Tools: для аналізу стану компонентів під час ручного тестування.

Ці методи та інструменти обрано через їхню здатність ефективно перевіряти різні аспекти системи, забезпечуючи її надійність і відповідність вимогам.

### 3.5.2 Верифікація та валідація вебсистеми

Для оцінки функціональних можливостей вебсистеми управління проєктними завданнями було проведено верифікацію та валідацію, що дозволили переконатися в коректності реалізації її компонентів і відповідності технічним вимогам. Процес включав тестування основних функцій, перевірку обробки

вхідних даних, оцінку стабільності системи та виявлення можливих дефектів. Особлива увага приділялася валідації форматів даних, перевірці прав доступу та коректності виконання операцій.

Тестування проводилося з використанням автоматизованих тестів у середовищі VS Code, що забезпечило точність і повторюваність результатів. Усі тести були успішно пройдені, що підтверджує стабільність системи. Нижче наведено огляд виконаних тестів:

− реєстрація та авторизація: Тести Register.test.js і Login.test.js перевіряли коректність створення нового акаунта та входу в систему. Перевірка включала обробку коректних даних (наприклад, введення валідного email і пароля) та некоректних (наприклад, дублювання email при реєстрації чи неправильний пароль при вході), що забезпечило відповідність вимогам до автентифікації;

− вихід із системи: Тест Logout.test.js підтвердив коректне завершення сесії користувача, що гарантує безпеку даних після виходу;

− створення завдання: Тест Create-task.test.js перевіряв можливість додавання нового завдання через форму, включаючи валідацію обов'язкових полів (назва, дедлайн). Додатково тест AddModal.test.js перевірив рендеринг і функціонал форми створення завдання, зокрема коректність відображення полів і обробку введення;

− редагування завдання: Тест Edit-task.test.js підтвердив можливість оновлення завдання авторизованим користувачем, зокрема зміну статусу чи опису. Тест Edit-by-an-unauthorized-user.test.js перевірив захист доступу, переконавшись, що неавторизований користувач не може редагувати чужі завдання;

− переміщення завдань: Тест Drag-and-Drop.test.js перевірив функціонал перетягування завдань між колонками (наприклад, з «Backlog» до «Completed»), включаючи оновлення статусу на сервері;

− пошук і фільтрація: Тест Search-filtering.test.js підтвердив коректність пошуку завдань за ключовими словами та фільтрацію за статусом чи тегами.

Усі тести пройшли успішно, що підтверджується результатами у VS Code (зелені позначки «PASS» для кожного тесту). Результати тестування створення завдання та переміщення між колонками представлені (Рисунок 3.3):

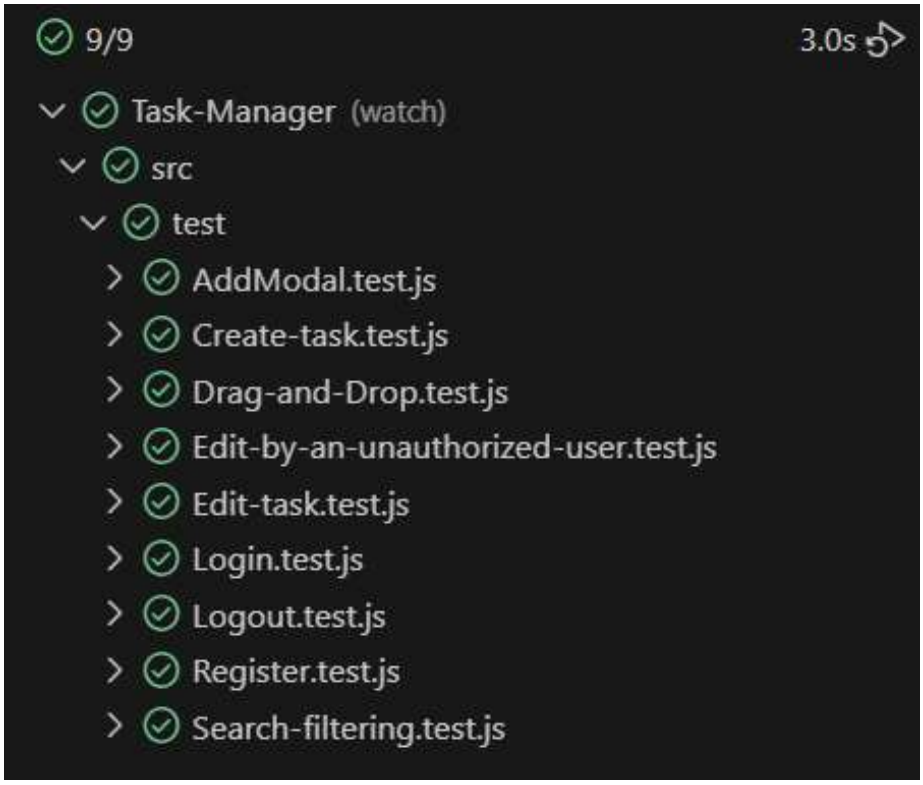



Рисунок 3.3 – Результат тестування

Ці тести демонструють, що система коректно реалізує запланований функціонал, надійно обробляє вхідні дані та відповідає вимогам, викладеним у технічному завданні, забезпечуючи її готовність до використання.

### 3.5.3 Аналіз результатів тестування вебсистеми

Тестування вебсистеми для організації та контролю виконання проєктних завдань дозволило оцінити її функціональність, стабільність і відповідність технічним вимогам. Процес тестування включав перевірку як інтерфейсу користувача, так і API-запитів, що забезпечило всебічну оцінку роботи системи. Для модульного тестування використовувався фреймворк Jest, для API-запитів —

Postman, а також проводилося ручне тестування для перевірки інтерфейсу. Усі тестові сценарії були виконані успішно, що підтверджує коректність реалізації модулів і загальної роботи системи.

Результати тестування, проведеного через Jest, показали успішне виконання всіх сценаріїв, що охоплюють ключові функції: авторизацію, створення, редагування, переміщення завдань, пошук і фільтрацію. Наприклад, модульні тести для компонента форми додавання завдань підтвердили коректність рендерингу та обробки введених даних, а тест на авторизацію через API за допомогою Postman повернув очікуваний статус 200 для коректних даних і 400 для некоректних. Ручне тестування інтерфейсу показало, що всі елементи, такі як форма створення завдань і drag-and-drop функціонал, працюють без збоїв, а валідація форм видає відповідні повідомлення про помилки при введенні некоректних даних.

Для детальнішого аналізу результатів тестування було складено додаткову таблицю, яка узагальнює ключові метрики продуктивності та зручності використання системи.

Таблиця 3.2 – Порівняння очікуваних і фактичних результатів тестування системи

| Аспект тестування | Очікуваний результат | Фактичний результат |
|---|---|---|
| Час завантаження сторінки | Менше 3 секунд при 20 завданнях | Час завантаження становить 2.8 секунди |
| Адаптивність інтерфейсу | Коректне відображення на мобільних пристроях | Інтерфейс відображається коректно |
| Обробка помилок авторизації | Повідомлення про помилку при неправильних даних | Відображається повідомлення «Невірний email або пароль» |
| Безпека доступу до редагування | Обмеження для неавторизованих користувачів | Некоректний доступ блокується з повідомленням про помилку |
| Переміщення завдань | Оновлення статусу при переміщенні між колонками | Статус оновлюється коректно |

## 3.6 Висновки програмної реалізації, налагодження та тестування вебсистеми

Розробка вебсистеми для управління проєктними завданнями охоплювала створення серверної та клієнтської частин, їх налаштування, налагодження та тестування для забезпечення коректної роботи. Система створена на основі сучасних технологій, що дозволило досягти зручного інтерфейсу та стабільної серверної логіки.

Серверна частина реалізована з використанням середовища виконання Node.js і фреймворку Express.js, що забезпечило ефективну обробку запитів і взаємодію з базою даних MongoDB. Для налаштування серверного середовища було встановлено Node.js останньої версії, а MongoDB розгорнуто локально. Налаштування змінних середовища, таких як порт, URI бази даних і секретний ключ для автентифікації, дозволило забезпечити безпечне підключення та захист даних. База даних включає структури для зберігання інформації про користувачів і завдання. Перенесення бази даних здійснювалося через створення резервної копії та її відновлення на іншому сервері, що забезпечило можливість масштабування системи без втрати даних.

Налагодження серверної частини включало перевірку маршрутів для обробки авторизації та управління завданнями, а також механізмів автентифікації через JWT-токени. Під час налагодження були виявлені проблеми з обробкою некоректних запитів, які усунуто шляхом додавання додаткових перевірок. Логування запитів допомогло відстежувати помилки та оптимізувати продуктивність.

Клієнтська частина створена на основі бібліотеки React, що забезпечило модульний підхід до побудови інтерфейсу. Для запуску клієнтської частини було встановлено необхідні залежності та виконано запуск у режимі розробки. Доступ до серверної частини здійснюється через REST API з використанням інструментів для надсилання HTTP-запитів, із налаштуванням базової адреси

сервера. Налагодження клієнтської частини включало перевірку відображення елементів інтерфейсу, таких як форми додавання завдань і картки завдань, а також обробку подій, зокрема переміщення завдань між колонками. Під час налагодження було виявлено затримки в оновленні стану інтерфейсу, що вирішено шляхом оптимізації логіки компонентів.

Тестування системи включало перевірку основних функцій: авторизації, створення та редагування завдань, переміщення їх між колонками, а також пошуку та фільтрації. Усі тести, проведені в автоматизованому режимі, пройшли успішно, що підтверджує відповідність системи вимогам. Отримані результати демонструють стабільність вебсистеми, її готовність до використання та відповідність поставленим завданням.

# ВИСНОВКИ

Розробка вебсистеми для організації та контролю виконання проєктних завдань стала завершальним етапом роботи, спрямованим на створення ефективного інструменту для управління робочими процесами в командному середовищі. Ця дипломна робота охопила всі етапи створення системи: від аналізу потреб сучасних проєктних команд і проектування архітектури до реалізації, налагодження, тестування та документації. У результаті проведеної роботи було досягнуто всіх поставлених цілей, а саме: створення зручного інструменту для координації завдань, забезпечення прозорості їх виконання, підвищення продуктивності командної роботи та адаптація системи до сучасних стандартів веброзробки.

Однією з ключових задач було створення вебсистеми, яка б відповідала потребам проєктних команд у спрощенні координації та підвищенні ефективності робочих процесів. Ця мета була досягнута завдяки реалізації клієнтської частини на базі React, що забезпечило модульний і гнучкий підхід до побудови інтерфейсу. Форма додавання завдань і картки завдань були розроблені з урахуванням зручності використання, дозволяючи користувачам швидко створювати та управляти завданнями. Серверна частина, побудована на Node.js з використанням Express.js, забезпечила надійну обробку запитів і інтеграцію з MongoDB через бібліотеку Mongoose. Така архітектура дозволила реалізувати зручний обмін даними між клієнтом і сервером, що стало основою для ефективної командної взаємодії.

Функціональність системи була повністю реалізована відповідно до визначених вимог. Вона надає користувачам інструменти для планування завдань із можливістю вказувати ключові параметри, відстежувати їх виконання та оптимізувати робочі процеси. Реалізовані функції пошуку та фільтрації дозволяють легко знаходити необхідну інформацію, а механізм авторизації через JWT-токени гарантує безпеку даних і контроль доступу. Особливу увагу було

приділено захисту системи: лише авторизовані користувачі з відповідними правами можуть вносити зміни, що забезпечує конфіденційність і порядок у роботі.

Процес реалізації включав детальний аналіз потреб користувачів і технічних вимог, що стало основою для створення чіткої архітектури. Розподіл логіки між серверною та клієнтською частинами сприяв їхній незалежності та легкості подальшого вдосконалення. Налагодження системи включало усунення проблем із обробкою запитів на сервері та оптимізацію оновлення стану в інтерфейсі, що забезпечило стабільну роботу. Тестування, проведене в автоматизованому режимі, охопило всі ключові функції: авторизацію, створення, редагування, переміщення завдань, пошук і фільтрацію. Успішне проходження тестів підтвердило відповідність системи вимогам і її готовність до практичного застосування.

Перевагою системи є її адаптивність і гнучкість. Інтерфейс протестовано на різних пристроях, що забезпечує комфортну роботу як на великих екранах, так і на мобільних гаджетах. Використання сучасних бібліотек для стилізації та обробки запитів дозволило створити привабливий і функціональний дизайн, який відповідає актуальним стандартам. Серверна частина, налаштована для роботи з MongoDB, може бути легко адаптована для хмарного розгортання, що робить систему масштабованим рішенням для команд різного розміру.

Тестування продемонструвало високу надійність системи. Автоматизовані тести, проведені в середовищі розробки, показало успішність для всіх сценаріїв, що свідчить про її стабільність і готовність до використання. Результати тестування, представлені на скриншотах, підтверджують коректність реалізації всіх функцій. Це доводить, що система не лише виконує покладені на неї завдання, але й створює передумови для підвищення ефективності командної роботи в реальних умовах.

Праця над дипломною дала змогу поглибити знання в галузі веброзробки, зокрема в роботі з React, Node.js, MongoDB та сучасними інструментами

тестування. Було освоєно принципи побудови REST API, оптимізації клієнт-серверної взаємодії та забезпечення безпеки даних. Досвід вирішення складних завдань, таких як налагодження асинхронних операцій і інтеграція компонентів, став значним внеском у професійний розвиток. Отримані навички можуть бути застосовані для подальших проєктів у сфері інформаційних технологій.

На основі проведеної роботи можна ствердити, що всі цілі, визначені на початку дипломної, були успішно досягнуті. Вебсистема для організації та контролю виконання проєктних завдань є повноцінним інструментом, який відповідає сучасним вимогам до продуктивності, безпеки та зручності. Вона готова до практичного використання в реальних проєктах, де потрібна ефективна координація завдань і співпраця між учасниками команди. У майбутньому систему можна розширити, додавши підтримку кількох мов чи інтеграцію з іншими інструментами, що зробить її ще більш універсальною.

Таким чином, розроблена вебсистема успішно виконує своє призначення, підтверджуючи виконання всіх поставлених завдань завдяки інтуїтивному інтерфейсу, надійній архітектурі та всебічному тестуванню. Ця робота є важливим кроком у розвитку навичок програмування та підтверджує готовність автора до подальших досліджень і професійної діяльності в сфері інформаційних технологій.

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ


1. Форкун, Ю., Мартинюк, В., Яшина, О. Метод розробки та проектування архітектурної складової програмного застосунку // MEASURING AND COMPUTING DEVICES IN TECHNOLOGICAL PROCESSES. URL: https://vottp.khmnu.edu.ua/index.php/vottp/article/view/196/178 — (дата звернення 03.01.25)

2. Форкун, Ю., Форкун, І., Яшина, О., Праворська, Н. Архітектурні методи оптимізації швидкодії та відмовостійкості програмних застосунків // MEASURING AND COMPUTING DEVICES IN TECHNOLOGICAL PROCESSES. URL: https://vottp.khmnu.edu.ua/index.php/vottp/article/view/183/163 (дата звернення 03.01.2025)

4. Lopatto I., Lebiga M., Forkun Y., Boyarchuk A. Method for Determining the Informativeness of the Software Requirements Specifications. — URL: https://ceur-ws.org/Vol-2853/paper15.pdf (дата звернення 03.06.2025).

