|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Хмельницький національний університет  Факультет інформаційних технологій  Кафедра комп'ютерних наук  **КУРСОВИЙ ПРОЄКТ**  з дисципліни «WEB-технології та WEB-дизайн» | | | | | | | |
| на тему | | *Вебсайт системи управління завданнями* | | | | | | | |
| Галузь знань | | | *12 – Інформаційні технології* | | | | |
|  | | | Шифр і назва галузі знань | | | | |
| Спеціальність | | | *122 – Комп’ютерні науки* | | | | |
|  | | | Шифр і назва спеціальності | | | | |
| Освітня програма | | | *Комп’ютерні науки* | | | | |
|  | | | Назва освітньої програми | | | | |
|  | | | | | | | |
| Виконав: | *студент групи КН-22-2* | | |  |  |  | *Максим САЄЦЬ* |
| Курс, група виконавця | | | |  | Підпис |  | Ім’я, прізвище |
| Керівник: | *к.т.н., доц. каф. КН* | | |  |  |  | *Руслан БАГРІЙ* |
| Науковий ступінь, посада | | | |  | Підпис |  | Ім’я, прізвище |
| Нормоконтроль: | | *к.т.н., доц. каф. КН* | |  |  |  | *Руслан БАГРІЙ* |
| Науковий ступінь, посада | | | |  | Підпис |  | Ім’я, прізвище |
| для | | | |  |  |  |  |
| До захисту допускаю: | | | |  |  |  |  |
| *Зав. кафедри КН, д.т.н., професор* | | | |  |  |  | *Олександр БАРМАК* |
| *29 травень* 2025 р. | | | |  | Підпис |  | Ім’я, прізвище |
| Хмельницький 2025 | | | | | | | |

**Анотація**

Тема курсового проєкту: *«Вебсайт системи управління завданнями»*

Виконавець курсового проєкту: *студент групи КН-22-2 Максим САЄЦЬ*

Керівник курсового проєкту: *к.т.н., доцент кафедри КН Руслан БАГРІЙ*

Курсовий проєкт містить:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Пояснювальна записка | | | | Кількість додатків |
| Сторінок | Рисунків | Таблиць | Джерел інформації |
| 47 | 21 | 9 | 18 | 5 |

Метою курсового проєкту є розробка вебплатформи системи управління проєктами для ефективної командної взаємодії та контролю виконання завдань. Для розробки використано мову програмування JavaScript, фреймворк React JS для клієнтської частини, середовище виконання Node.js з Express для серверної логіки, базу даних PostgreSQL та сервіс Firebase Authentication для безпечної автентифікації користувачів.

Результатом роботи є вебсайт, що дозволяє користувачам реєструватися та входити в систему, створювати, редагувати й видаляти проєкти та завдання, призначати відповідальних, встановлювати терміни, відстежувати статуси в реальному часі, залишати коментарі та генерувати звіти про прогрес виконання.

Ключові слова: система управління, вебплатформа, завдання, проєкти, Node.js, React JS, Firebase.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Виконавець: | *студент 3 курсу, група КН-22-2* |  |  |  | *Максим САЄЦЬ* |
| Курс, група виконавця | |  | Підпис |  | Ім’я, прізвище |

**Зміст**

[Перелік скорочень 3](#_Toc577661342)

[Вступ 4](#_Toc784150161)

[Розділ 1 5](#_Toc196460395)

[Характеристика предметної області та постановка задачі 6](#_Toc598629514)

[1.1 Аналіз предметної області 6](#_Toc61603514)

[1.2 Аналіз існуючого програмного забезпечення предметної області 9](#_Toc269957712)

[1.3 Мета, задачі та вимоги до реалізації інформаційної системи 13](#_Toc1703550322)

[Розділ 2 14](#_Toc2036000646)

[Проєктування інформаційної системи 15](#_Toc1080797515)

[2.1 Функціональна структура інформаційної системи 15](#_Toc1769858860)

[2.2 Інформаційна структура системи 17](#_Toc1683393521)

[2.2.1 Проектування веб-інтерфейсу інформаційної системи 17](#_Toc1901351247)

[2.2.2 Інформаційна модель 21](#_Toc1165850495)

[2.3 Веб-технології та засоби розробки інформаційної системи 25](#_Toc1229824123)

[Розділ 3 27](#_Toc1148479288)

[Програмна реалізація інформаційної системи 28](#_Toc1167422900)

[3.1 Структура та функціональне призначення складових системи 28](#_Toc1896923625)

[3.2 Особливості реалізації програмних складових системи 30](#_Toc871881315)

[3.3 Тестування інформаційної системи 34](#_Toc1747481970)

[3.4 Інструкція користувача 38](#_Toc330272533)

[3.5 Вимоги до розгортання інформаційної системи 44](#_Toc1981050342)

[Висновки 45](#_Toc1855011704)

[Перелік посилань 46](#_Toc125985182)

Додатки

Перелік скорочень

|  |  |
| --- | --- |
| **Скорочення, термін, позначення** | **Пояснення** |
| БД | База даних |
| SQL | Structured Query Language |
| URL | Uniform Resource Locator |
| CI/CD | Continuous Integration / Continuous Deployment |
| UUID | Universally Unique Identifier |
| HTML | Hyper Text Markup Language |
| MVC | Model View Controller |
| ORM | Object-Relation Mapping |
| SPA | Single Page ApplicationC |
| CLI | Command Line Interface |

Вступ

Хороше управління проєктами є ключовим чинником успіху будь-якої організації, оскільки дозволяє оптимізувати ресурси, дотримуватися встановлених термінів та забезпечувати високу якість результату. У сучасних умовах цифрової трансформації та розповсюдження віддаленої роботи потреба в прозорих і адаптивних інструментах управління проєктами зросла в рази, адже понад 70 % IT-проектів відчувають затримки через недостатню комунікацію та неузгодженість процесів [1].

Системи управління проєктами класифікують за методологічними підходами (Waterfall, Agile, Scrum), за галузевою специфікою (ІТ, будівництво, маркетинг) та за функціональними можливостями (планування ресурсів, трекінг завдань, звітування, аналіз ризиків). Наприклад, рішення на базі Agile-методології сприяють швидкому реагуванню на зміни вимог, а системи з розвиненим модулем звітності автоматизують формування KPI і дозволяють керівникам приймати рішення на підставі актуальних даних.

Попри широкий вибір готових рішень (Jira, Asana, Trello тощо), багато компаній стикаються з необхідністю кастомізації інтеграція з внутрішніми сервісами, власні правила бізнес-логіки й специфічні шаблони процесів. Саме це обумовлює потребу в розробці гнучкої веб-платформи, яка поєднуватиме простий інтерфейс для користувачів і відкриту архітектуру для швидкого масштабування та розширення функцій [2].