5. Коваль А.І., Яшина О.М., Радельчук Г.І., Форкун Ю.В. Порівняння об’єктно-орієнтованої та функційної парадигм програмування у проектуванні програмного забезпечення — URL: https://elar.khmnu.edu.ua/server/api/core/bitstreams/0ba48ff1-05d4-4090-a90f-c7b85f1af4d4/content (дата звернення 03.06.2025)

6. Setting Up ReactJs with Vite: A Tutorial. URL: https://www.geeksforgeeks.org/how-to-setup-reactjs-with-vite/ (дата звернення 08.01.25)

7. Why Use React with Vite for Development. URL: https://dev.to/doccaio/react-vite-why-use-cg2 (дата звернення 11.01.25)

8. Understanding Kanban Boards for Task Management. URL: https://www.atlassian.com/agile/kanban/boards (дата звернення 16.01.25)

9. Kanban Task Management. URL: https://kanbanboard.co.uk/kanban-task-management-system (дата звернення 16.01.25)

10. Comprehensive Guide to Kanban Boards Usage. URL: https://businessmap.io/kanban-resources/getting-started/what-is-kanban-board (дата звернення 16.01.25)

11. Beginner's Guide to Kanban for Agile Teams. URL: https://asana.com/resources/what-is-kanban (дата звернення 19.01.25)

12. Kanban Methodology in Modern Work Management. URL: https://vertocloud.co.uk/blog/what-is-kanban-methodology/ (дата звернення 21.01.25)

13. Kanban Boards for Project Management Explained. URL: https://www.projectmanager.com/guides/kanban (дата звернення 22.01.25)

14. What is a use case diagram? URL: https://miro.com/diagramming/what-is-a-use-case-diagram/ (дата звернення 22.01.25)

15. User Authentication with JWT in Node.js MongoDB. URL: https://www.loginradius.com/blog/engineering/nodejs-and-mongodb-application-authentication-by-jwt (дата звернення 02.02.25)

16. Model-View-Controller (MVC) Architecture: Benefits and Challenges. URL: https://www.taazaa.com/mvc-architecture-benefits-and-challenges/ (дата звернення 03.02.25)

17. Microservices vs. Monolithic Architecture. URL: https://dev.to/webdevlapani/building-accessible-react-components-with-react-aria-55l8 (дата звернення 04.02.25)

18. Custom JWT Authentication in MongoDB Atlas. URL: https://www.mongodb.com/docs/atlas/app-services/authentication/custom-jwt/ (дата звернення 04.02.25)

19. JWT Authentication with MongoDB in .NET. URL: https://medium.com/nerd-for-tech/net-jwt-authentication-with-mongodb-9bca4a33d3f0 (дата звернення 04.02.25)

20. 10 Types of Software Architecture Patterns. URL: https://distantjob.com/blog/software-architecture-patterns/ (дата звернення 06.02.25)

21. JWT Authentication in Spring Boot with MongoDB. URL: https://www.hungrycoders.com/blog/jwt-authentication-in-spring-boot-with-mongodb (дата звернення 08.02.25)

22. Spring Boot MongoDB JWT with Spring Security. URL: https://www.bezkoder.com/spring-boot-jwt-auth-mongodb/ (дата звернення 14.02.25)

23. Node.js MongoDB JWT Authentication with Refresh Tokens. URL: https://jasonwatmore.com/post/2020/06/17/nodejs-mongodb-api-jwt-authentication-with-refresh-tokens (дата звернення 16.02.25)

24. Implementing Drag and Drop in React Components. URL: https://www.geeksforgeeks.org/implement-drag-and-drop-using-react-component/ (дата звернення 26.02.25)

25. Using HTML Drag-and-Drop API in React. URL: https://www.smashingmagazine.com/2020/02/html-drag-drop-api-react/ (дата звернення 27.02.25)

26. React-Draggable Component for Drag-and-Drop Features. URL: https://www.npmjs.com/package/react-draggable (дата звернення 04.03.25)

27. MongoDB Schema Design Best Practices. URL: https://www.mongodb.com/developer/products/mongodb/mongodb-schema-design-best-practices (дата звернення 04.03.25)

28. React Components for Drag and Drop - Pluralsight. URL: https://www.pluralsight.com/resources/blog/guides/drag-and-drop-react-components (дата звернення 05.03.25)

29. What Is MongoDB? An Expert Guide. URL: https://www.oracle.com/ua/database/mongodb/ (дата звернення 05.03.25)

30. Best Web Development Stacks to Use in 2025. URL: https://www.nobledesktop.com/classes-near-me/blog/best-web-development-stacks (дата звернення 06.03.25)

31. What is MERN Stack? A Complete Guide [2025] - GUVI Blogs. URL: https://www.guvi.in/blog/guide-for-mern-stack/ (дата звернення 06.03.25)

32. MERN stack tutorial: The Complete Guide with Examples. URL: https://deadsimplechat.com/blog/mern-stack-the-complete-guide/ (дата звернення 07.03.25)

33. How to Add Drag and Drop in React with React Beautiful DnD. URL: https://www.freecodecamp.org/news/how-to-add-drag-and-drop-in-react-with-react-beautiful-dnd/ (дата звернення 08.03.25)

34. Advantages and Benefits of React JS: The Essential Tool for Modern Developers. URL: https://maybe.works/blogs/react-js-advantages (дата звернення 12.03.25)

35. What reasons to use React for web development?. URL: https://www.netguru.com/blog/why-use-react (дата звернення 13.03.25)

36. JWT vs. OAuth: Comparing Authentication Protocols for Web Apps. URL: https://frontegg.com/blog/oauth-vs-jwt (дата звернення 13.03.25)

37. Security Best Practices for MERN Stack Applications. URL: https://auth0.com/blog/security-best-practices-for-mern-stack-applications/ (дата звернення 14.03.25)

38. Building Accessible UI Components in React with ARIA. URL: https://dev.to/webdevlapani/building-accessible-react-components-with-react-aria-55l8 (дата звернення 18.03.25)

39. Jest testing: A complete tutorial. URL: https://www.testim.io/blog/jest-testing-a-helpful-introductory-tutorial/ (дата звернення 29.03.25)

40. Visual Testing with Jest. URL: https://www.geeksforgeeks.org/testing-with-jest/ (дата звернення 03.04.25)

ДОДАТОК А

(обов’язковий)

# ТЕХНІЧНЕ ЗАВДАННЯ

**Введення**

Ця робота виконується в рамках дипломного проєкту з розробки вебсистеми, призначеної для організації та контролю виконання проєктних завдань. У сучасному цифровому світі, де швидкість і ефективність є ключовими факторами успіху, автоматизовані рішення для управління проєктами стають незамінними. Такі системи допомагають оптимізувати робочі процеси, забезпечувати прозорість виконання завдань і сприяти ефективній командній роботі, особливо в умовах віддаленої співпраці та розподілених команд.

Умовне позначення системи: «Вебсистема для управління проєктними завданнями».

## 1 Підстава для розробки

Розробка здійснюється на основі документа «Завдання на дипломний проєкт», затвердженого завідувачем кафедри інформаційних технологій.

Назва розробки: Вебсистема для координації та моніторингу виконання проєктних завдань.

## 2 Призначення розробки

Вебсистема для координації та моніторингу виконання проєктних завдань призначена для забезпечення зручного та ефективного інструменту, який дозволяє командам планувати, контролювати та оптимізувати робочі процеси в реальному часі. Система спрямована на підтримку різних груп користувачів — менеджерів, виконавців і адміністраторів — у створенні структурованих робочих просторів, де можна визначати завдання, відстежувати їхній прогрес і координувати діяльність незалежно від географічного розташування.

Основний функціонал вебсистеми включає:

– створення, редагування та видалення проєктних завдань із зазначенням пріоритетів, дедлайнів і виконавців;

– організацію завдань у візуальні дошки з можливістю перетягування між статусами;

– сповіщення змін і створення завдань, нагадування про дедлайн;

– безпечне управління доступом із розподілом ролей і прав для різних користувачів.

Вебсистема розробляється для компаній і команд, які прагнуть підвищити ефективність управління проєктами, автоматизувати рутинні операції та забезпечити прозорість робочих процесів. Вона сумісна з усіма сучасними веббраузерами та адаптована для використання на різних пристроях із доступом до Інтернету, включаючи комп'ютери, планшети та смартфони.

## 3 Вимоги до програмного забезпечення

### 3.1 Функціональні вимоги

Функціональні вимоги визначають основні можливості вебсистеми для організації та контролю виконання проєктних завдань, забезпечуючи зручність і ефективність для всіх груп користувачів — менеджерів, виконавців і адміністраторів.

Реєстрація користувача: користувач може створити обліковий запис, вказавши ім'я, електронну пошту та пароль. Пароль має бути не коротшим за 10 символів і містити великі та малі літери, цифри й спеціальні символи. Після реєстрації на email надсилається лист із посиланням для підтвердження облікового запису.

Автентифікація користувача: система забезпечує вхід за допомогою електронної пошти та пароля. У разі введення некоректних даних відображається повідомлення про помилку, а після 5 невдалих спроб вхід блокується на 10 хвилин.

Створення проєктного завдання: користувач може додати нове завдання, вказавши назву, опис, дедлайн, пріоритет (низький, середній, високий), виконавця та прикріпити файли (наприклад, зображення чи документи). Дані зберігаються в базі даних із автоматичним оновленням статусу.

Редагування та видалення завдань: користувач має можливість змінювати будь-які параметри завдання (назву, опис, дедлайн, пріоритет, виконавця) або видаляти завдання. Зміни відображаються в реальному часі для всіх учасників проєкту.

Візуалізація завдань: завдання відображаються у форматі дошки Kanban із колонками, що представляють статуси (наприклад, «Заплановано», «У роботі», «Завершено»). Користувач може переміщувати завдання між колонками методом drag-and-drop.

Фільтрація та пошук завдань: система дозволяє шукати завдання за ключовими словами (назва, опис, виконавець) і фільтрувати їх за статусом, пріоритетом або дедлайном із можливістю сортування за датою створення чи дедлайном.

Сповіщення про події: користувачі отримують сповіщення про наближення дедлайнів, призначення на завдання чи зміну статусу через інтерфейс системи або на електронну пошту.

### 3.2 Нефункціональні вимоги

Нефункціональні вимоги визначають якісні характеристики вебсистеми для організації та контролю виконання проєктних завдань, які впливають на її продуктивність, безпеку, зручність і надійність.

Швидкодія: час завантаження основної сторінки не має перевищувати 2 секунди за стандартного інтернет-з’єднання, а обробка запитів (наприклад, створення завдання) — не більше 1,5 секунди для забезпечення плавної роботи.

Перехід між розділами: перемикання між різними сторінками чи модулями системи (наприклад, від дошки завдань до звітів) має відбуватися протягом 1 секунди для підтримки безперервного робочого процесу.

Безпека: система використовує протокол HTTPS для шифрування даних під час передачі між клієнтом і сервером. Паролі захищаються алгоритмом хешування (наприклад, bcrypt), а автентифікація реалізується через JSON Web Token (JWT) із захистом від підроблення токенів.

Інтерфейс і зручність: дизайн системи розроблено з урахуванням принципів UX/UI, із чіткою навігацією, легко читаемим шрифтом (наприклад,

Roboto або Open Sans) і гармонійною кольоровою палітрою, що мінімізує втому очей і забезпечує комфортне використання.

Доступність: вебсистема працює цілодобово (24/7), із плановими перервами для технічних оновлень чи усунення збоїв, які не перевищують 1% загального часу роботи на місяць.

### 3.3 Умови експлуатації

Вебсистема може використовуватись у сучасних браузерах на різних пристроях, таких як комп’ютери, планшети чи смартфони.

### 3.4 Вимоги до технічного забезпечення та продуктивності

Вебсистема для організації та контролю виконання проєктних завдань розроблена для роботи через браузери на пристроях із доступом до Інтернету, не потребуючи локальної установки на мобільних пристроях чи комп’ютерах. Доступ до системи здійснюється виключно через вебінтерфейс, що забезпечує гнучкість використання на різних платформах.

Для стабільної та ефективної роботи вебсистеми рекомендуються такі технічні характеристики:

Операційні системи: сумісність із Windows, Linux, macOS, а також Android і iOS через підтримку сучасних веббраузерів (Chrome, Firefox, Safari, Edge).

Процесор: двоядерний процесор із частотою від 1,2 ГГц або вищою для оптимальної продуктивності.

Оперативна пам’ять: мінімально 2 ГБ, рекомендується 4 ГБ і більше для комфортної роботи з великою кількістю завдань.

Вільне місце: щонайменше 1 ГБ для кешування даних браузера та тимчасового зберігання файлів, завантажених у систему.

Інтернет-з’єднання: стабільна швидкість не нижче 10 Мбіт/с для швидкого завантаження сторінок і обробки запитів у реальному часі.

### 3.5 Вимоги до програмної сумісності та інтеграцій

Вебсистема для організації та контролю виконання проєктних завдань буде розроблена з використанням сучасного технологічного стеку, що забезпечить її

продуктивність, масштабованість і зручність у розробці. Для створення фронтенду буде використано бібліотеку React, яка дозволить реалізувати динамічний і адаптивний користувацький інтерфейс. Бекенд буде розроблено на основі Node.js із застосуванням фреймворку Express.js для створення ефективного RESTful API та обробки запитів. Для зберігання даних, таких як завдання, профілі користувачів і сповіщення, буде використано базу даних MongoDB, яка забезпечує гнучкість і масштабованість.

Для забезпечення додаткової функціональності та оптимізації роботи системи буде здійснено інтеграцію з такими технологіями та сервісами:

JSON Web Token (JWT): буде застосовано для реалізації безпечної автентифікації та авторизації користувачів, що гарантуватиме захист токенів і управління сесіями.

Цей технологічний стек і набір інтеграцій буде використано для створення надійної, безпечної та масштабованої вебсистеми, яка відповідатиме потребам користувачів у сфері управління проєктними завданнями.

### 3.6 Вимоги до інтерфейсу та дизайну

Вебсистема для організації та контролю виконання проєктних завдань матиме наступні вимоги до інтерфейсу та дизайну, які забезпечать зручність і привабливий вигляд для користувачів:

Кольорова гама: буде використано комбінацію сірих, блакитних і зелених відтінків для створення сучасного та спокійного візуального стилю, із акцентними кольорами (наприклад, зелений для підтверджень) для виділення ключових елементів.

Навігація: панель навігації буде розміщена у верхній частині екрана (хедер), де передбачатимуться кнопки для переходу до дошки завдань, аналітики, профілю користувача та пошуку завдань, із логічним розташуванням для швидкого доступу.

Адаптивність: інтерфейс буде розроблено з урахуванням респонсивного дизайну, що забезпечить коректне відображення на пристроях різної роздільної здатності, включаючи комп’ютери, планшети та смартфони.

Ці вимоги до дизайну сприятимуть створенню інтуїтивно зрозумілого та естетично привабливого інтерфейсу, що відповідатиме принципам UX/UI та потребам користувачів.

## 4 Вимоги до програмної документації

При передачі вебсистеми для організації та контролю виконання проєктних завдань замовнику буде надано наступний комплект документації та матеріалів:

– програмний код;

– опис функціональних можливостей;

– інструкція користування;

– технічне завдання.

## 5 Стадії та етапи розробки

| Стадія розробки | Етапи робіт | Зміст робіт |
|---|---|---|
| Технічне завдання | 01.01.25 – 20.02.25 | Формулювання вимог до системи та документація, призначення, опис цілей та завдань |
| Ескізний проєкт | 21.02.25 – 20.03.25 | Розробка дизайну інтерфейсу, обрання середовища програмування |
| Технічний проєкт | 21.03.25 – 30.04.25 | Опис технічних характеристик та архітектури |
| Робочий проєкт | 01.05.25 – 25.05.25 | Реалізація логіки та основних функціональних можливостей вебзастосунку. Інтеграція з базою даних та взаємодія із зовнішніми сервісами |
| Розробка програмної документації | Травень 2025 | Оформлення документації, передбаченої технічним завдання |
| Тестування системи | 26.05.25 – 30.05.25 | Проведення різних тестів для перевірки правильності роботи функцій вебзастосунку |

| | | |
|---|---|---|
| Впровадження | З 01.06.2025 | Підготовка до розгортання системи, налаштування домену та передача вебзастосунку на експлуатацію користувачам |

## 6 Порядок перевірки та приймання

Процес перевірки та приймання вебзастосунку здійснюється відповідно до визначених раніше вимог та містить такі етапи:

– функціональне тестування: перевірка основного функціоналу розробленого вебзастосунку;

– тестування продуктивності: оцінка часу завантаження сторінок, відповіді від сервера тощо;

– дотримання норм безпеки: перевірка захисту персональних даних, шифрування паролів, захист від атак;

– кросбраузерне тестування: перевірка правильності роботи в різних браузерах та пристроях;

– тестування інтеграції: перевірка взаємодії програмного забезпечення із сторонніми сервісами та API;

– тестування інтерфейсу: оцінка того, наскільки зручний та інтуїтивний інтерфейс різних сторінок та елементів вебзастосунку;

– відповідність документації: перевірка на відповідність встановленим вимогам та дотримання всіх інструкцій і також вирішення різноманітних проблем.

ДОДАТОК Б

(обов’язковий)

# КОД ПРОГРАМИ

## Task.js

```
import mongoose from 'mongoose';

const TaskSchema = new mongoose.Schema({
  userId: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },
// Відповідальний користувач
  creatorId: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true
}, // Початковий власник завдання
  title: { type: String, required: true },
  description: { type: String, default: '' },
  priority: { type: String, enum: ['low', 'medium', 'high'], default: 'low' },
  deadline: { type: Date, required: true },
  assignee: { type: String, default: '' }, // Email відповідального користувача
  tags: [{ title: String, bg: String, text: String }],
  status: { type: String, enum: ['backlog', 'pending', 'todo', 'doing', 'done'],
default: 'backlog' },
  order: { type: Number, default: 0 },
  completed: { type: Boolean, default: false },
  createdAt: { type: Date, default: Date.now },
}, { timestamps: true });

const Task = mongoose.model('Task', TaskSchema);

export default Task;
```

## User.js

```
import mongoose from 'mongoose';

const UserSchema = new mongoose.Schema({
    username: { type: String, required: true },
    email: { type: String, required: true, unique: true },
    password: { type: String, required: true }
}, { timestamps: true });

const User = mongoose.model('User', UserSchema);

export default User;
```

## tasks.js

```
import express from 'express';
import Task from '../models/Task.js';
```

```
import User from '../models/User.js'; // Додаємо модель User для пошуку за email
import jwt from 'jsonwebtoken';
import dotenv from 'dotenv';

dotenv.config();

const router = express.Router();
const JWT_SECRET = process.env.JWT_SECRET;

const authMiddleware = (req, res, next) => {
    const token = req.header('Authorization');
    if (!token) return res.status(401).json({ message: 'Немає доступу' });

    try {
        const tokenString = token.replace('Bearer ', '');
        const decoded = jwt.verify(tokenString, JWT_SECRET);
        req.user = { id: decoded.userId };
        if (!req.user.id) {
            return res.status(401).json({ message: 'Невірний токен: userId
відсутній' });
        }
        next();
    } catch (error) {
        console.error('Помилка верифікації токену:', error);
        res.status(401).json({ message: 'Невірний токен' });
    }
};

// Новий маршрут для перевірки email користувача
router.post('/check-email', authMiddleware, async (req, res) => {
    try {
        const { email } = req.body;
        if (!email) {
            return res.status(400).json({ message: 'Email є обов’язковим' });
        }

        const user = await User.findOne({ email });
        if (!user) {
            return res.status(404).json({ message: 'Користувач із таким email не
знайдений' });
        }

        res.json({ userId: user._id, email: user.email });
    } catch (error) {
        console.error('Помилка перевірки email:', error);
        res.status(500).json({ message: 'Помилка сервера' });
    }
});

// Отримати таски користувача
router.get('/', authMiddleware, async (req, res) => {
    try {
```

```
        // Отримуємо завдання, де користувач є або відповідальним (userId), або
творцем (creatorId)
        const tasks = await Task.find({
            $or: [
                { userId: req.user.id },
                { creatorId: req.user.id }
            ]
        });
        res.json(tasks);
    } catch (error) {
        console.error('Помилка отримання задач:', error);
        res.status(500).json({ message: 'Помилка сервера' });
    }
});

// Створити нову таску
router.post('/', authMiddleware, async (req, res) => {
    try {
        const { title, deadline, assignee, status, priority, description, tags,
order, completed } = req.body;
        if (!title || !deadline) {
            return res.status(400).json({ message: 'Назва та дедлайн є
обов’язковими' });
        }

        let assigneeUserId = req.user.id; // За замовчуванням відповідальний —
це сам користувач
        let assigneeEmail = '';

        // Якщо передано email відповідального, перевіряємо, чи існує такий
користувач
        if (assignee) {
            const user = await User.findOne({ email: assignee });
            if (!user) {
                return res.status(404).json({ message: 'Користувач із таким
email не знайдений' });
            }
            assigneeUserId = user._id; // Змінюємо userId на ідентифікатор
відповідального
            assigneeEmail = assignee;
        }

        const newTask = new Task({
            userId: assigneeUserId, // Відповідальний користувач
            creatorId: req.user.id, // Творець завдання
            title,
            deadline,
            assignee: assigneeEmail,
            status: status || 'backlog',
            priority: priority || 'low',
            description: description || '',
            tags: tags || [],
            order: order ?? 0,
```

```
            completed: completed || false,
        });

        await newTask.save();
        res.status(201).json(newTask);
    } catch (error) {
        console.error('Помилка створення задачі:', error);
        res.status(500).json({ message: 'Помилка сервера' });
    }
});

// Оновити таску
router.put('/:id', authMiddleware, async (req, res) => {
    try {
        const { status, title, deadline, assignee, completed, priority,
description, tags, order } = req.body;
        const task = await Task.findById(req.params.id);

        if (!task) {
            return res.status(404).json({ message: 'Задача не знайдена' });
        }

        if (!task.userId || !task.creatorId) {
            return res.status(400).json({ message: 'Задача не має userId або
creatorId. Оновіть базу даних.' });
        }

        // Перевіряємо, чи користувач має доступ до задачі (є творцем або
відповідальним)
        if (task.userId.toString() !== req.user.id && task.creatorId.toString()
!== req.user.id) {
            return res.status(403).json({ message: 'Немає доступу до цієї
задачі' });
        }

        // Якщо передано email відповідального, перевіряємо, чи існує такий
користувач
        if (assignee !== undefined) {
            if (assignee) {
                const user = await User.findOne({ email: assignee });
                if (!user) {
                    return res.status(404).json({ message: 'Користувач із таким
email не знайдений' });
                }
                task.userId = user._id; // Змінюємо userId на ідентифікатор
відповідального
                task.assignee = assignee;
            } else {
                // Якщо assignee пустий, повертаємо userId до творця
                task.userId = task.creatorId;
                task.assignee = '';
            }
        }
```

```
        // Оновлюємо інші поля задачі
        if (status) task.status = status;
        if (title) task.title = title;
        if (deadline) task.deadline = deadline;
        if (typeof completed === 'boolean') task.completed = completed;
        if (priority) task.priority = priority;
        if (description !== undefined) task.description = description;
        if (tags) task.tags = tags;
        if (order !== undefined) task.order = order;

        await task.save();
        res.json(task);
    } catch (error) {
        console.error('Помилка оновлення задачі:', error);
        res.status(500).json({ message: 'Помилка сервера' });
    }
});

// Видалити таску
router.delete('/:id', authMiddleware, async (req, res) => {
    try {
        const task = await Task.findById(req.params.id);
        if (!task) {
            return res.status(404).json({ message: 'Задача не знайдена' });
        }

        if (!task.userId || !task.creatorId) {
            return res.status(400).json({ message: 'Задача не має userId або
creatorId. Оновіть базу даних.' });
        }

        if (task.userId.toString() !== req.user.id && task.creatorId.toString()
!== req.user.id) {
            return res.status(403).json({ message: 'Немає доступу до цієї
задачі' });
        }

        await Task.findByIdAndDelete(req.params.id);
        res.json({ message: 'Задача видалена' });
    } catch (error) {
        console.error('Помилка видалення задачі:', error);
        res.status(500).json({ message: 'Помилка сервера' });
    }
});

export default router;
```

## users.js

```
import express from "express";
import bcrypt from "bcryptjs";
import jwt from "jsonwebtoken";
```

```
import User from "../models/User.js";
import dotenv from "dotenv";

dotenv.config();

const router = express.Router();
const JWT_SECRET = process.env.JWT_SECRET;

// Реєстрація
router.post("/register", async (req, res) => {
  const { username, email, password } = req.body;
  try {
    const hashedPassword = await bcrypt.hash(password, 10);
    const newUser = new User({ username, email, password: hashedPassword });
    await newUser.save();
    res.status(201).json({ message: "User registered successfully" });
  } catch (error) {
    res.status(500).json({ message: "Error registering user", error });
  }
});

// Логін
router.post("/login", async (req, res) => {
  try {
    const { email, password } = req.body;
    console.log("Отриманий запит:", req.body);

    const user = await User.findOne({ email });
    if (!user)
      return res.status(400).json({ message: "Невірний email або пароль" });

    const isMatch = await bcrypt.compare(password, user.password);
    if (!isMatch)
      return res.status(400).json({ message: "Невірний email або пароль" });

    console.log("JWT_SECRET:", JWT_SECRET);
    const token = jwt.sign({ userId: user._id }, JWT_SECRET, {
      expiresIn: "1h",
    });

    res.json({ token, userId: user._id });
  } catch (error) {
    console.error("Помилка логіну:", error);
    res.status(500).json({ message: "Помилка сервера", error: error.message });
  }
});

export default router;
```

## db.js

```
import mongoose from "mongoose";
import dotenv from "dotenv";
```

```
dotenv.config();

const connectDB = async () => {
  try {
    const conn = await mongoose.connect(process.env.MONGO_URI, {
      useNewUrlParser: true,
      useUnifiedTopology: true,
    });
    console.log(`MongoDB Connected: ${conn.connection.host}`);
  } catch (error) {
    console.error(error);
    process.exit(1);
  }
};

export default connectDB;
```

server.js

```
import dotenv from "dotenv";
import express from "express";
import mongoose from "mongoose";
import cors from "cors";
import userRoutes from "./routes/users.js";
import taskRoutes from "./routes/tasks.js";
import connectDB from "./db.js";

dotenv.config();

const app = express();

app.use(
  cors({
    origin: "http://localhost:5173", // Фронтенд на Vite
    credentials: true,
  })
);

app.use(express.json());

// Логування змінних середовища
console.log("Mongo URI:", process.env.MONGO_URI);
console.log("JWT Secret:", process.env.JWT_SECRET);

// Маршрути
app.use("/api/users", userRoutes);
app.use("/api/tasks", taskRoutes);

const port = 3000;
app.listen(port, () => console.log(`Сервер запущено на порту ${port}`));

connectDB();
```

## Login.tsx

```
import React, { useState } from "react";
import { useNavigate } from "react-router-dom";
import { authApi } from "../../services/api";
import { useAuth } from "../../context/AuthContext";

const Login = () => {
  const navigate = useNavigate();
  const { login } = useAuth();
  const [formData, setFormData] = useState({
    email: "",
    password: "",
  });

  const handleSubmit = async (e: React.FormEvent) => {
    e.preventDefault();
    try {
      const response = await authApi.login({
        email: formData.email,
        password: formData.password,
      });
      localStorage.setItem("token", response.data.token);
      login(formData.email);
      navigate("/");
    } catch (error: any) {
      console.error("Login error:", error.response?.data || error.message);
    }
  };

  return (
    <div className="min-h-screen flex items-center justify-center bg-gray-100">
      <div className="bg-white p-8 rounded-lg shadow-md w-96">
        <h2 className="text-2xl font-semibold mb-6 text-center">Login</h2>
        <form onSubmit={handleSubmit} className="space-y-4">
          <div>
            <input
              type="email"
              placeholder="Email"
              className="w-full px-4 py-2 border rounded-lg focus:outline-none
focus:ring-2 focus:ring-orange-400"
              value={formData.email}
              onChange={(e) =>
                setFormData({ ...formData, email: e.target.value })
              }
            />
          </div>
          <div>
            <input
              type="password"
              placeholder="Password"
              className="w-full px-4 py-2 border rounded-lg focus:outline-none
focus:ring-2 focus:ring-orange-400"
              value={formData.password}
```

```
              onChange={(e) =>
                setFormData({ ...formData, password: e.target.value })
              }
            />
          </div>
          <button
            type="submit"
            className="w-full bg-orange-400 text-white py-2 rounded-lg hover:bg-
orange-500 transition-colors"
          >
            Login
          </button>
        </form>

        <div className="text-center mt-4">
          <p className="text-gray-600">Don't have an account?</p>
          <button
            onClick={() => navigate("/register")}
            className="text-orange-500 hover:underline"
          >
            Register
          </button>
        </div>
      </div>
    </div>
  );
};

export default Login;
```

## Register.tsx

```
import React, { useState } from "react";
import { useNavigate } from "react-router-dom";
import { authApi } from "../../services/api";

const Register = () => {
  const navigate = useNavigate();
  const [formData, setFormData] = useState({
    username: "",
    email: "",
    password: "",
  });

  const handleSubmit = async (e: React.FormEvent) => {
    e.preventDefault();
    try {
      await authApi.register({ username: formData.username, email:
formData.email, password: formData.password });
      navigate("/login"); // Перенаправлення на логін після реєстрації
    } catch (error) {
      console.error("Register error:", error);
    }
```

```
  };

  return (
    <div className="min-h-screen flex items-center justify-center bg-gray-100">
      <div className="bg-white p-8 rounded-lg shadow-md w-96">
        <h2 className="text-2xl font-semibold mb-6 text-center">Register</h2>
        <form onSubmit={handleSubmit} className="space-y-4">
          <div>
            <input
              type="text"
              placeholder="Username"
              className="w-full px-4 py-2 border rounded-lg focus:outline-none
focus:ring-2 focus:ring-orange-400"
              value={formData.username}
              onChange={(e) => setFormData({ ...formData, username:
e.target.value })}
            />
          </div>
          <div>
            <input
              type="email"
              placeholder="Email"
              className="w-full px-4 py-2 border rounded-lg focus:outline-none
focus:ring-2 focus:ring-orange-400"
              value={formData.email}
              onChange={(e) => setFormData({ ...formData, email: e.target.value
})}
            />
          </div>
          <div>
            <input
              type="password"
              placeholder="Password"
              className="w-full px-4 py-2 border rounded-lg focus:outline-none
focus:ring-2 focus:ring-orange-400"
              value={formData.password}
              onChange={(e) => setFormData({ ...formData, password:
e.target.value })}
            />
          </div>
          <button
            type="submit"
            className="w-full bg-orange-400 text-white py-2 rounded-lg hover:bg-
orange-500 transition-colors"
          >
            Register
          </button>
          <p className="text-center mt-4">
            Already have an account?{" "}
            <a href="/login" className="text-orange-500 hover:underline">
              Login
            </a>
          </p>
```

```
        </form>
      </div>
    </div>
  );
};

export default Register;
```