Планується розробити вебплатформу системи управління проєктами, що об’єднує інтуїтивні інструменти планування, призначення завдань, моніторингу статусів та формування звітів. Особливу увагу приділено адаптивності інтерфейсу, користувачі можуть налаштовувати робочі області під свій стиль роботи. Таким чином, створена вебплатформа має поєднує сучасні методи розробки та продумані бізнес-процеси, щоб забезпечити організаціям підвищення ефективності командної роботи та своєчасне досягнення цілей.

Розділ 1

Характеристика предметної області та постановка задачі

1.1 Аналіз предметної області

У повсякденному житті будь-яка група людей стикається з необхідністю спільно виконувати різноманітні завдання для планування зустріч, розподілу обов’язків у різних сферах, стеження за термінами виконання та обмінювання інформацією. Навіть невелика команда без злагодженої системи управління швидко ризикує зіткнутися з непорозуміннями, пропущеними дедлайнами та дублюванням роботи.

Сучасні цифрові інструменти дозволяють уникнути цих проблем, об’єднавши весь процес роботи над проєктами в єдиному веб-інтерфейсі. Багато компаній і команд уже використовують готові онлайн-сервіси, які пропонують можливості створення завдань, відстеження їхнього стану та спілкування в рамках проєкту. Проте більшість таких рішень виявляються надто складними для швидкого запуску або, навпаки, занадто примітивними й не пристосованими до реальних робочих процесів організації. На практиці користувачі вимушені підлаштовувати свою систему під інтерфейс і логіку сервісу, що забирає час і знижує ефективність.

Саме тому виникає потреба в легкій, швидкій у розгортанні платформі, яка поєднуватиме простоту користування з гнучкістю налаштувань. Така система має враховувати різний досвід команди, дозволяти швидко створювати проєкти та завдання, чітко відображати статуси робіт і надавати зручний спосіб для обговорення деталей. Крім того, надзвичайно важливо, щоб інтерфейс залишався зрозумілим навіть для тих, хто не має спеціальної підготовки в питаннях управління проєктами.

У будь-якій організації ефективне управління проєктами є запорукою своєчасного досягнення цілей, раціонального використання ресурсів і підтримки високої якості результату.

Проєктний цикл в організації охоплює низку послідовних етапів, спочатку відбувається ініціація проєкту з формулюванням чіткого бачення та мети, ідентифікацією зацікавлених сторін і визначенням необхідних ресурсів, потім у фазі планування створюється детальна структура робіт із розбивкою на завдання, оцінкою трудовитрат, встановленням дедлайнів і призначенням відповідальних виконавців, далі на етапі виконання команда реалізує заплановані роботи, обмінюється повідомленнями в межах проєкту та фіксує поточний статус завдань, після чого у процесі моніторингу й контролю відстежується реальний прогрес, здійснюється управління ризиками та за потреби коригується початковий план, і на етапі завершення готується підсумковий звіт та оцінюється досягнуті результати [3].

У сучасних веб-застосунках для управління такими проєктами передбачені різні ролі та права доступу, менеджер проєкту створює й налаштовує нові проєкти, розподіляє завдання та формує звіти, учасник команди отримує призначені завдання, оновлює їхній статус і залишає коментарі, а клієнт має можливість переглядати хід виконання, затверджувати ключові етапи та приймати фінальні звіти.

Основні сценарії взаємодії користувачів із вебплатформою охоплюють кілька узагальнених етапів. Спочатку відбувається реєстрація та перевірка прав доступу через захищений механізм аутентифікації, що дає змогу кожному учаснику отримати індивідуальний інтерфейс і відповідні привілеї. Далі користувачі задають рамки проєктної діяльності, формуючи структуру робіт і визначаючи строки їх виконання, після чого система надає інструменти для створення, перегляду й оновлення завдань із можливістю змінювати їхній статус і сортувати за ключовими критеріями. Також появляється можливість підтримувати обмін повідомленнями та ведення дискусій у контексті конкретних задач і тем, сприяючи оперативному вирішенню питань і координації дій. Наприкінці кожного робочого циклу платформа генерує зведені звіти на основі накопичених даних про виконані завдання та затрачений час, дозволяючи оцінити загальну ефективність і виявити основні тенденції в роботі команди [4]. На основі вказаних процесів сформовано інформаційну модель предметної області (табл. 1.1), що відображає основні сутності системи та їхні атрибути. (таблиця 1.1).

Таблиця 1.1 – Інформаційна модель предметної області вебсайту системи управління завданнями

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Сутність** | **Опис** | **Атрибути** |
| 1 | Користувач | Зареєстрована особа, що використовує систему: менеджер, учасник або клієнт | Ім’я, email, пароль, роль, avatar, дата реєстрації |
| 2 | Проєкт | Сукупність завдань та ресурсів, об’єкт управління | Назва, опис, власник, дата створення, дедлайн |
| 3 | Завдання | Одиниця роботи в межах проєкту | Назва, опис, відповідальний, статус, пріоритет, дедлайн |
| 4 | Коментар | Повідомлення користувача стосовно певного завдання | Автор, текст, дата створення |
| 5 | Звіт | Структуровані дані про виконання проєкту за обраний період | Назва, період\_від, період\_до, згенеровано\_дата |
| 6 | Ресурс | Людський чи матеріальний ресурс, задіяний у виконанні завдань | Назва, тип (людський/технічний), доступність |
| 7 | Роль | Права доступу користувача до функціоналу системи | Назва (менеджер/учасник/клієнт), опис праві |

Модель демонструє взаємодію основних елементів системи управління завданнями та формує цілісну картину структури даних. Користувачі ініціюють команди у рамках яких створюються завдання й виділяються необхідні ресурси, до кожного завдання може додаватися обговорення у вигляді коментарів, а підсумкові результати фіксуються у звітах.

Отже інформаційна модель закладає міцний фундамент для ефективної роботи всієї системи, а чітка структура даних гарантує однозначність зберігання й обробки інформації. Завдяки такому підходу розробка та підтримка платформи відбуваються швидше, впровадження нових функцій і масштабування виконуються з мінімальними зусиллями, що в кінцевому результаті сприяє стабільності, гнучкості та довготривалому розвитку вебсайту системи управління завданнями.