## index.tsx

```
/* eslint-disable @typescript-eslint/no-explicit-any */
import { DragDropContext, Droppable, Draggable } from "@hello-pangea/dnd";
import { useState, useEffect } from "react";
import { tasksApi } from "../../services/api";
import { Board } from "../../data/board";
import { Columns, TaskT } from "../../types";
import { IoAddOutline } from "react-icons/io5";
import { FaGripVertical } from "react-icons/fa";
import AddModal from "../../components/Modals/AddModal";
import { useSearch } from "../../context/SearchContext";

interface Notification {
  id: string;
  message: string;
  timestamp: number;
}

const addNotification = (message: string) => {
  const notifications: Notification[] = JSON.parse(
    localStorage.getItem("notifications") || "[]"
  );
  const newNotification: Notification = {
    id: Date.now().toString(),
    message,
    timestamp: Date.now(),
  };
  notifications.push(newNotification);
  localStorage.setItem("notifications", JSON.stringify(notifications));
};

const Home = () => {
  const { searchText, selectedTags, setSelectedTags } = useSearch();
  const [columns, setColumns] = useState<Columns>(
    Object.keys(Board).reduce((acc, key) => {
      acc[key] = { ...Board[key], items: [] };
      return acc;
    }, {} as Columns)
  );
  const [modalOpen, setModalOpen] = useState(false);
  const [taskModalOpen, setTaskModalOpen] = useState(false);
  const [selectedColumn, setSelectedColumn] = useState("");
  const [selectedTask, setSelectedTask] = useState<TaskT | null>(null);
  const [isEditing, setIsEditing] = useState(false);
```

```
const [editedTask, setEditedTask] = useState<TaskT | null>(null);

useEffect(() => {
  const fetchTasks = async () => {
    try {
      const response = await tasksApi.getTasks();
      console.log("Дані з сервера:", response.data);
      const tasks: TaskT[] = response.data;

      const newColumns: Columns = {
        backlog: { name: "Backlog", items: [] },
        pending: { name: "Pending", items: [] },
        todo: { name: "To Do", items: [] },
        doing: { name: "Doing", items: [] },
        done: { name: "Done", items: [] },
      };

      tasks.forEach((task, index) => {
        const column = task.status || "backlog";
        newColumns[column].items.push({
          ...task,
          id: task._id || `task-${index}`,
          title: task.title || "Untitled Task",
          description: task.description || "",
          deadline: new Date(task.deadline).getTime(),
          createdAt: new Date(task.createdAt).getTime(),
          tags: task.tags || [],
          priority: task.priority || "low",
          status: task.status || "backlog",
          order: task.order ?? index,
          assignee: task.assignee || "",
          completed: task.completed || false,
        });
      });

      Object.keys(newColumns).forEach((col) => {
        newColumns[col].items.sort((a, b) => (a.order || 0) - (b.order || 0));
      });

      setColumns(newColumns);
    } catch (error) {
      console.error("Помилка завантаження завдань:", error);
      addNotification("Не вдалося завантажити завдання. Спробуйте ще раз.");
    }
  };
  fetchTasks();
}, []);

useEffect(() => {
  const checkDeadlines = () => {
    const now = Date.now();
    Object.entries(columns).forEach(([, column]) => {
      column.items.forEach((task) => {
```

```
          if (task.deadline && !task.completed) {
            const timeLeft = task.deadline - now;
            if (
              timeLeft <= 24 * 60 * 60 * 1000 &&
              timeLeft > 23 * 60 * 60 * 1000
            ) {
              const message = `Завдання "${task.title}" у колонці
"${column.name}" закінчується через день!`;
              addNotification(message);
            } else if (
              timeLeft <= 60 * 60 * 1000 &&
              timeLeft > 59 * 60 * 1000
            ) {
              const message = `Завдання "${task.title}" у колонці
"${column.name}" закінчується через годину!`;
              addNotification(message);
            }
          }
        });
      });
    };

    const interval = setInterval(checkDeadlines, 60000);
    return () => clearInterval(interval);
  }, [columns]);

  const getAllTags = () => {
    const allTags = new Set<string>();
    Object.values(columns).forEach((column) => {
      column.items.forEach((task) => {
        task.tags.forEach((tag) => {
          allTags.add(tag.title);
        });
      });
    });
    return Array.from(allTags);
  };

  const filteredColumns = () => {
    const newColumns = { ...columns };
    Object.keys(newColumns).forEach((columnId) => {
      newColumns[columnId] = {
        ...newColumns[columnId],
        items: newColumns[columnId].items.filter((task) => {
          const matchesSearch =
            task.title.toLowerCase().includes(searchText.toLowerCase()) ||
            task.tags.some((tag) =>
              tag.title.toLowerCase().includes(searchText.toLowerCase())
            );
          const matchesTags =
            selectedTags.length === 0 ||
            task.tags.some((tag) => selectedTags.includes(tag.title));
          return matchesSearch && matchesTags;
```

```
      }),
    };
  });
  return newColumns;
};

const toggleTagFilter = (tag: string) => {
  setSelectedTags((prev: string[]) =>
    prev.includes(tag)
      ? prev.filter((t: string) => t !== tag)
      : [...prev, tag]
  );
};

const openModal = () => {
  setModalOpen(true);
};

const closeModal = () => {
  setModalOpen(false);
  setSelectedColumn("");
};

const openTaskModal = (task: TaskT, columnId: string) => {
  setSelectedTask(task);
  setSelectedColumn(columnId);
  setEditedTask({ ...task });
  setTaskModalOpen(true);
  setIsEditing(false);
};

const closeTaskModal = () => {
  setTaskModalOpen(false);
  setSelectedTask(null);
  setSelectedColumn("");
  setIsEditing(false);
};

const handleAddTask = async (taskData: TaskT) => {
  try {
    const response = await tasksApi.createTask({
      title: taskData.title,
      description: taskData.description,
      priority: taskData.priority || "low",
      deadline: new Date(taskData.deadline),
      assignee: taskData.assignee,
      tags: taskData.tags,
      status: selectedColumn,
      order: columns[selectedColumn].items.length,
      completed: taskData.completed || false,
    });

    const newTask: TaskT = {
```

```
        ...taskData,
        id: response.data._id,
        createdAt: new Date(response.data.createdAt).getTime(),
        priority: response.data.priority || taskData.priority || "low",
        status: response.data.status || selectedColumn,
        order: response.data.order ?? columns[selectedColumn].items.length,
        description: response.data.description || taskData.description,
        assignee: response.data.assignee || taskData.assignee,
        tags: response.data.tags || taskData.tags,
        completed: response.data.completed || false,
      };

      const newBoard = { ...columns };
      newBoard[selectedColumn].items.push(newTask);
      setColumns(newBoard);
      addNotification(
        `Завдання "${taskData.title}" створено в колонці
"${newBoard[selectedColumn].name}"`
      );
    } catch (error) {
      console.error("Помилка створення завдання:", error);
      addNotification("Не вдалося створити завдання. Спробуйте ще раз.");
    }
  };

  const handleDeleteTask = async () => {
    if (!selectedTask || !selectedColumn) return;
    try {
      await tasksApi.deleteTask(selectedTask.id);
      const newBoard = { ...columns };
      newBoard[selectedColumn].items = newBoard[selectedColumn].items.filter(
        (item) => item.id !== selectedTask.id
      );
      newBoard[selectedColumn].items.forEach((item, index) => {
        item.order = index;
        tasksApi.updateTask(item.id, { order: index }).catch((error) =>
          console.error(`Помилка оновлення порядку завдання ${item.id}:`, error)
        );
      });
      setColumns({ ...newBoard });
      addNotification(`Завдання "${selectedTask.title}" видалено`);
      closeTaskModal();
    } catch (error) {
      console.error("Помилка видалення завдання:", error);
      addNotification("Не вдалося видалити завдання. Спробуйте ще раз.");
    }
  };

  const handleCompleteTask = async () => {
    if (!selectedTask || !selectedColumn) return;
    try {
      await tasksApi.updateTask(selectedTask.id, { completed: true });
      const newBoard = { ...columns };
```

```
      const taskIndex = newBoard[selectedColumn].items.findIndex(
        (item) => item.id === selectedTask.id
      );
      if (taskIndex !== -1) {
        newBoard[selectedColumn].items[taskIndex] = {
          ...newBoard[selectedColumn].items[taskIndex],
          completed: true,
        };
        setColumns({ ...newBoard });
        addNotification(`Завдання "${selectedTask.title}" завершено`);
        closeTaskModal();
      }
    } catch (error) {
      console.error("Помилка завершення завдання:", error);
      addNotification("Не вдалося завершити завдання. Спробуйте ще раз.");
    }
  };

  const handleEditTask = async () => {
    if (!editedTask || !selectedColumn) return;
    try {
      const updatedTaskData = {
        title: editedTask.title,
        description: editedTask.description,
        priority: editedTask.priority || "low",
        deadline: new Date(editedTask.deadline),
        assignee: editedTask.assignee,
        tags: editedTask.tags || [],
        status: selectedColumn,
        order: editedTask.order ?? 0,
        completed: editedTask.completed || false,
      };

      const response = await tasksApi.updateTask(editedTask.id,
updatedTaskData);

      const newBoard = { ...columns };
      const taskIndex = newBoard[selectedColumn].items.findIndex(
        (item) => item.id === editedTask.id
      );
      if (taskIndex !== -1) {
        newBoard[selectedColumn].items[taskIndex] = {
          ...editedTask,
          ...response.data,
          id: response.data._id || editedTask.id,
          createdAt: new Date(response.data.createdAt).getTime(),
          deadline: new Date(response.data.deadline).getTime(),
          order: response.data.order ?? editedTask.order ?? 0,
        };
        setColumns(newBoard);
        setSelectedTask(newBoard[selectedColumn].items[taskIndex]);
        setIsEditing(false);
        addNotification(`Завдання "${editedTask.title}" оновлено`);
```

```
    }
  } catch (error) {
    console.error("Помилка оновлення завдання:", error);
    addNotification("Не вдалося оновити завдання. Спробуйте ще раз.");
  }
};

const getPriorityColor = (priority: string | undefined) => {
  const priorityLower = priority?.toLowerCase() || "low";
  switch (priorityLower) {
    case "high":
      return "bg-red-200 text-red-800 border-red-300";
    case "medium":
      return "bg-yellow-200 text-yellow-800 border-yellow-300";
    case "low":
      return "bg-green-200 text-green-800 border-green-300";
    default:
      return "bg-gray-200 text-gray-800 border-gray-300";
  }
};

const handleDragEnd = async (result: any) => {
  const { source, destination } = result;

  if (!destination) return;

  if (
    source.droppableId === destination.droppableId &&
    source.index === destination.index
  ) {
    return;
  }

  const newColumns = { ...columns };
  const sourceColumn = newColumns[source.droppableId];
  const destColumn = newColumns[destination.droppableId];
  const [movedTask] = sourceColumn.items.splice(source.index, 1);

  movedTask.status = destination.droppableId;
  movedTask.order = destination.index;
  destColumn.items.splice(destination.index, 0, movedTask);

  sourceColumn.items.forEach((item, index) => {
    item.order = index;
    tasksApi.updateTask(item.id, { order: index }).catch((error) =>
      console.error(`Помилка оновлення порядку завдання ${item.id}:`, error)
    );
  });

  destColumn.items.forEach((item, index) => {
    item.order = index;
    tasksApi.updateTask(item.id, { order: index }).catch((error) =>
      console.error(`Помилка оновлення порядку завдання ${item.id}:`, error)
```

```
      );
    });

    setColumns(newColumns);

    try {
      await tasksApi.updateTask(movedTask.id, {
        status: destination.droppableId,
        order: destination.index,
      });
      addNotification(
        `Завдання "${movedTask.title}" переміщено до "${destColumn.name}"`
      );
    } catch (error) {
      console.error("Помилка оновлення статусу завдання:", error);
      addNotification("Не вдалося перемістити завдання. Спробуйте ще раз.");
      const rollbackColumns = { ...columns };
      sourceColumn.items.splice(source.index, 0, movedTask);
      destColumn.items.splice(destination.index, 1);
      setColumns(rollbackColumns);
    }
  };

  const allTags = getAllTags();
  const displayedColumns = filteredColumns();

  // Явно задаємо порядок колонок
  const columnOrder = ["backlog", "pending", "todo", "doing", "done"];

  return (
    <div className="min-h-screen bg-[#f5e9e2] flex flex-col pt-[70px] pl-[60px]
md:pl-[230px]">
      {/* Фільтр за тегами */}
      <div className="px-5 py-2 flex flex-wrap gap-2">
        <h3 className="text-sm font-medium text-gray-700 mr-2">Filter by
tags:</h3>
        {allTags.length > 0 ? (
          allTags.map((tag) => (
            <button
              key={tag}
              onClick={() => toggleTagFilter(tag)}
              className={`px-3 py-1 rounded-full text-sm font-medium transition-
colors ${
                selectedTags.includes(tag)
                  ? "bg-orange-400 text-white"
                  : "bg-gray-200 text-gray-700 hover:bg-gray-300"
              }`}
            >
              {tag}
            </button>
          ))
        ) : (
          <p className="text-sm text-gray-500">Тегів поки немає</p>
```

```
        )}
        {selectedTags.length > 0 && (
          <button
            onClick={() => setSelectedTags([])}
            className="text-sm text-orange-500 hover:underline"
          >
            Очистити фільтр
          </button>
        )}
      </div>

      {/* Колонки */}
      <DragDropContext onDragEnd={handleDragEnd}>
        <div className="flex-1 w-full overflow-x-auto px-5 py-6">
          <div className="flex gap-6 min-w-max">
            {columnOrder.map((columnId) => {
              const column = displayedColumns[columnId];
              if (!column) return null; // Якщо колонки немає, пропускаємо
              return (
                <div
                  className="w-[300px] flex flex-col bg-white rounded-xl shadow-
md"
                  key={columnId}
                >
                  {/* Заголовок колонки */}
                  <div className="px-4 py-3 bg-gradient-to-r from-orange-400 to-
amber-300 text-white rounded-t-xl">
                    <h2 className="text-lg font-medium">{column.name}</h2>
                    <span className="text-xs opacity-80">
                      ({column.items.length} tasks)
                    </span>
                  </div>

                  {/* Область завдань */}
                  <Droppable droppableId={columnId}>
                    {(provided: any) => (
                      <div
                        ref={provided.innerRef}
                        {...provided.droppableProps}
                        className="p-4 flex-1 max-h-[calc(100vh-250px)]
overflow-y-auto bg-white"
                      >
                        {column.items.length === 0 ? (
                          <div className="text-center py-4 text-gray-500">
                            No tasks
                          </div>
                        ) : (
                          column.items.map((item: TaskT, index: number) => (
                            <Draggable
                              key={item.id}
                              draggableId={item.id}
                              index={index}
                            >
```

```
                                {(provided: any) => (
                                  <div
                                    ref={provided.innerRef}
                                    {...provided.draggableProps}
                                    className={`w-full mb-3 p-3 bg-white rounded-
lg border border-gray-200 hover:bg-orange-50 hover:border-orange-200 transition-
all duration-200 ${
                                      item.completed ? "opacity-60 line-through" :
""
                                    }`}
                                  >
                                    <div className="flex items-center justify-
between">
                                      <button
                                        onClick={() => openTaskModal(item,
columnId)}
                                        className="flex-1 text-left text-gray-800
font-medium"
                                      >
                                        {item.title || `Task ${index + 1}`}
                                      </button>
                                      <div className="flex items-center gap-2">
                                        <span
                                          className={`px-2 py-1 rounded-full text-
xs font-semibold ${getPriorityColor(
                                            item.priority
                                          )}`}
                                        >
                                          {item.priority || "Low"}
                                        </span>
                                        <div
                                          {...provided.dragHandleProps}
                                          className="cursor-grab text-gray-500
hover:text-orange-500"
                                        >
                                          <FaGripVertical size={16} />
                                        </div>
                                      </div>
                                    </div>
                                    {item.description && (
                                      <p className="text-sm text-gray-600 mt-1
truncate">
                                        {item.description}
                                      </p>
                                    )}
                                  </div>
                                )}
                              </Draggable>
                            ))
                          )}
                          {provided.placeholder}
                        </div>
                      )}
```

```
                  </Droppable>

                  {/* Кнопка "Add Task" */}
                  <button
                    onClick={() => {
                      setSelectedColumn(columnId);
                      openModal();
                    }}
                    className="m-3 flex items-center justify-center gap-2 py-2
px-4 bg-orange-400 text-white rounded-lg hover:bg-orange-500 transition-all
duration-200"
                  >
                    <IoAddOutline size={18} />
                    Add Task
                  </button>
                </div>
              );
            })}
          </div>
        </div>
      </DragDropContext>

      {/* Модалка для додавання завдання */}
      <AddModal
        isOpen={modalOpen}
        onClose={closeModal}
        setOpen={setModalOpen}
        handleAddTask={handleAddTask}
      />

      {/* Модалка для перегляду/редагування завдання */}
      {taskModalOpen && selectedTask && (
        <div className="fixed inset-0 bg-black bg-opacity-50 flex items-center
justify-center z-50">
          <div className="bg-white rounded-xl p-6 w-[500px] max-h-[80vh]
overflow-y-auto shadow-xl">
            <h3 className="text-xl font-semibold text-gray-800 mb-4">
              {isEditing
                ? "Edit Task"
                : selectedTask.title || "Task Details"}
            </h3>
            {isEditing ? (
              <div className="space-y-4">
                <input
                  type="text"
                  value={editedTask?.title || ""}
                  onChange={(e) =>
                    setEditedTask({ ...editedTask!, title: e.target.value })
                  }
                  className="w-full p-2 border rounded"
                />
                <textarea
                  value={editedTask?.description || ""}
```

```
                  onChange={(e) =>
                    setEditedTask({
                      ...editedTask!,
                      description: e.target.value,
                    })
                  }
                  className="w-full p-2 border rounded"
                />
                <select
                  value={editedTask?.priority || "low"}
                  onChange={(e) =>
                    setEditedTask({ ...editedTask!, priority: e.target.value })
                  }
                  className="w-full p-2 border rounded"
                >
                  <option value="low">Low</option>
                  <option value="medium">Medium</option>
                  <option value="high">High</option>
                </select>
                <input
                  type="datetime-local"
                  value={
                    editedTask?.deadline
                      ? new Date(editedTask.deadline)
                          .toISOString()
                          .slice(0, 16)
                      : ""
                  }
                  onChange={(e) =>
                    setEditedTask({
                      ...editedTask!,
                      deadline: new Date(e.target.value).getTime(),
                    })
                  }
                  className="w-full p-2 border rounded"
                />
                <input
                  type="text"
                  value={editedTask?.assignee || ""}
                  onChange={(e) =>
                    setEditedTask({ ...editedTask!, assignee: e.target.value })
                  }
                  placeholder="Assignee"
                  className="w-full p-2 border rounded"
                />
                <button
                  onClick={handleEditTask}
                  className="w-full py-2 bg-blue-500 text-white rounded
hover:bg-blue-600"
                >
                  Save
                </button>
              </div>
```

```
            ) : (
              <div className="space-y-4">
                <div>
                  <label className="text-sm text-gray-600 font-medium">
                    Title:
                  </label>
                  <p className="text-gray-800">{selectedTask.title}</p>
                </div>
                <div>
                  <label className="text-sm text-gray-600 font-medium">
                    Description:
                  </label>
                  <p className="text-gray-800 whitespace-pre-wrap">
                    {selectedTask.description}
                  </p>
                </div>
                <div>
                  <label className="text-sm text-gray-600 font-medium">
                    Priority:
                  </label>
                  <p
                    className={`inline-block px-2 py-1 rounded-full text-sm
font-semibold ${getPriorityColor(
                      selectedTask.priority
                    )}`}
                  >
                    {selectedTask.priority || "Low"}
                  </p>
                </div>
                <div>
                  <label className="text-sm text-gray-600 font-medium">
                    Deadline:
                  </label>
                  <p className="text-gray-800">
                    {new Date(selectedTask.deadline).toLocaleString()}
                  </p>
                </div>
                <div>
                  <label className="text-sm text-gray-600 font-medium">
                    Assignee:
                  </label>
                  <p className="text-gray-800">
                    {selectedTask.assignee || "Not assigned"}
                  </p>
                </div>
                {selectedTask.image && (
                  <div>
                    <label className="text-sm text-gray-600 font-medium">
                      Image:
                    </label>
                    <img
                      src={selectedTask.image}
                      alt={selectedTask.alt || "Task image"}
```

```
                    className="mt-2 max-w-full rounded-lg"
                  />
                </div>
              )}
              <div>
                <label className="text-sm text-gray-600 font-medium">
                  Tags:
                </label>
                <div className="flex flex-wrap gap-2 mt-2">
                  {selectedTask.tags.map((tag, index) => (
                    <span
                      key={index}
                      className={`${tag.bg} ${tag.text} px-2 py-1 rounded-full
text-sm`}
                    >
                      {tag.title}
                    </span>
                  ))}
                </div>
              </div>
              <div>
                <label className="text-sm text-gray-600 font-medium">
                  Status:
                </label>
                <p className="text-gray-800">
                  {selectedTask.completed ? "Completed" : "Pending"}
                </p>
              </div>
              <div>
                <label className="text-sm text-gray-600 font-medium">
                  Created:
                </label>
                <p className="text-gray-800">
                  {new Date(selectedTask.createdAt).toLocaleString()}
                </p>
              </div>
            </div>
          )}

          <div className="mt-6 flex gap-3">
            {!isEditing && (
              <button
                onClick={() => setIsEditing(true)}
                className="flex-1 py-2 bg-yellow-500 text-white rounded
hover:bg-yellow-600"
              >
                Edit
              </button>
            )}
            <button
              onClick={handleCompleteTask}
```

```
                  className="flex-1 py-2 bg-gradient-to-r from-green-500 to-
emerald-500 text-white rounded-lg hover:from-green-600 hover:to-emerald-600
transition-all"
                  disabled={selectedTask.completed}
                >
                  {selectedTask.completed ? "Completed" : "Complete"}
                </button>
                <button
                  onClick={handleDeleteTask}
                  className="flex-1 py-2 bg-gradient-to-r from-red-500 to-rose-500
text-white rounded-lg hover:from-red-600 hover:to-rose-600 transition-all"
                >
                  Delete
                </button>
              </div>
              <button
                onClick={closeTaskModal}
                className="w-full mt-3 py-2 bg-gray-200 text-gray-800 rounded-lg
hover:bg-gray-300 transition-colors"
              >
                Close
              </button>
            </div>
          </div>
        )}
      </div>
    );
};

export default Home;
```

## AddModal.tsx

```
/* eslint-disable @typescript-eslint/no-explicit-any */
import React, { useState, useEffect } from "react";
import { getRandomColors } from "../../helpers/getRandomColors";
import { v4 as uuidv4 } from "uuid";
import { TaskT } from "../../types";
import { tasksApi } from "../../services/api";

interface Tag {
  title: string;
  bg: string;
  text: string;
}

interface AddModalProps {
  isOpen: boolean;
  onClose: () => void;
  setOpen: React.Dispatch<React.SetStateAction<boolean>>;
  handleAddTask: (taskData: TaskT) => void;
}
```

```
const AddModal = ({ isOpen, onClose, setOpen, handleAddTask }: AddModalProps) =>
{
  const initialTaskData: TaskT = {
    id: uuidv4(),
    title: "",
    description: "",
    priority: "",
    deadline: 0,
    image: "",
    alt: "",
    tags: [],
    completed: false,
    createdAt: Date.now(),
    assignee: "",
  };

  const [taskData, setTaskData] = useState<TaskT>(initialTaskData);
  const [tagTitle, setTagTitle] = useState("");
  const [emailError, setEmailError] = useState<string | null>(null);
  const [isCheckingEmail, setIsCheckingEmail] = useState(false);

  // Перевірка email при зміні поля assignee
  useEffect(() => {
    const checkEmail = async () => {
      if (!taskData.assignee) {
        setEmailError(null);
        return;
      }

      setIsCheckingEmail(true);
      try {
        await tasksApi.checkEmail(taskData.assignee);
        setEmailError(null); // Email існує
      } catch (error: any) {
        setEmailError(error.response?.data?.message || 'Помилка перевірки
email');
      } finally {
        setIsCheckingEmail(false);
      }
    };

    const timeout = setTimeout(checkEmail, 500); // Дебонсинг для зменшення
запитів
    return () => clearTimeout(timeout);
  }, [taskData.assignee]);

  const handleChange = (
    e: React.ChangeEvent<HTMLInputElement | HTMLSelectElement |
HTMLTextAreaElement>
  ) => {
    const { name, value } = e.target;
    setTaskData({ ...taskData, [name]: name === "deadline" ? Number(value) :
value });
```

```
  };

  const handleImageChange = (e: React.ChangeEvent<HTMLInputElement>) => {
    if (e.target.files && e.target.files[0]) {
      const reader = new FileReader();
      reader.onload = function (e) {
        if (e.target) {
          setTaskData({ ...taskData, image: e.target.result as string });
        }
      };
      reader.readAsDataURL(e.target.files[0]);
    }
  };

  const handleAddTag = () => {
    if (tagTitle.trim() !== "") {
      const { bg, text } = getRandomColors();
      const newTag: Tag = { title: tagTitle.trim(), bg, text };
      setTaskData({ ...taskData, tags: [...taskData.tags, newTag] });
      setTagTitle("");
    }
  };

  const closeModal = () => {
    setOpen(false);
    onClose();
    setTaskData(initialTaskData);
    setEmailError(null);
  };

  const handleSubmit = () => {
    if (!taskData.title || !taskData.description || !taskData.priority ||
!taskData.deadline) {
      alert("Будь ласка, заповніть усі обов’язкові поля (Назва, Опис, Пріоритет,
Дедлайн)");
      return;
    }

    if (emailError) {
      alert("Будь ласка, виправте помилку з email відповідального");
      return;
    }

    handleAddTask({ ...taskData, createdAt: Date.now() });
    closeModal();
  };

  return (
    <div
      className={`w-screen h-screen place-items-center fixed top-0 left-0
${isOpen ? "grid" : "hidden"}`}
    >
      <div
```

```
        className="w-full h-full bg-black opacity-70 absolute left-0 top-0 z-20"
        onClick={closeModal}
      ></div>
      <div className="md:w-[30vw] w-[90%] bg-white rounded-lg shadow-md z-50
flex flex-col items-center gap-3 px-5 py-6">
        <input
          type="text"
          name="title"
          value={taskData.title}
          onChange={handleChange}
          placeholder="Назва *"
          className="w-full h-12 px-3 outline-none rounded-md bg-gray-100 border
border-gray-300 text-sm font-medium focus:border-orange-400"
        />
        <textarea
          name="description"
          value={taskData.description}
          onChange={handleChange}
          placeholder="Опис *"
          className="w-full h-20 px-3 py-2 outline-none rounded-md bg-gray-100
border border-gray-300 text-sm font-medium focus:border-orange-400 resize-none"
        />
        <select
          name="priority"
          onChange={handleChange}
          value={taskData.priority}
          className="w-full h-12 px-2 outline-none rounded-md bg-gray-100 border
border-gray-300 text-sm focus:border-orange-400"
        >
          <option value="">Оберіть пріоритет *</option>
          <option value="low">Низький</option>
          <option value="medium">Середній</option>
          <option value="high">Високий</option>
        </select>
        <input
          type="datetime-local"
          name="deadline"
          value={
            taskData.deadline
              ? new Date(taskData.deadline).toISOString().slice(0, 16)
              : ""
          }
          onChange={(e) =>
            setTaskData({ ...taskData, deadline: new
Date(e.target.value).getTime() })
          }
          className="w-full h-12 px-3 outline-none rounded-md bg-gray-100 border
border-gray-300 text-sm focus:border-orange-400"
        />
        <div className="w-full">
          <input
            type="email"
            name="assignee"
```

```
            value={taskData.assignee || ""}
            onChange={handleChange}
            placeholder="Email відповідального (необов’язково)"
            className={`w-full h-12 px-3 outline-none rounded-md bg-gray-100
border ${emailError ? 'border-red-500' : 'border-gray-300'} text-sm
focus:border-orange-400`}
            disabled={isCheckingEmail}
          />
          {emailError && <p className="text-red-500 text-sm mt-
1">{emailError}</p>}
          {isCheckingEmail && <p className="text-gray-500 text-sm mt-
1">Перевірка email...</p>}
        </div>
        <input
          type="text"
          value={tagTitle}
          onChange={(e) => setTagTitle(e.target.value)}
          placeholder="Назва тегу"
          className="w-full h-12 px-3 outline-none rounded-md bg-gray-100 border
border-gray-300 text-sm focus:border-orange-400"
        />
        <button
          className="w-full rounded-md h-9 bg-gradient-to-r from-orange-400 to-
amber-300 text-white font-medium hover:from-orange-500 hover:to-amber-400
transition-all"
          onClick={handleAddTag}
        >
          Додати тег
        </button>
        <div className="w-full">
          {taskData.tags.length > 0 && <span className="text-sm text-gray-
600">Теґи:</span>}
          <div className="flex flex-wrap gap-2 mt-1">
            {taskData.tags.map((tag, index) => (
              <div
                key={index}
                className="inline-block px-3 py-1 text-[13px] font-medium
rounded-full"
                style={{ backgroundColor: tag.bg, color: tag.text }}
              >
                {tag.title}
              </div>
            ))}
          </div>
        </div>
        <div className="w-full flex items-center gap-4 justify-between">
          <input
            type="text"
            name="alt"
            value={taskData.alt}
            onChange={handleChange}
            placeholder="Опис зображення"
```

```
            className="w-full h-12 px-3 outline-none rounded-md bg-gray-100
border border-gray-300 text-sm focus:border-orange-400"
          />
          <input
            type="file"
            name="image"
            onChange={handleImageChange}
            className="w-full text-sm text-gray-600"
          />
        </div>
        <button
          className="w-full mt-3 rounded-md h-9 bg-gradient-to-r from-orange-400
to-amber-300 text-white font-medium hover:from-orange-500 hover:to-amber-400
transition-all"
          onClick={handleSubmit}
          disabled={isCheckingEmail}
        >
          Додати завдання
        </button>
      </div>
    </div>
  );
};

export default AddModal;
```

## AuthContext.tsx

```
import { createContext, useContext, useState, ReactNode } from "react";
import { Navigate } from "react-router-dom";

interface AuthContextType {
  user: string | null;
  login: (user: string) => void;
  logout: () => void;
}

const AuthContext = createContext<AuthContextType | undefined>(undefined);

export function AuthProvider({ children }: { children: ReactNode }) {
  const [user, setUser] = useState<string | null>(null);

  const login = (userData: string) => {
    setUser(userData);
    localStorage.setItem("user", userData);
  };

  const logout = () => {
    setUser(null);
    localStorage.removeItem("user");
    localStorage.removeItem("token");
  };
```

```
  return (
    <AuthContext.Provider value={{ user, login, logout }}>
      {children}
    </AuthContext.Provider>
  );
}

export function useAuth() {
  const context = useContext(AuthContext);
  if (!context) {
    throw new Error("useAuth must be used within an AuthProvider");
  }
  return context;
}

export function ProtectedRoute({ children }: { children: ReactNode }) {
  const { user } = useAuth();

  if (!user) {
    return <Navigate to="/login" replace />;
  }

  return <>{children}</>;
}
```