1.2 Аналіз існуючого програмного забезпечення предметної області

На ринку існує безліч сервісів для організації спільної роботи над проєктами, від простих інструментів для невеликих команд до потужних платформ для корпоративного рівня. При виборі рішення важливо враховувати не лише набір функцій, але й те, наскільки легко співробітникам звикнути до інтерфейсу та робочої логіки, а також вартість впровадження й подальшої підтримки.

Нижче наведено аналіз трьох провідних платформ, а саме «Trello», «Asana» і «Jira». Trello побудовано на інтуїтивній канбан-дошці, де дошки виступають основними робочими просторами, списки відображають статуси або категорії задач, а картки вміщують конкретні завдання. Користувачі можуть створювати картки, перетягувати їх між списками для відображення змін статусу, використовувати кольорові позначки для відокремлення пріоритетів або типів робіт і задавати терміни виконання, що дозволяє зберігати візуальну легкість і швидко реагувати на зміни.

Мінімалістичний інтерфейс Trello є водночас його силою, адже простота сприяє швидкому запуску роботи й легкому залученню нових учасників, проте може обмежувати потреби у складнішій аналітиці, розгалужених робочих процесах і деталізованих звітах [5]. Головну сторінку показано на рисунку 1.1.

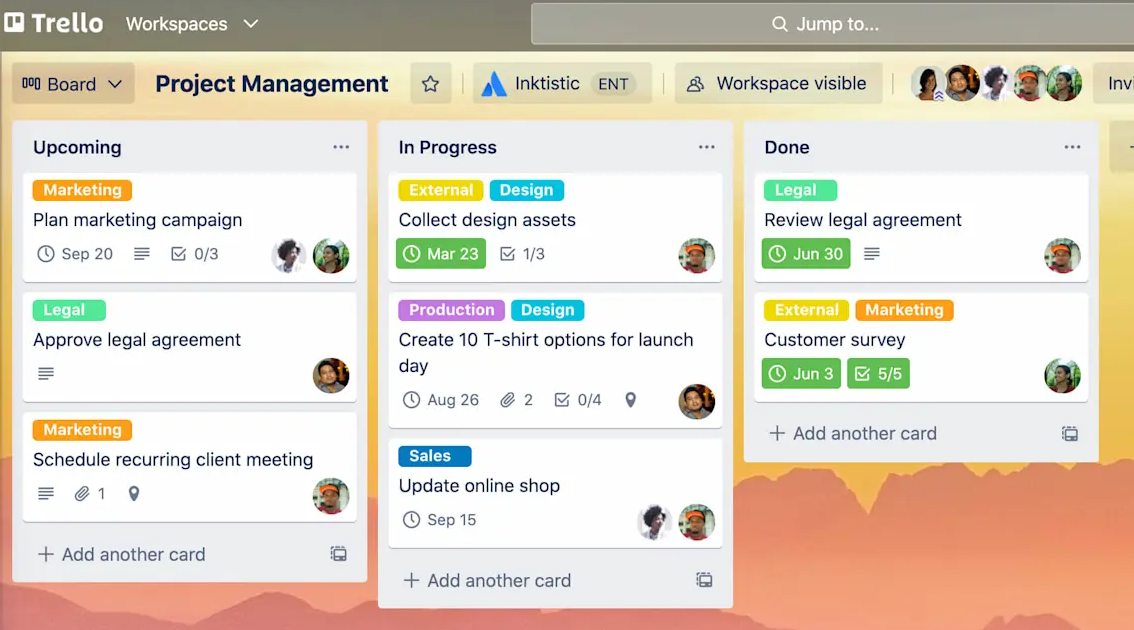


Рисунок 1.1 – Головна сторінка сайту «Trello» [6]

Система має наступні переваги:

* швидкий старт із мінімальними налаштуваннями і відсутністю громіздких меню дає змогу розпочати роботу ліченими хвилинами;
* гнучкі розширення Power-Ups забезпечують інтеграцію з календарем, месенджерами й хмарними сервісами, а також відкривають можливості для власної автоматизації;
* мобільних додатків для iOS і Android гарантує доступ до завдань з будь-якого пристрою й підтримує синхронізацію в реальному часі.

Недоліками системи є:

* в безкоштовному тарифі доступні лише базові табличні звіти без вбудованої аналітики й графіків;
* відсутність формальних залежностей між завданнями й обмеження на кількість Power-Ups та розмір вкладених файлів.

Наступним прикладом є сайт «Asana», що поєднує кілька режимів перегляду проєкту, такі як список, канбан-дошку, календар та хронологію і пропонує готові шаблони для найрізноманітніших бізнес-процесів, які можна використовувати одразу після реєстрації. Система дозволяє переключатися між різними видами відображення одним кліком, що забезпечує оптимальну картину як для детальної роботи з окремими завданнями, так і для стратегічного планування масштабних ініціатив, а можливість налаштувати автоматичні правила дає змогу, переміщати завдання при зміні статусу, додаванні підзадач або досягненні певного дедлайну без залучення людини.

Asana підтримує глибоку інтеграцію з популярними інструментами Google Workspace, Microsoft Teams, Salesforce, Zoom та GitHub що робить її зручною для корпоративного використання та побудови єдиного робочого простору (рисунок 1.2).

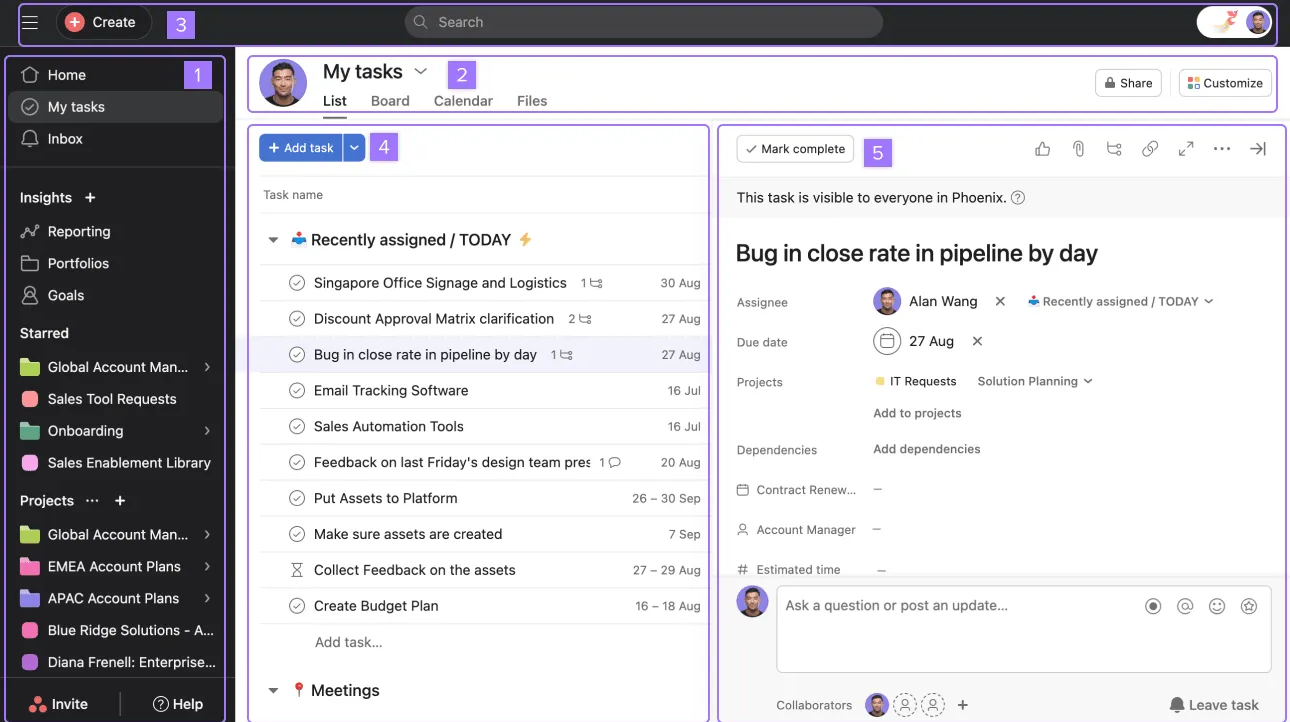


Рисунок 1.2 – Інтерфейс «Asana» [7]

Asana забезпечує гнучке налаштування полів задач через додавання власних атрибутів, що дає змогу адаптувати систему під специфічні потреби команди та краще структурувати інформацію, а функція глобального пошуку і фільтрації дозволяє миттєво знаходити потрібні елементи без втрати часу на ручний перегляд. У мобільному застосунку підтримується офлайн режим що дає змогу працювати з завданнями навіть за відсутності інтернет-з’єднання і синхронізувати всі зміни після відновлення доступу.

Проте можливості кастомізації полів та автоматизації вимагають доступу до платних планів і можуть бути недоступні в безкоштовному тарифі, а складність налаштувань іноді потребує залучення IT-спеціалістів або довшого часу на навчання. Під час роботи з великими обсягами даних іноді можна помітити уповільнення інтерфейсу навіть за хорошого інтернет-з’єднання що може вплинути на продуктивність користувачів. Також інтеграції з менш поширеними сервісами доводиться налаштовувати вручну через API що може стати зайвим витратним етапом на старті проєкту.

Іншим прикладом є i «Jira», що представлена як комплексна платформа для управління складними процесами та великими командами в ІТ середовищі з можливістю адаптації до різних методологій від Scrum до Kanban і гібридних схем роботи. Система забезпечує центральну координацію завдань через настроювані робочі процеси, де кожна транзиція задачі може супроводжуватися визначеними правилами, ескалаціями та підтвердженнями. Аналітичні інструменти та звітність відображають поточний стан проектів у динаміці завдяки burndown, control chart, velocity і cumulative flow діаграмам. Глибока інтеграція з іншими продуктами екосистеми допомагає централізувати документацію, контроль версій та CI/CD під єдиним дахом. Масштабованість платформи забезпечується численними плагінами та API, що дозволяють доповнювати ядро системи новими функціями без змін базового коду (рисунок 1.3).

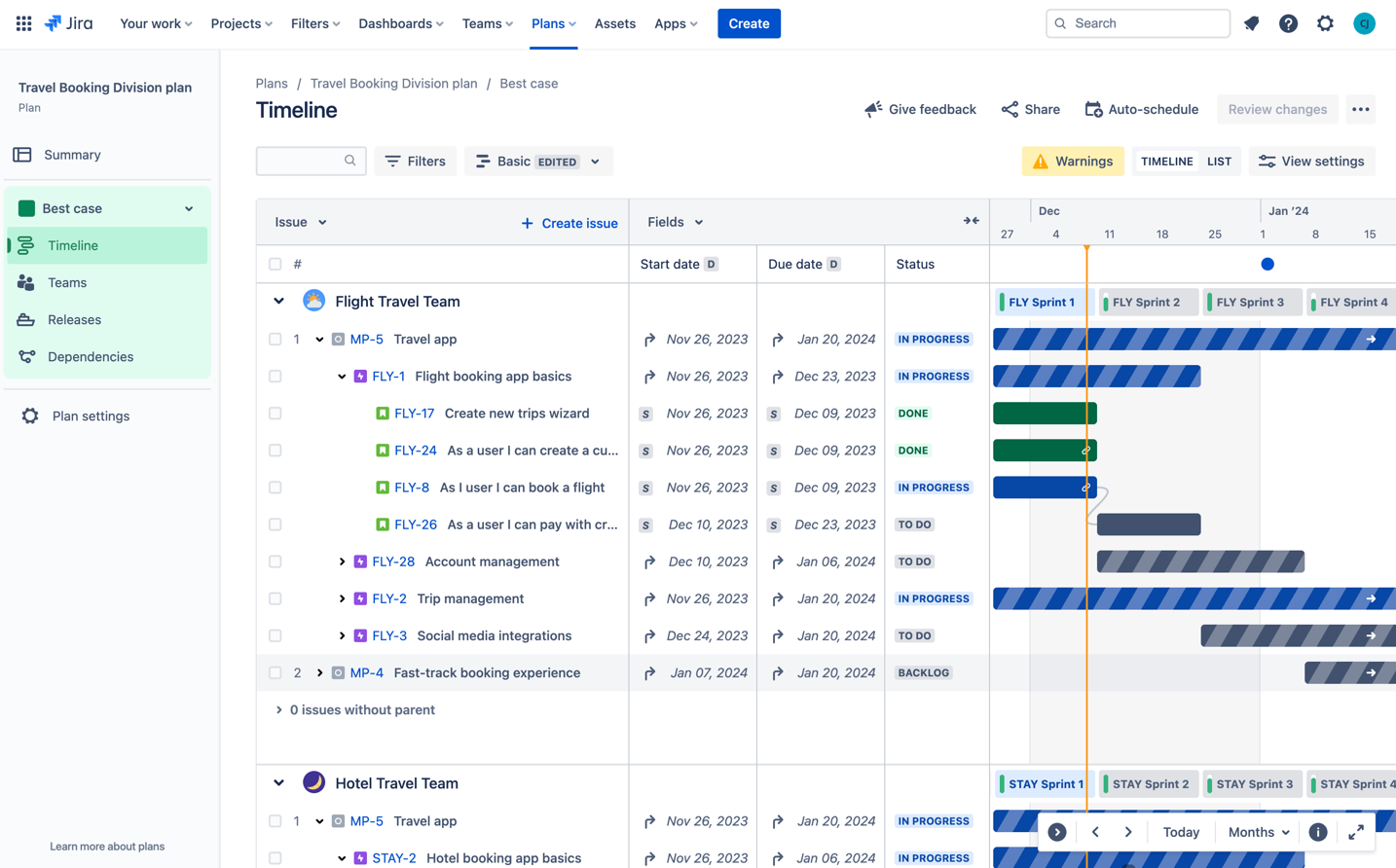


Рисунок 1.3 – Головне вікно системи «Jira» [8]