## SearchContext.tsx

```
import { createContext, useContext, useState, ReactNode } from "react";
import debounce from "lodash/debounce";

interface SearchContextType {
  searchText: string;
  setSearchText: (text: string) => void;
  selectedTags: string[];
  setSelectedTags: React.Dispatch<React.SetStateAction<string[]>>; // Оновлений
тип
  clearSearch: () => void;
}

const SearchContext = createContext<SearchContextType | undefined>(undefined);

export const SearchProvider = ({ children }: { children: ReactNode }) => {
  const [searchText, setSearchText] = useState("");
  const [selectedTags, setSelectedTags] = useState<string[]>([]);

  const debouncedSetSearchText = debounce((text: string) => {
    setSearchText(text);
  }, 300);

  const clearSearch = () => {
    setSearchText("");
    setSelectedTags([]);
  };
```

```
  return (
    <SearchContext.Provider
      value={{
        searchText,
        setSearchText: debouncedSetSearchText,
        selectedTags,
        setSelectedTags, // Передаємо функцію setSelectedTags напряму
        clearSearch,
      }}
    >
      {children}
    </SearchContext.Provider>
  );
};

export const useSearch = () => {
  const context = useContext(SearchContext);
  if (!context) {
    throw new Error("useSearch must be used within a SearchProvider");
  }
  return context;
};
```

## App.tsx

```
import { Routes, Route } from "react-router-dom";
import { AuthProvider, ProtectedRoute } from "./context/AuthContext";
import { SearchProvider } from "./context/SearchContext"; // Додаємо
SearchProvider
import Layout from "./layout";
import Boards from "./pages/Boards";
import Notifications from "./pages/Notifications";
import Login from "./pages/Auth/Login";
import Register from "./pages/Auth/Register";

function App() {
  return (
    <AuthProvider>
      <SearchProvider> {/* Додаємо SearchProvider */}
        <Routes>
          <Route path="/login" element={<Login />} />
          <Route path="/register" element={<Register />} />
          <Route
            path="/"
            element={
              <ProtectedRoute>
                <Layout />
              </ProtectedRoute>
            }
          >
            <Route index element={<Boards />} />
            <Route path="notifications" element={<Notifications />} />
```

```
          </Route>
        </Routes>
      </SearchProvider>
    </AuthProvider>
  );
}

export default App;
```

## onDragEnd.ts

```
import { DropResult } from "@hello-pangea/dnd";
import { Columns, TaskT } from "../types";
import { tasksApi } from "../services/api";

export const onDragEnd = async (
  result: DropResult,
  columns: Columns,
  setColumns: React.Dispatch<React.SetStateAction<Columns>>,
  setTaskModalOpen: React.Dispatch<React.SetStateAction<boolean>>,
  setSelectedTask: React.Dispatch<React.SetStateAction<TaskT | null>>,
  setSelectedColumn: React.Dispatch<React.SetStateAction<string>>,
  setEditedTask: React.Dispatch<React.SetStateAction<TaskT | null>>,
  setIsEditing: React.Dispatch<React.SetStateAction<boolean>>
) => {
  const { source, destination } = result;

  // Якщо немає пункту призначення, нічого не робимо
  if (!destination) return;

  const sourceColumn = columns[source.droppableId];
  const destColumn = columns[destination.droppableId];
  const sourceItems = [...sourceColumn.items];
  const destItems = [...destColumn.items];

  // Видаляємо завдання з вихідної колонки
  const [movedTask] = sourceItems.splice(source.index, 1);

  // Додаємо завдання до цільової колонки
  if (source.droppableId === destination.droppableId) {
    // Переміщення в межах однієї колонки
    sourceItems.splice(destination.index, 0, movedTask);
  } else {
    // Переміщення між колонками
    destItems.splice(destination.index, 0, movedTask);

    // Оновлюємо статус завдання на бекенді
    try {
      await tasksApi.updateTask(movedTask.id, {
        status: destination.droppableId,
      });
    } catch (error) {
      console.error("Помилка оновлення статусу завдання:", error);
```

```
      // Відкат змін у разі помилки
      return;
    }
  }

  // Оновлюємо стан колонок
  const newColumns = {
    ...columns,
    [source.droppableId]: {
      ...sourceColumn,
      items: sourceItems,
    },
    [destination.droppableId]: {
      ...destColumn,
      items: destItems,
    },
  };

  setColumns(newColumns);

  // Закриваємо модалку, якщо вона відкрита
  setTaskModalOpen(false);
  setSelectedTask(null);
  setSelectedColumn("");
  setEditedTask(null);
  setIsEditing(false);
};
```

## Notification.tsx

```
import { useState, useEffect } from "react";

interface Notification {
  id: string;
  message: string;
  timestamp: number;
}

const Notifications = () => {
  const [notifications, setNotifications] = useState<Notification[]>([]);

  // Завантажуємо сповіщення з localStorage при монтуванні компонента
  useEffect(() => {
    const storedNotifications = localStorage.getItem("notifications");
    if (storedNotifications) {
      setNotifications(JSON.parse(storedNotifications));
    }
  }, []);

  // Очищаємо сповіщення
  const clearNotifications = () => {
    setNotifications([]);
    localStorage.removeItem("notifications");
```

```
  };

  return (
    <div className="min-h-screen bg-[#f5e9e2] flex flex-col pt-[70px] pl-[60px]
md:pl-[230px]">
      <div className="flex-1 px-5 py-6">
        {notifications.length === 0 ? (
          <div className="text-center py-4 text-gray-500">
            No notifications yet
          </div>
        ) : (
          <div className="space-y-4">
            <div className="bg-white rounded-xl shadow-md p-4">
              {notifications.map((notification) => (
                <div
                  key={notification.id}
                  className="py-2 border-b border-gray-200 last:border-b-0"
                >
                  <p className="text-gray-800">{notification.message}</p>
                  <p className="text-sm text-gray-500 mt-1">
                    {new Date(notification.timestamp).toLocaleString()}
                  </p>
                </div>
              ))}
            </div>
            <button
              onClick={clearNotifications}
              className="mt-4 py-2 px-4 bg-orange-400 text-white rounded-lg
hover:bg-orange-500 transition-all duration-200"
            >
              Clear Notifications
            </button>
          </div>
        )}
      </div>
    </div>
  );
};

export default Notifications;
```

## Navbar.tsx

```
import { Link } from "react-router-dom";
import {
  IoChevronDownOutline,
  IoPersonOutline,
  IoSearchOutline,
  IoShareSocialOutline,
  IoSettingsOutline,
  IoNotificationsOutline,
  IoCloseOutline,
} from "react-icons/io5";
```

```
import { useSearch } from "../../context/SearchContext"; // Додаємо useSearch

const Navbar = () => {
  const { searchText, setSearchText, clearSearch } = useSearch();

  const handleSearchChange = (e: React.ChangeEvent<HTMLInputElement>) => {
    const text = e.target.value;
    setSearchText(text);
  };

  const handleClearSearch = () => {
    clearSearch();
  };

  return (
    <div className="md:w-[calc(100%-230px)] w-[calc(100%-60px)] fixed flex
items-center justify-between pl-2 pr-6 h-[70px] top-0 md:left-[230px] left-
[60px] border-b border-slate-300 bg-[#fff]">
      <div className="flex items-center gap-3 cursor-pointer">
        <IoPersonOutline color="#fb923c" width={"28px"} height={"28px"} />
        <span className="text-orange-400 font-semibold md:text-lg text-sm
whitespace-nowrap">
          Taskly
        </span>
        <IoChevronDownOutline color="#fb923c" width={"16px"} height={"16px"} />
      </div>
      <div className="md:w-[800px] w-[130px] bg-gray-100 rounded-lg px-3 py-
[10px] flex items-center gap-2 relative">
        <IoSearchOutline color={"#999"} />
        <input
          type="text"
          placeholder="Search by title or tags"
          className="w-full bg-gray-100 outline-none text-[15px]"
          value={searchText}
          onChange={handleSearchChange}
        />
        {searchText && (
          <button
            onClick={handleClearSearch}
            className="absolute right-3 text-gray-500 hover:text-gray-700"
          >
            <IoCloseOutline size={18} />
          </button>
        )}
      </div>
      <div className="md:flex hidden items-center gap-4">
        <div className="grid place-items-center bg-gray-100 rounded-full p-2
cursor-pointer">
          <IoShareSocialOutline color={"#444"} />
        </div>
        <div className="grid place-items-center bg-gray-100 rounded-full p-2
cursor-pointer">
          <IoSettingsOutline color={"#444"} />
```

```
        </div>
        <Link to="/notifications">
          <div className="grid place-items-center bg-gray-100 rounded-full p-2
cursor-pointer">
            <IoNotificationsOutline color={"#444"} />
          </div>
        </Link>
      </div>
    </div>
  );
};

export default Navbar;
```

## Task.tsx

```
import { IoTimeOutline } from "react-icons/io5";
import { TaskT } from "../../types";

interface TaskProps {
  task: TaskT;
  provided: any;
}

const Task = ({ task, provided }: TaskProps) => {
  const { title, description, priority, deadline, image, alt, tags } = task;

  // Форматування дедлайну
  const formattedDeadline = new Date(deadline).toLocaleString();

  return (
    <div
      ref={provided.innerRef}
      {...provided.draggableProps}
      {...provided.dragHandleProps}
      className="w-full cursor-grab bg-[#fff] flex flex-col justify-between gap-
3 items-start shadow-sm rounded-xl px-3 py-4"
    >
      {image && alt && (
        <img src={image} alt={alt} className="w-full h-[170px] rounded-lg" />
      )}
      <div className="flex items-center gap-2">
        {tags.map((tag) => (
          <span
            key={tag.title}
            className="px-[10px] py-[2px] text-[13px] font-medium rounded-md"
            style={{ backgroundColor: tag.bg, color: tag.text }}
          >
            {tag.title}
          </span>
        ))}
      </div>
      <div className="w-full flex items-start flex-col gap-0">
```

```
        <span className="text-[15.5px] font-medium text-[#555]">{title}</span>
        <span className="text-[13.5px] text-gray-500">{description}</span>
      </div>
      <div className="w-full border border-dashed"></div>
      <div className="w-full flex items-center justify-between">
        <div className="flex items-center gap-1">
          <IoTimeOutline color={"#666"} width="19px" height="19px" />
          <span className="text-[13px] text-gray-700">{formattedDeadline}</span>
        </div>
        <div
          className={`w-[60px] rounded-full h-[5px] ${
            priority === "high"
              ? "bg-red-500"
              : priority === "medium"
              ? "bg-orange-500"
              : "bg-blue-500"
          }`}
        ></div>
      </div>
    </div>
  );
};

export default Task;
```

ДОДАТОК В

(обов’язковий)

# ПРЕЗЕНТАЦІЙНІ СЛАЙДИ

**Хмельницький національний університет**
**Факультет інформаційних технологій**
**Кафедра інженерії програмного забезпечення**

Кваліфікаційна робота на тему:

«Вебсистема для організації та контролю виконання проєктних завдань»

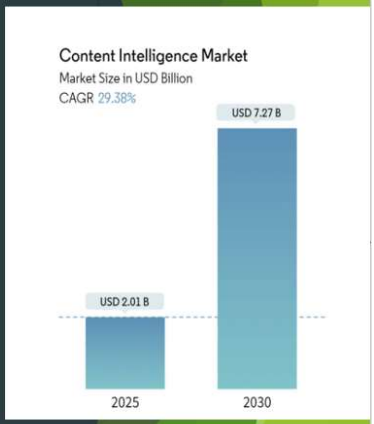
Підготував:
студент групи ІПЗ-21-1
Мазур Дмитрій Ігорович

Керівник кваліфікаційної роботи
канд. техн. наук, доцент
Форкун Юрій Вікторович

## Актуальність теми

2

У сучасному світі, де технології визначають темп розвитку бізнесу та командної роботи, ефективне управління проєктами стає ключовим фактором успіху. Актуальність теми підкреслюється стрімким зростанням попиту на цифрові інструменти, особливо в умовах віддаленої роботи, що стала нормою для багатьох організацій. За даними Mordor Intelligence, у 2025 році глобальний ринок програмного забезпечення для управління проєктами зростає на 11,9%, відображаючи потребу в інноваційних рішеннях.




Важливість розробки вебсистеми для організації та контролю проєктних завдань полягає в її здатності забезпечити чітке відстеження статусу завдань, адаптивний інтерфейс із drag-and-drop функціоналом, який підходить для користувачів із різним рівнем підготовки, а також економію до 30% часу на рутинні операції.

# Мета та Завдання

Метою роботи є розробка вебсистеми для організації та контролю виконання проєктних завдань, яка забезпечить зручне управління завданнями, прозорість робочих процесів і адаптивність для користувачів із різними рівнями доступу.

| Етап | Завдання |
|---|---|
| **Аналіз** | Вивчення сучасних підходів до управління проєктами та потреб команд. |
| **Визначення вимог** | Формування переліку функціональних і нефункціональних вимог до системи. |
| **Проєктування** | Розробка архітектури системи на основі MVC для забезпечення масштабованості. |
| **Вибір технологій** | Аналіз і обґрунтування використання MERN-стеку для ефективної реалізації. |
| **Розробка** | Створення модулів авторизації, управління завданнями та дошки Kanban. |
| **Тестування** | Перевірка стабільності, продуктивності та коректності роботи системи. |
| **Оцінка** | Аналіз результатів і визначення напрямків подальшого розвитку |



# Змістовий аналіз предметної області, її структурних та функціональних особливостей

Аналіз предметної області розкриває сучасні виклики управління проєктами, де цифрові інструменти стають ключем до організації робочих процесів у світі віддаленої роботи. Предметна область зосереджується на потребах команд у координації завдань і прозорості, що набуває особливого значення в умовах глобальних змін. Усе частіше компанії звертаються до інноваційних рішень, таких як дошки завдань, для підвищення ефективності та адаптації до динамічних умов. Цей аналіз закладає основу для створення вебсистеми, яка відповідатиме сучасним потребам команд у гнучкому та прозорому управлінні проєктами.

# Аналіз наявного програмно-технічного забезпечення



**Порівння функцій вже існуючих вебсистем**

| Функціонал | Trello | Asana | Jira |
|---|---|---|---|
| **Реєстрація та авторизація** | + | + | + |
| **Дошка завдань (Kanban)** | + | + | + |
| **Переміщення завдань (drag-and-drop)** | + | + | + |
| **Фільтрація завдань за статусом/пріоритетом** | +/- (обмежено без преміум) | + | + |
| **Розмежування доступу (ролі)** | +/- (обмежено без преміум) | + | + |
| **Адаптивність для мобільних пристроїв** | + | + | + |
| **Інтеграція з іншими сервісами** | +/- (обмежено без преміум) | + | + |
| **Розширені фільтри** | - | - | +/- (обмежено) |
| **Кастомізація інтерфейсу (зміна тем)** | - | +/- (обмежено без преміум) | + |

# Визначення функціональних та нефункціональних вимог до ПЗ



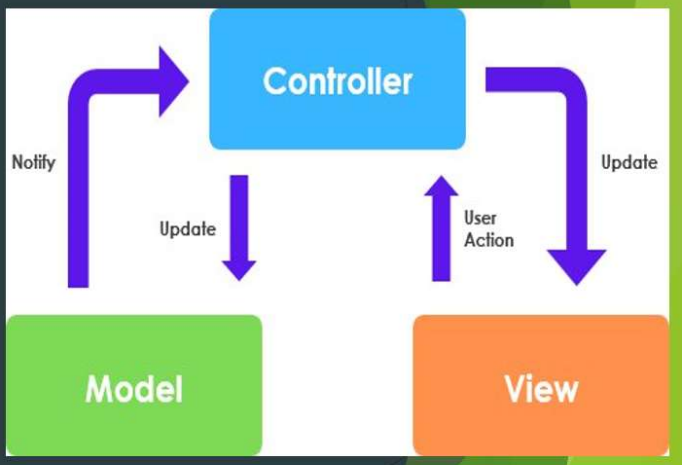
**Функціональні та не функціональні вимоги**

| Функціональні вимоги | Нефункціональні вимоги |
|---|---|
| Реєстрація та авторизація користувачів | Час завантаження сторінки < 3 секунди. |
| Створення, редагування та видалення завдань. | Адаптивність інтерфейсу для мобільних пристроїв. |
| Прикріплення фото та файлів до завдань | Стійкість до помилок авторизації або створення завдань з повідомленнями. |
| Переміщення завдань між колонками (Kanban) | Код 200 для коректних API-запитів. |
| Фільтрація завдань за пріоритетом і статусом | Захист від несанкціонованого доступу через коректну обробку JWT-токенів. |
| Розмежування доступу та відповідального за завдання | Захист паролів користувачів через хешування за стандартом bcrypt. |
| Сповіщення про зміни і нагадування | Сучасний дизайн із можливістю зміни тем для кращого користувацького досвіду. |
| Пошук завдань по тегах або назвах | |

# Вибір типу архітектури та шаблонів проєктування



Для вебсистеми обрано архітектуру MVC (Model-View-Controller), яка діє як диригент оркестру, гармонійно поєднуючи логіку, інтерфейс і дані. Model управляє даними та бізнес-логікою, View створює зручний інтерфейс для користувача, а Controller обробляє запити, забезпечуючи плавну взаємодію. Це дозволяє системі легко масштабуватися, спрощує оновлення дошки Kanban і гарантує гнучкість для майбутніх змін.




Діаграма архітектурного патерну MVC

# Опис декомпозиції, залежностей, інтерфейсів



**Декомпозиція**:

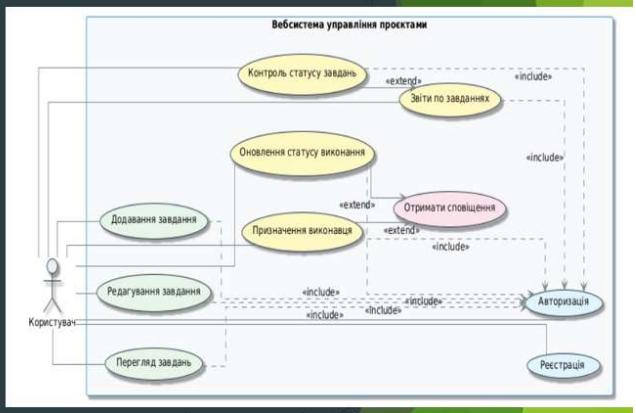
- модуль авторизації (реєстрація, логін);
- модуль управління завданнями (створення, редагування);
- модуль дошки Kanban (відображення статусів, перетягування);
- модуль сповіщення.

**Залежності**:

- MongoDB як центральна база даних для зберігання даних завдань і профілів.

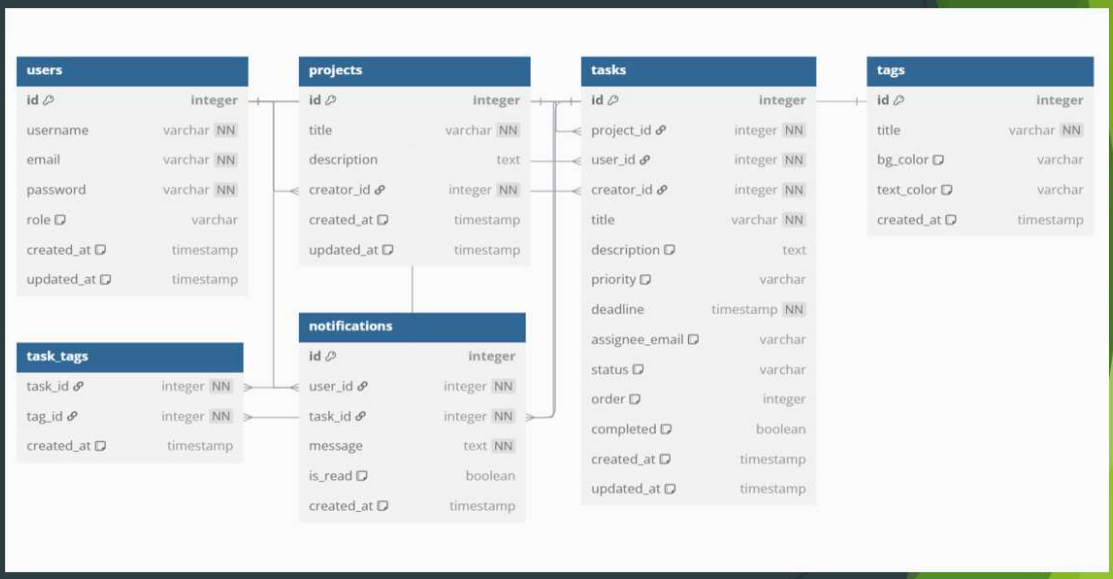
**Інтерфейси**:

- реалізовано через React із респонсивним дизайном;
- панель навігації для швидкого доступу до дошки, профілю, звітів.




Діаграма варіантів використання

# Проєктування модулів і даних



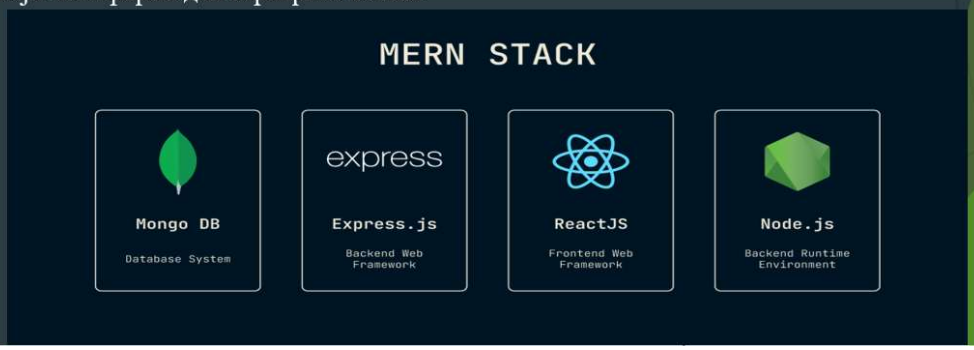

ER-діаграма



# Аналіз та вибір технологій

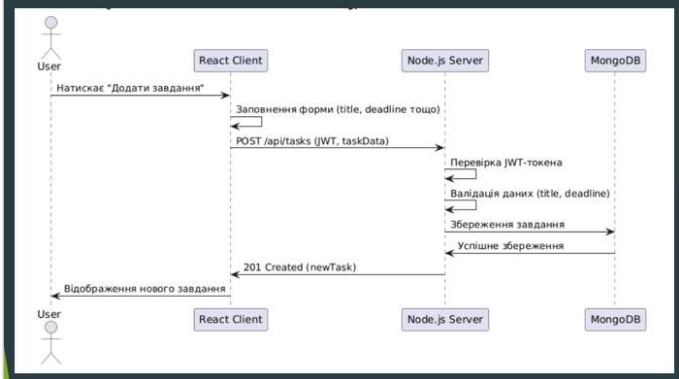
Аналіз та вибір технологій для вебсистеми став вирішальним етапом, адже сучасні проєкти потребують інструментів, які поєднують швидкість, гнучкість і зручність. Для розробки обрано MERN-стек, що включає MongoDB, Express.js, React і Node.js.

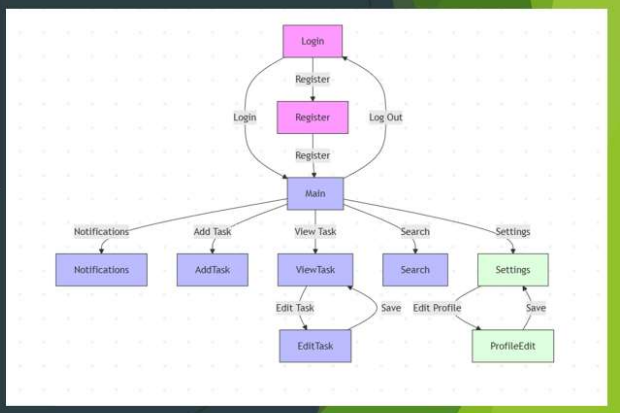
- MongoDB: NoSQL-база для зберігання даних
- Express.js: фреймворк для створення серверних маршрутів.
- React: бібліотека для інтерактивного інтерфейсу.
- Node.js: платформа для серверної логіки.



# Реалізація модулів і база даних

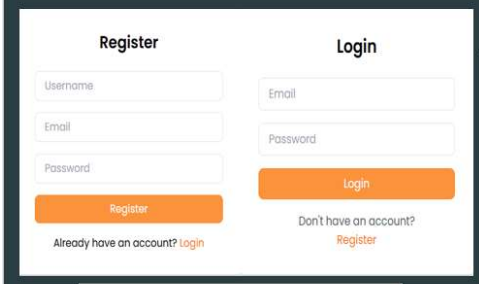


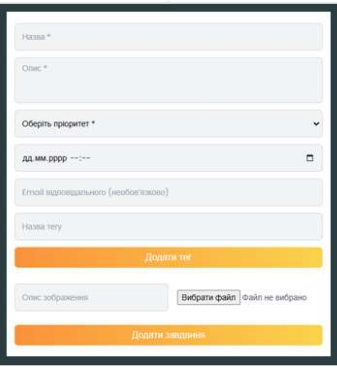


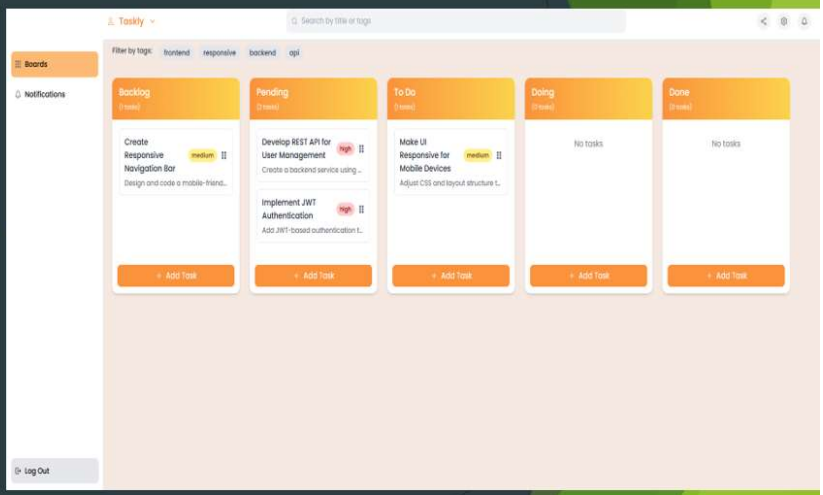




# Реалізація модулів і база даних





# Вимоги до технічного та програмного забезпечення



**Програмні вимоги**

Для роботи системи необхідні сучасні інструменти та стабільне програмне забезпечення:

- Node.js (версія 18 або вище) для серверної частини.
- MongoDB (локально або через MongoDB Atlas) для бази даних.
- Сучасний веббраузер (Chrome, Firefox, Edge версії не нижче 2023 року) для клієнтської частини.
- Відсутність блокування портів (наприклад, 3000) брандмауером.
- Стабільне інтернет-з'єднання для обміну даними між клієнтом і сервером.

**Технічні вимоги**

Система потребує мінімальних ресурсів для безперебійної роботи:

- Процесор: не менше 2 ядер із частотою 2.0 GHz.
- Оперативна пам'ять: 4 GB для сервера, 2 GB для клієнта.
- Дисковий простір: 10 GB для даних і логів.
- Мінімальна роздільна здатність екрана: 1280x720.
- Стабільне інтернет-з'єднання для взаємодії з сервером.

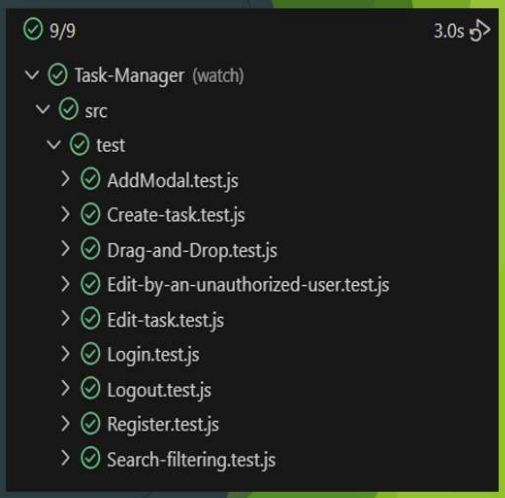
# Тестування ПЗ



Тестування вебсистеми було проведено для перевірки її стабільності, продуктивності та зручності використання в реальних умовах.

Інструменти для тестування:

- Jest для модульного тестування клієнтської частини (React-компонентів).
- Supertest для перевірки серверних API на базі Express.js.
- MongoDB Memory Server для симуляції бази даних під час тестів.