Серед сильних сторін системи є гнучке налаштування робочих процесів з автоматичними тригерами що дозволяє адаптувати систему під різні підходи, діаграми що спрощують моніторинг проекту, а також інтеграція з Confluence, Bitbucket і Bamboo забезпечує єдине середовище для кодування, документації й CI/CD. Проте проблемами є надмірна складність у початковому налаштуванні що вимагає адміністратора з досвідом та окремих ролей. Також інтерфейс перевантажений функціоналом, більша частина якого не використовується в невеликих командах, а висока вартість ліцензій навіть для обмеженої кількості користувачів може виявитися бар’єром для малого бізнесу.

Огляд найпопулярніших систем управління проєктами демонструє, що кожна з них пропонує власний набір рішень для командної роботи. Проте жодна не забезпечує одночасно зручність використання, глибину функціональності та прозорість аналітики без надмірної складності чи фінансового навантаження. Проведений аналіз виявив ключовий запит користувачів на інструмент, який поєднує простоту та налаштовуваність, дозволяє швидко візуалізувати прогрес у вигляді канбан-дошки, таблиць чи календаря, підтримує гнучке планування завдань, автоматизацію рутинних дій, а також надає аналітику у вигляді зручних звітів, придатних до експорту.

Отже, перспективи розвитку систем управління проєктами полягають у створенні універсальних рішень, що поєднують інтуїтивно зрозумілий інтерфейс із потужними інструментами для аналізу, гнучкого планування та автоматизації. Зростаючий попит на ефективні, адаптивні платформи відкриває нові можливості для розробників у напрямку вдосконалення користувацького досвіду, підвищення продуктивності команд і створення рішень, які максимально точно відповідають сучасним викликам цифрового середовища.

1.3 Мета, задачі та вимоги до реалізації інформаційної системи

Метою курсового проєкту є створення веб-платформи для управління проєктами, яка забезпечить прозору взаємодію в команді під час планування, виконання та моніторингу завдань. Платформа має підтримувати зручну авторизацію, а також функціонал для керування проєктами, завданнями, коментарями та звітами.

Основні функції включають:

* реалізація реєстрації та входу;
* розробка архітектури з урахуванням потреб і функцій зареєстрованих користувачів;
* створення модулів для керування командами й завданнями;
* впровадження коментування та аналітики;
* створення зручного інтерфейсу для редагування завдань та організації комунікації щодо них;
* побудова організованої структури для зберігання даних про проєкти, завдання та користувачів;
* реалізація адаптивного інтерфейсу для мобільних і десктопних пристроїв;
* тестування ключових функціональних компонентів системи.

Поставлені завдання дозволять створити зручну та ефективну веб-платформу для управління завданнями, яка забезпечить прозору комунікацію і контроль над виконанням завдань у команді. Платформа сприятиме підвищенню продуктивності, організованості та якості роботи за рахунок інтеграції необхідних інструментів керування.

# 

# Розділ 2

# Проєктування інформаційної системи

2.1 Функціональна структура інформаційної системи

Розробка функціональної структури інформаційної системи керування завданнями вимагає ретельного продумування кожного модуля та взаємозв’язків між ними задля забезпечення гнучкості, масштабованості й надійності рішення. Лише чітко спланована архітектура дозволяє точно визначити взаємодію між компонентами, узгодити бізнес-процеси та мінімізувати ризики помилок уже на стадії розробки. У випадку вебсайту системи управління завданнями основна увага приділяється модулю створення й обробки завдань, який дає змогу користувачу миттєво створювати запис із назвою, описом і дедлайном, призначати відповідальну особу, змінювати статус виконання та додавати коментарі і файли. Інтерфейс цього модуля передбачає інтуїтивно зрозумілі форми для введення даних і автоматичне збереження змін, що суттєво скорочує час взаємодії та мінімізує можливість помилок при постановці завдань. Перевагою такого підходу є те, що користувачі можуть оперативно вносити правки, залишати коментарі й додавати вкладення без переходу в інші розділи системи, що підвищує ефективність командної роботи.

Не менш важливою складовою є функціонал пошуку та фільтрації, що дає змогу відбирати завдання за датою створення, терміном виконання, пріоритетом чи відповідальним виконавцем без перезавантаження сторінки, підтримуючи високу швидкість відгуку навіть при великій кількості записів у базі. Для відстеження часових рамок і координування роботи варто створити модуль календаря, який автоматично відображає дедлайни по всіх активних завданнях та проектам. Такий модуль підвищує прозорість процесів та знижує ризик затримок при реалізації складних проектів.

Інтерфейс керування профілем користувача включає можливості редагування персональних даних, зміну аватару й керування налаштуваннями сповіщень про зміни статусу завдань або нові коментарі. Такий рівень персоналізації дозволяє кожному учаснику системи отримувати актуальну інформацію у зручному форматі та підтримувати високу залученість до процесу.

На основі функціональної структури та потреб було створено діаграму варіантів для зареєстрованих користувачів системи керування завданнями [9]. Що об’єднує в собі логічно пов’язані модулі для створення й обробки завдань, можливості пошуку й фільтрації (рисунок 2.1).

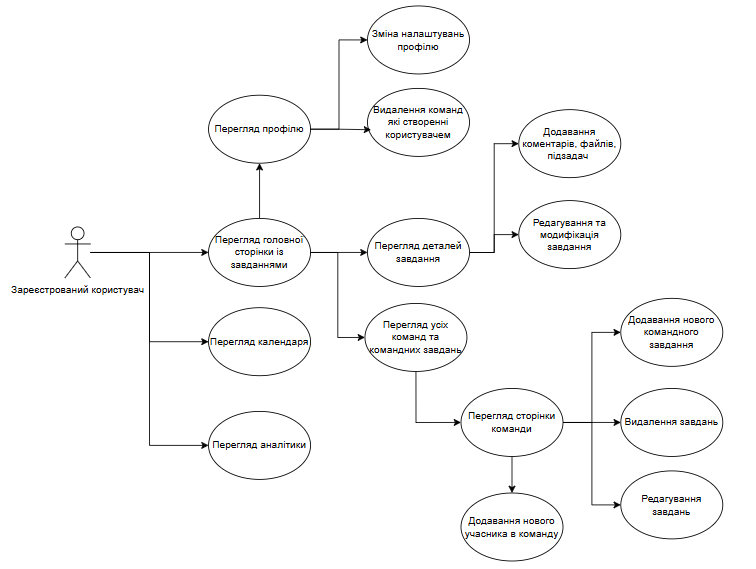


Рисунок 2.1 – Діаграма варіантів використання вебсайта системи для управління завданнями для зареєстрованих користувачів

У розробленій діаграмі варіантів використання для зареєстрованого користувача відображено основні сценарії роботи з системою через простий інтерфейс. Спочатку користувач потрапляє на головну сторінку із завданнями, де може створити нову задачу, відфільтрувати їх за пріоритетом, датою чи виконавцем і переглянути зміни. При переході до деталей завдання доступні функції додавання коментарів, вкладення файлів і створення підзадач, а також змінення статусу виконання. Розділ із переглядом усіх команд та командних завдань веде на сторінку конкретної групи, де без ролей адміністратора можна додавати нових учасників, створювати або редагувати задачі й видаляти зайві записи. Управління профілем реалізовано через можливість зміни персональних даних, оновлення аватару, видалення власних команд.

Отже аналіз бізнес-процесів і групова класифікація функцій у діаграмі варіантів використання забезпечують цілісне бачення архітектури системи управління завданнями та створюють надійну основу для розробки системи.

2.2 Інформаційна структура системи

2.2.1 Проектування веб-інтерфейсу інформаційної системи

Проєктування інформаційної структури є невід’ємною складовою успішної розробки будь-якої вебсистеми адже саме від узгодженого поділу даних і логіки взаємодії залежить швидкість роботи користувачів на платформі та надійність функціоналу. Без чітко окреслених блоків інтерфейсу і продуманого розташування контенту зростає ризик плутанини, помилок під час виконання завдань і зниження продуктивності команди.

Вебсайт системи управління завданнями має забезпечувати одночасно простоту навігації й гнучкість для розширення нових можливостей у майбутньому. Тому на етапі проєктування важливо визначати не тільки структуру сторінок і основні шаблони відображення даних, але й взаємозв’язки між модулями що дозволяє своєчасно виявляти потенційні проблеми й оптимізувати подальшу розробку [10]. Добре спланована інформаційна структура гарантує логічну послідовність користувацьких сценаріїв, сприяє прозорому обміну даними між компонентами та створює стійку основу для масштабування і підтримки рішення. Головна сторінка вебсайту представлена на рисунку 2.2.

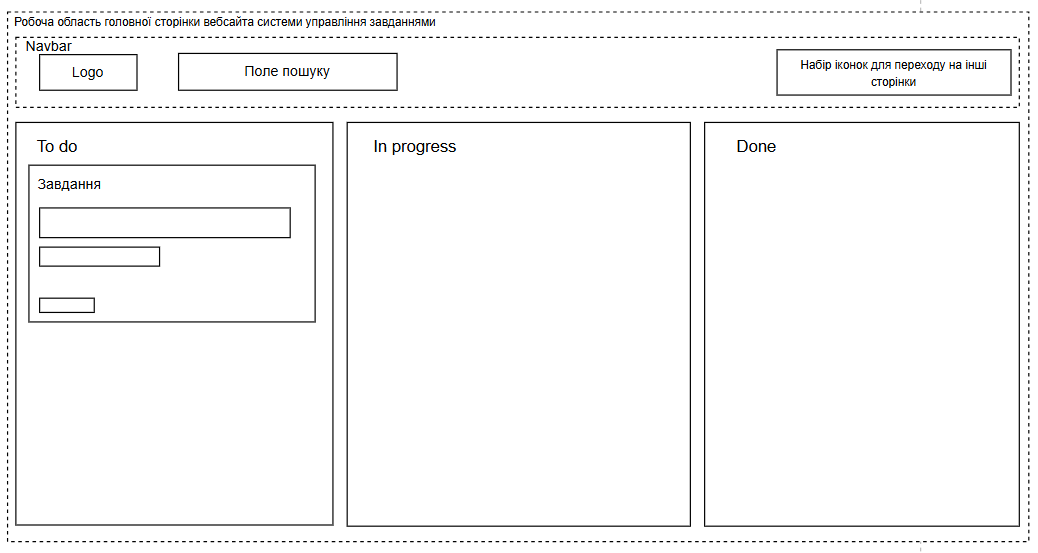


Рисунок 2.2 – Схема вебінтерфейсу головної сторінки вебсайту системи управління завданнями

З рисунку видно, що у верхній частині інтерфейсу розміщений фіксований блок навігації, в якому ліворуч знаходиться логотип, що служить швидким поверненням на головну, посередині поле для пошуку завдань за ключовими словами, а праворуч набір піктограм для переходу до календаря, статистики чи налаштувань. Така організація заголовка гарантує, що користувач завжди має під рукою інструменти для орієнтації в системі, навіть прокручуючи довгі списки завдань.

Основна робоча область розділена на три вертикальні колонки, які відповідають статусам завдань to do, in progress та done. У першій колонці відображаються нові завдання зі скороченим описом, терміном виконання та кнопкою для переходу до перегляду детальної інформації. У центральній колонці розташовані поточні завдання, а в правій завершені завдання. Користувач може перетягувати картки між колонками, що змінює статус завдань і дозволяє миттєво оновлювати їхній стан без зайвих кліків [11].

Спроєктована структура інтерфейсу головної сторінки дозволяє скоротити час навчання нових користувачів гарантувати швидкий доступ до ключових функцій і забезпечити чітку візуальну ієрархію яка мінімізує вірогідність помилок при роботі з великими обсягами завдань.

Також важливою частиною інтерфейсу системи є сторінка профілю користувача (рисунок 2.3).