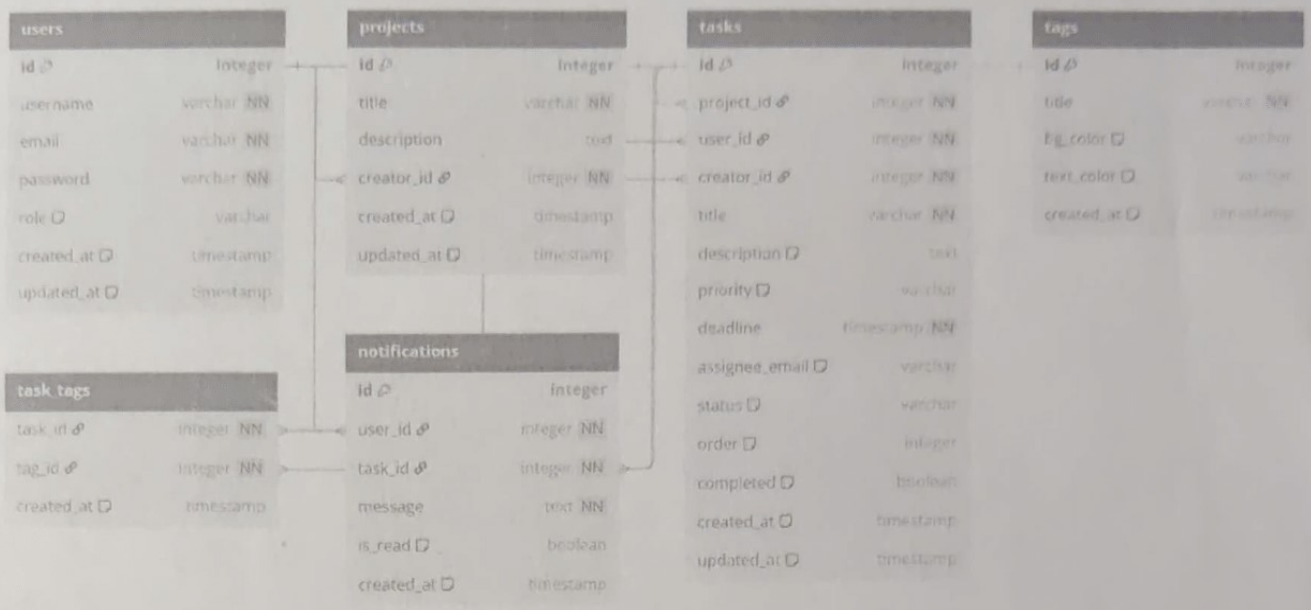
# Висновки



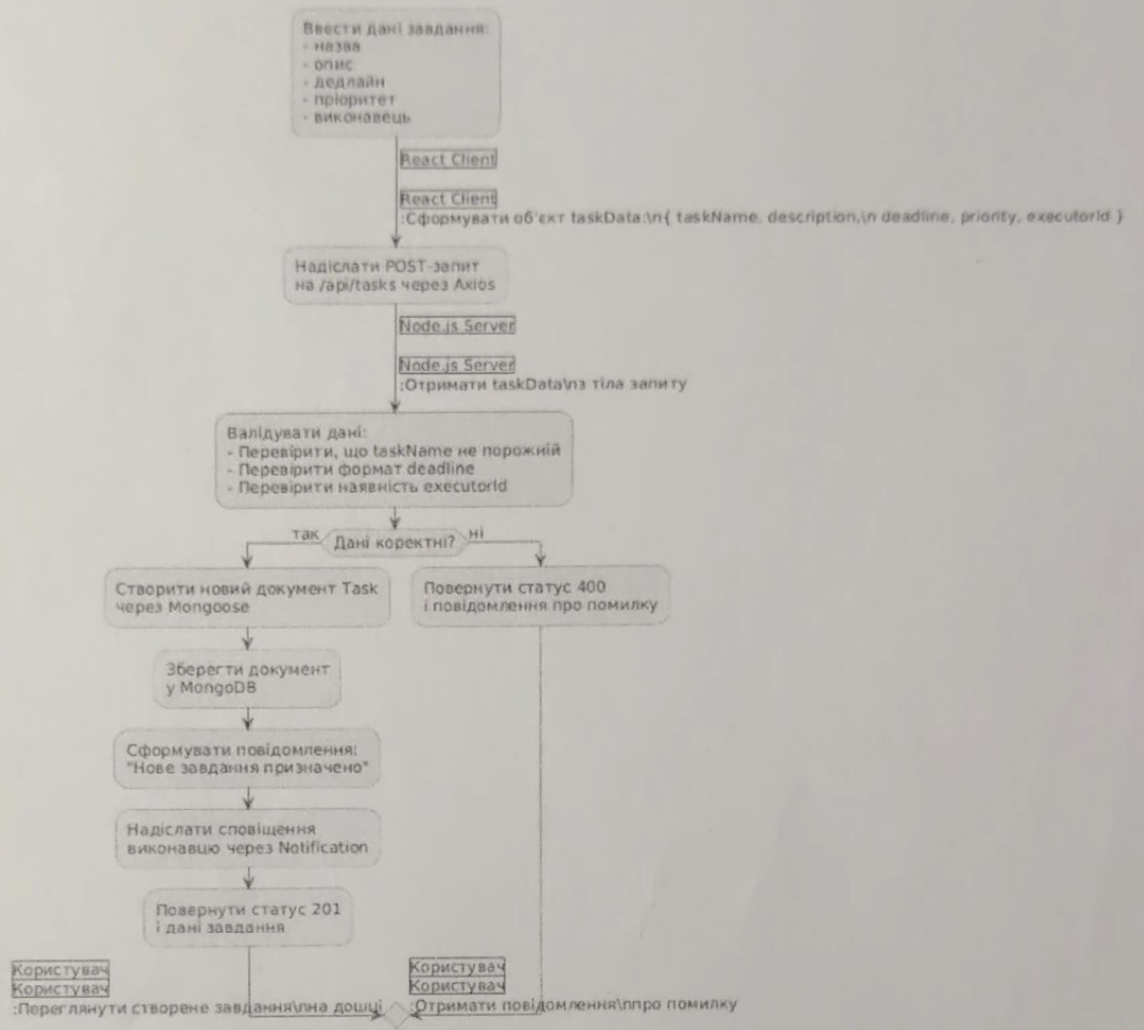
| Поставлене завдання | Результат виконання |
|---|---|
| Провести аналіз предметної області та сучасних підходів до управління проєктами. | Досліджено сучасні інструменти, такі як Kanban, виявлено потребу в прозорості. |
| Сформувати функціональні та нефункціональні вимоги до системи. | Складено вимоги: авторизація, фільтрація завдань, адаптивність, швидкість роботи. |
| Спроєктувати архітектуру системи на основі патерну MVC. | Розроблено MVC-архітектуру для поділу логіки, даних і інтерфейсу. |
| Обгрунтувати вибір стеку технологій (MERN). | Вибрано MERN-стек за гнучкість, швидкість і цілісність JavaScript. |
| Реалізувати основні модулі вебсистеми: авторизацію, управління завданнями, дошку Kanban. | - Авторизація: Втілено систему входу/реєстрації з використанням JWT для безпечного доступу, інтеграцією bcrypt для хешування паролів та захистом від несанкціонованих дій.<br>- Управління завданнями: Створено функціонал для створення, редагування та видалення завдань із підтримкою фільтрації за пріоритетом і статусом, а також логікою для збереження змін у MongoDB.<br>- Дошка Kanban: Розроблено інтерактивну дошку з drag-and-drop переміщенням завдань між колонками, реалізовану через React-beautiful-dnd, із реальним оновленням статусів. |
| Здійснити програмну реалізацію клієнтської та серверної частини | - Клієнтська частина: Реалізовано на React із маршрутизацією через React Router, інтерактивним інтерфейсом і стилізацією через Tailwind CSS.<br>- Серверна частина: Побудовано на Node.js і Express.js із REST API, інтеграцією MongoDB через Mongoose і обробкою запитів через middleware. |
| Виконати тестування для забезпечення стабільності та продуктивності системи. | Проведено тестування: завантаження < 3 секунд, API-запити стабільні, безпека забезпечена. |

Дякую за увагу!

# ГРАФІЧНА ЧАСТИНА




| | | | | | | |
|---|---|---|---|---|---|---|
| | | | | | КвРІПЗ.2101078.01.07.ПЗ | |
| Змн. | Арк. | № докум. | Підпис | Дата | Вебсистема для організації та контролю виконання проєктних завдань | Літ. / Маса. / Масштаб |
| Розробив | | Мазур Д. І. | | 28.05 | | |
| Керівник | | Форкун Ю. В. | | 28.05 | Логічна модель бази даних | Аркуш 1 / Аркушів 2 |
| Консульт. | | | | | | |
| Перевірив | | | | | | ХНУ, ІПЗ-21-1 |
| Н. Контр. | | Яшина О. М. | | 28.05 | | |
| Зав. каф. | | Бедратюк Л. П. | | 28.05 | | |




| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | КвРІПЗ.2101078.01.07.ПЗ | | |
| Змн. | Арк. | № докум. | Підпис | Дата | Вебсистема для організації та контролю виконання проєктних завдань<br>Блок-схема алгоритму створення завдання | Літ. | Маса. Масштаб |
| *Розробив* | | Мазур Д. І. | | 21.05 | | | |
| *Керівник* | | Форкун Ю. В. | | 28.05 | | Аркуш 2 | Аркушів 2 |
| *Консульт.* | | | | | | | |
| Перевірив | | | | | | | |
| *Н. Контр.* | | Яшина О. М. | | 28.05 | | ХНУ, ІПЗ-21-1 | |
| *Зав. каф.* | | Бедратюк Л. П. | | 28.05 | | | |

# СУПРОВІДНІ ДОКУМЕНТИ

Завідувачу кафедри інженерії програмного
забезпечення проф. Леоніду БЕДРАТЮКУ
здобувача вищої освіти
Мазура Дмитрія Ігоровича
факультет ІТ, IV курс, група ІПЗ-21-1

**ЗАЯВА**

З правилами чинного Положення про систему забезпечення академічної доброчесності в Хмельницькому національному університеті, згідно з яким виявлення академічного плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту і застосування заходів дисциплінарної та академічної відповідальності, ознайомлений. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на наявність академічного плагіату оповіщений та надаю свою згоду на обробку й збереження університетом моєї роботи в інституційному репозитарії Хмельницького національного університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-обчислювального комплексу StrikePlagiarism та/або програмно-технічного засобу AntiPlagiarism і використання роботи для виявлення академічного плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення текстових збігів у роботах.

Робота надається для перевірки в електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

29.05.2025
дата

підпис

# Anti-Plagiarism (UA) v-15.281 Educational

**The maximum coincidence with one document 4.0%**

Dictionaries check: en_US, ru_RU, ua_UA. **Errors in the documents: 13%**

| | Document | | Sum coincidence on the DB | |
|---|---|---|---|---|
| ID: 242376<br>Title: БКР_Вебсистема для організації та контролю виконання проєктних завдань<br>Added in a DB: 2025-05-29<br>Authors: Мазур Дмитрій<br>Heads: Форкун Ю.В., кад. техн.наук, доцент<br>Consultants:<br>Opponents: | Symbols | Lexemes | Symbols | Lexemes |
| | 95902 | 1422 | 4515 (5%) | 60 (4%) |

Plagiarism sources

| ID | Description | Plagiarism presence in the document | |
|---|---|---|---|
| | | Symbols | Lexemes |

**Протокол аналізу звіту подібності науковим керівником**

Заявляю, що я ознайомився (-лась) з Повним звітом подібності, який був згенерований Системою виявлення і запобігання плагіату щодо роботи:

**Автор:** Мазур Дмитро

**Співавтор:**

**Назва:** БКР_Вебсистема для організації та контролю виконання проєктних завдань

**Науковий керівник:**

**Підрозділ:** Кафедра інженерії програмного забезпечення

**Коефіцієнт подібності 1:**3.8%

**Коефіцієнт подібності 2:**0%

**Мікропробіли:** 0

**Заміна букв:** 0

**Інтервали:** 0

**Білі знаки:** 4

**Дата створення звіту:** 2025-05-28 21:19:16.0

**Після аналізу Звіту подібності констатую наступне:**

☑ **Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається.**

☐ **Запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Таким чином робота повертається на доопрацювання.**

☐ **Виявлено запозичення і плагіат або навмисні текстові спотворення (маніпуляції), як передбачувані спроби укриття плагіату, які роблять роботу невідповідною вимогам законодавства (Ст. 32. ЗУ Про вищу освіту, пункт 3.1, Ст. 42. ЗУ Про освіту) та вимог НАЗЯВО (Критерій 5), а також кодексу етики і процедурам. Таким чином робота не приймається.**

Обґрунтування:

*Дата* експерт

28.05.2025

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

**РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ**
освітнього ступеня «Бакалавр»

Дипломник Мазур Дмитрій Ігорович

Тема Вебсистема для організації та контролю виконання проєктних завдань

Спеціальність 121 – Інженерія програмного забезпечення

**Обсяг кваліфікаційної роботи:**

Кількість листів креслень 2; кількість сторінок записки 72.
1.Короткий зміст пояснювальної записки та прийнятих рішень У кваліфікаційній роботі розглянуто створення вебсистеми для організації та контролю виконання проєктних завдань. Проведено аналіз предметної області, сформульовано вимоги, обґрунтовано архітектурні та технологічні рішення. Реалізовано функціональний застосунок із можливістю збереження, фільтрації та експорту інформації про подорожі. Проведено тестування, що підтвердило коректну роботу програмного забезпечення.

2. Висновок про відповідність роботи поставленому завданню Кваліфікаційна робота виконана відповідно до поставленого завдання та з дотриманням всіх вимог.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки та передових методів роботи У вступі обґрунтовано актуальність теми, сформульовано мету, завдання, об’єкт і предмет дослідження. У першому розділі проведено змістовний аналіз предметної області. Проаналізовано сучасні сервіси, їх функціонал, переваги та недоліки. Виділено ключові проблеми, які вирішує розроблювана вебсистема, сформульовано функціональні й нефункціональні вимоги. У другому розділі спроєктовано архітектуру вебсистема на основі шаблону MVC та клієнт-серверної моделі. Розроблено структуру бази даних, повну ER-діаграму, структуру взаємодії між компонентами. Обґрунтовано вибір стеку технологій MERN. Створено інтерфейс користувача з урахуванням зручності використання. У третьому розділі реалізовано серверну і клієнтську частини, створено базу даних, функціональні модулі для створення завдань, редагування, розподілення відповідальності. Проведено тестування вебсистеми та проаналізовано результати, що підтверджують відповідність функціоналу поставленим вимогам.

4. Позитивні сторони роботи Актуальність обраної теми полягає у потребі зручного цифрового інструменту для організації і контролю виконання проєктних завдань. Робота відзначається глибоким аналізом аналогів, якісним проєктуванням, застосуванням сучасних технологій (MongoDB, Express.js, React, Node.js, JWT), а також орієнтацією на зручність користувача.

5. Негативні сторони роботи Обмежена реалізація фільтрації завдань – доцільно було б

6. Оцінка графічного оформлення та пояснювальної записки Графічне оформлення відповідає тематиці роботи та містить діаграми архітектури, послідовностей, компонентів тощо. Пояснювальна записка оформлена згідно вимог і має логічну структуру викладення матеріалу.

7. Відгук про кваліфікаційну роботу в цілому Кваліфікаційна робота є завершеною, цілісною та технічно грамотною. У пояснювальній записці видно послідовність дій — від аналізу задачі до її практичного втілення. Сучасні технології та обрані архітектурні рішення дали змогу створити працездатний сервіс із достатнім рівнем зручності. Матеріал структурований, а наведені ілюстрації доречно доповнюють основний текст.

8. Інші зауваження

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

РЕЦЕНЗЕНТ Чешун Віктор Миколайович
канд. техн. наук, доц.
доцент кафедри кібербезпеки

"04" червня 2025 р. 
(підпис)

# РІШЕННЯ ЕКСПЕРТНОЇ КОМІСІЇ КАФЕДРИ
# ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Назва кваліфікаційної роботи Вебсистема для організації та контролю виконання проєктних завдань

Автор Мазур Дмитрій Ігорович

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Рівень вищої освіти Перший (бакалаврський)

Спеціальність 121 «Інженерія програмного забезпечення»

Науковий керівник Форкун Юрій Вікторович, канд. техн. наук, доцент

На основі аналізу кваліфікаційної роботи на дотримання вимог академічної доброчесності (у т.ч. відсутності ознак академічного плагіату) з урахуванням результатів перевірки роботи спеціалізованим програмним засобом(ами) комісія зробила такий висновок:

| № | Висновок | Позначка про відповідність |
|---|---|---|
| 1 | Ознаки академічного плагіату | |
| 1.1 | Запозичення, виявлені в роботі, є законними і не є академічним плагіатом (далі – зазначаються підстави віднесення запозичень до правомірних, якщо потрібно). Робота приймається до захисту. | **відповідає** |
| 1.2 | Виявлені запозичення не є академічним плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи (далі – зазначаються детальні та аргументовані підстави віднесення запозичень до правомірних). Робота приймається до захисту, але має бути відкоригована. | |
| 1.3 | Виявлені запозичення не є академічним плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота може бути допущена до захисту після того як буде відкоригована та доопрацьована і успішно пройде повторну перевірку на академічний плагіат. | |
| 1.4 | Робота містить навмисні текстові спотворення, передбачувані спроби укриття текстових запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту. | |
| 2 | Інші види порушень академічної доброчесності | |

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) у тексті кваліфікаційної роботи системами перевірки на плагіат виявлено схожість з деякими документами в частині загальновживаних обов'язкових словосполучень у стандартних бланках (титулка, завдання, анотація, відомість документів), у структурі змісту, назвах розділів/підрозділів тощо, у назвах публікацій та у переліку джерел посилання;

2) в якості запозичень системою було зафіксовано деякі послідовності вихідного коду і посилання на бібліотеки, які є стандартними мовними конструкціями програмування та не можуть розглядатися як об'єкт авторських прав і, відповідно, їх порушення;

3) усі запозичення є фрагментарними або мають належним чином оформлені посилання;

4) виявлені модифікації тексту не впливають на відсоток схожості.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів ідентичності/схожості, складає 3,8%, що, з урахуванням наведених обґрунтувань, відповідає характеру теми і свідчить на користь кваліфікаційної роботи.

Дата

Завідувач кафедри _____ Підпис Леонід БЕДРАТЮК Ім'я ПРІЗВИЩЕ

Гарант освітньої програми _____ Підпис Леонід БЕДРАТЮК Ім'я ПРІЗВИЩЕ

Керівник кваліфікаційної роботи _____ Підпис Юрій ФОРКУН Ім'я ПРІЗВИЩЕ