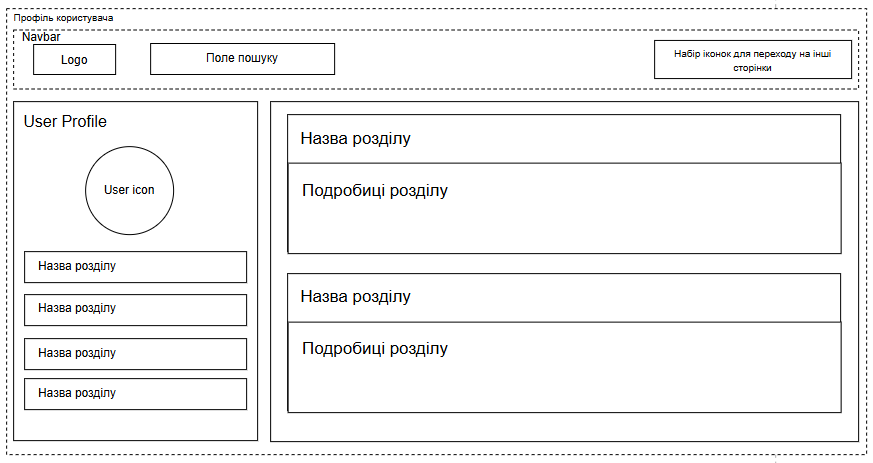


Рисунок 2.3 – Схема інтерфейсу сторінки профілю користувача

Сторінка профілю користувача вирізняється двокамерним макетом, в якому ліва панель відведена під персональні дані, а права під контент із деталями. Аватар та ім’я активного користувача розташовані у лівому верхньому куті, під якими знаходяться кнопки для перегляду різних розділів профілю списку власних завдань, налаштування облікового запису, повідомлення та інші модулі. Така компоновка дозволяє миттєво орієнтуватися у персональних даних та переходити до потрібного функціоналу в один клік.

У правій частині екрану розташовано блоки з назвами розділів та підрозділів для взаємодії та перегляду. Завдяки чіткій візуальній ієрархії користувач без зусиль перемикається між різними секціями свого профілю і доповнює або редагує дані.

Для перегляду подробиць завдання та можливості редагувати та доповнювати інформацією, створено схему інтерфейса вікна завдання при натиску на картку завдання (рисунок 2.4).

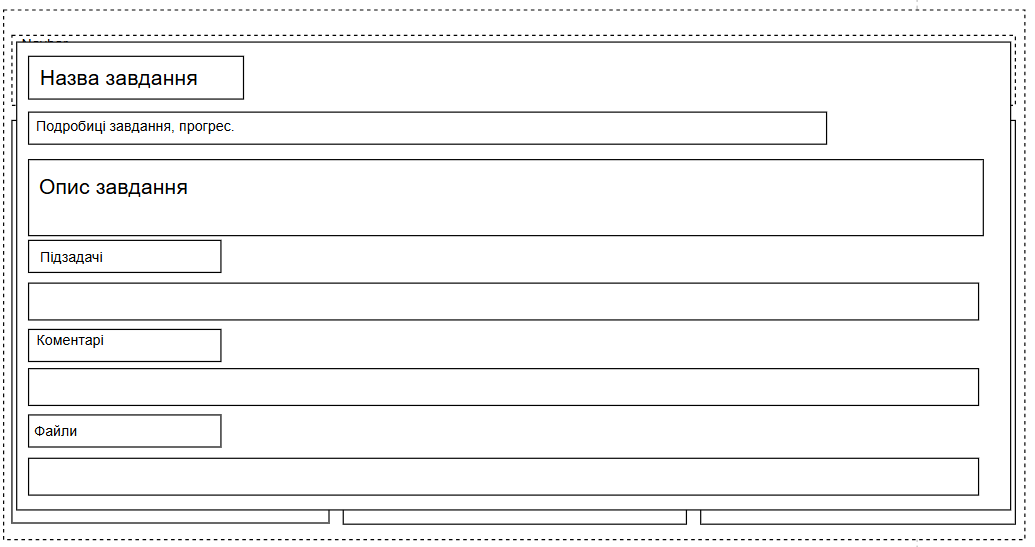


Рисунок 2.4 – Схема інтерфейсу вікна подробиць завдання

При натисканні на конкретне завдання відкривається детальна картка із зоною для назви, розгорнутим описом, зазначенням дати створення, терміну виконання, пріоритету і поля для коментарів та вкладень, а можливість додавання нотаток і файлів робить картку універсальним інструментом для командної співпраці. Кожен елемент інтерфейсу вебсистеми управління завданнями створений для зручності користувача та майбутнє розширення функціоналу. Чітко означені зони пошуку та навігації, продуманий поділ контенту на статусні колонки, інтуїтивне структурування профілю та детальної картки завдання забезпечують логічну послідовність користувацьких дій.

Завдяки такому підходу мінімізується час навчання нових учасників, підвищується швидкодія роботи з великими масивами завдань і створюється надійна база для масштабування платформи відповідно до нових вимог бізнесу. У підсумку ретельне проектування інформаційної архітектури вебсистеми управління завданнями дозволяє створити інтерфейс, у якому користувач швидко знаходить потрібні інструменти, легко перемикається між завданнями та отримує максимум функціональних можливостей без плутанини чи зайвих кроків.

2.2.2 Інформаційна модель

Надійне зберігання та обробка даних є фундаментальною умовою для будь-якої системи управління завданнями, оскільки від цього залежить не лише коректність виконання бізнес-логіки, але й загальна масштабованість і стійкість платформи під час інтенсивної взаємодії користувачів [12]. Тому в ході проєктування була виділена чітка схема, що описує сутності предметної області та їх взаємозв’язки, а потім сформована даталогічна модель структури бази даних (рисунок 2.5).

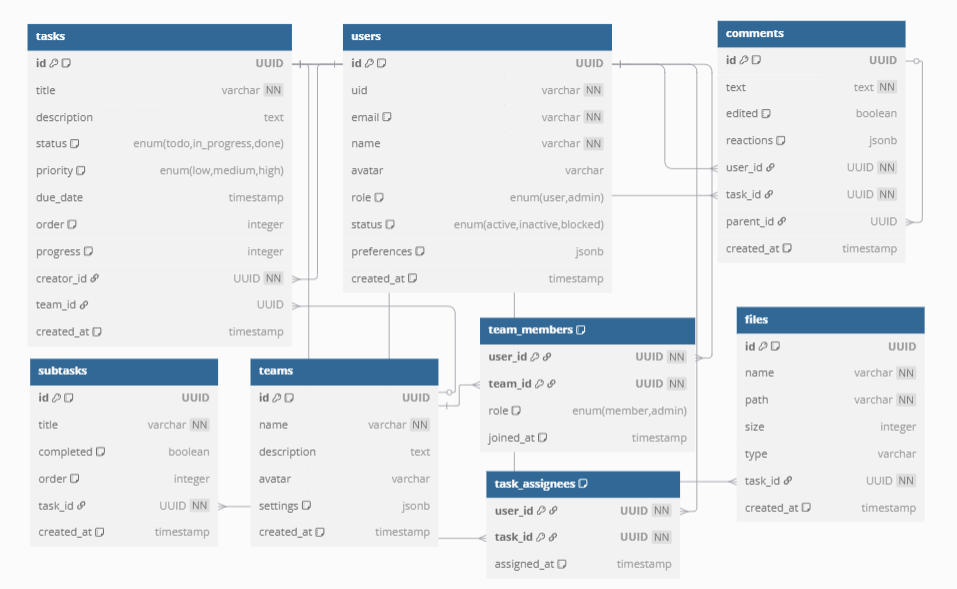


Рисунок 2.5 – Даталогічна модель БД вебсайту системи керування завданнями

Модель демонструє, як основні об’єкти такі як, облікові записи користувачів, командні об’єднання, завдання, прикріплені файли й коментарі інтегруються в єдину цілісність завдяки спроєктованим зовнішнім ключам. На рисунку видно, що таблиці Users та Tasks утворюють основу інформаційної моделі, маючи найбільшу кількість зв’язків з іншими сутностями й виконуючи роль центральних точок доступу до даних про користувачів і їхні завдання, тоді як таблиця Teams відображає групові об’єднання, Files зберігає метадані прикріплених ресурсів, а Comments відповідає за історію комунікації в межах завдань.

У таблиці 2.1 наведено атрибути таблиці Users, що відповідає за зберігання облікових записів платформи. Набір полів містить унікальний ідентифікатор ID, зовнішній ідентифікатор UID для інтеграції з Firebase Authentication, електронну адресу й ім’я, шлях до аватару, а також атрибути ролі та статусу, які регулюють рівень доступу та поточний стан облікового запису. А поле preferences у форматі jsonb дозволяє зберігати індивідуальні налаштування інтерфейсу, часові мітки created\_at і updated\_at фіксують момент створення та останнього оновлення профілю, що необхідно для аудиту та аналізу активності.

Таблиця 2.1 – Атрибути таблиці "Users"

|  |  |  |
| --- | --- | --- |
| **Назва** | **Тип даних** | **Опис** |
| id | UUID | Первинний ключ, унікальний ідентифікатор користувача |
| uid | varchar | Унікальний рядковий ідентифікатор для зовнішньої авторизації |
| email | varchar | Унікальна адреса електронної пошти |
| name | varchar | Назва облікового запису |
| avatar | varchar | Зображення користувача |
| role | enum | Роль користувача, що визначає рівень доступу |
| status | enum | Стан облікового запису |
| preferences | jsonb | Налаштування інтерфейсу |

В наступній таблиці описано інформацію про Tasks, тобто окремі одиниці роботи на платформі. Атрибути цієї сутності включають заголовок і текстовий опис, статус виконання, пріоритет, термін виконання due\_date, відсоток завершення progress і порядковий номер order для сортування. Зовнішні ключі creator\_id та team\_id встановлюють належність завдання до автора та проєкту, а created\_at і updated\_at дозволяють відстежувати хронологію змін [13].

Таблиця 2.2 – Атрибути таблиці "Tasks"

|  |  |  |
| --- | --- | --- |
| **Назва** | **Тип даних** | **Опис** |
| id | UUID | Первинний ключ, унікальний ідентифікатор завдання |
| title | varchar | Заголовок завдання |
| description | text | Детальний опис завдання |
| status | enum | Стан виконання завдання |
| priority | enum | Пріорітет виконання |
| due\_data | timestamp | Крайній термін завершення |
| progress | integer | Відсоток виконання |
| creator\_id | UUID | Зовнішній ключ, що посилається на таблицю users |
| team\_id | UUID | Зовнішній ключ, що посилається на таблицю teams |

Наступна таблиця описує структуру Teams, де зберігаються дані про організаційні об’єднання користувачів у робочі групи. Для кожного запису передбачено поля ідентифікатора, назви та докладного опису команди, а також jsonb-налаштування settings, які визначають правила запрошення нових учасників та обміну файлами. Зовнішній ключ owner\_id пов’язує команду з користувачем-власником, а мітка часу created\_at фіксує момент заснування групи (таблиця 2.3).

Таблиця 2.3 – Атрибути таблиці "Teams"

|  |  |  |
| --- | --- | --- |
| **Назва** | **Тип даних** | **Опис** |
| id | UUID | Первинний ключ, за яким ідентифікується команда |
| name | varchar | Назва команди |
| description | text | Опис команди, мети та сфери діяльності |
| settings | jsonb | Параметри керування доступом та обміном |
| created\_at | timestamp | Дата створення команди |

Таблиця Files приділяє увагу збереженню файлів, прикріплених до завдань, й у ній містяться метадані унікальний ID, оригінальна назва, шлях у файловому сховищі path, розмір у байтах size, а також зовнішній ключ task\_id, що пов’язує файл із конкретним завданням. Поле created\_at дає змогу впорядкувати документи за часом завантаження (таблиця 2.4).

Таблиця 2.4 – Атрибути таблиці "Files"

|  |  |  |
| --- | --- | --- |
| **Назва** | **Тип даних** | **Опис** |
| id | UUID | Первинний ключ, унікальний ідентифікатор файлу |
| name | varchar | Назва файлу |
| size | integer | Розмір файлу в байтах |
| type | varchar | Тип файлу |
| task\_id | UUID | Зовнішній ключ на tasks, до якого прикріплено файл |
| created\_at | timestamp | Час завантаження файлу |

Таблиця 2.5 орієнтована на організацію обговорень і зберігає коментарі до завдань. Кожен запис має власний ID, текст повідомлення text, логічний прапорець edited для позначення змінених коментарів і структуру reactions у форматі jsonb для накопичення реакцій користувачів. Зовнішні ключі user\_id, task\_id і parent\_id задають авторство, прив’язку до завдання й ієрархію відповідей відповідно, а мітка часу created\_at фіксує момент публікації коментаря.

Таблиця 2.5 – Атрибути таблиці "Comments"

|  |  |  |
| --- | --- | --- |
| **Назва** | **Тип даних** | **Опис** |
| id | UUID | Первинний ключ, унікальний ідентифікатор коментаря |
| text | text | Текст повідомлення |
| edited | boolean | Позначка про редагування |
| reactions | jsonb | Збір реакцій користувачів |
| user\_id | UUID | Зовнішній ключ на users, автор коментаря |
| task\_id | UUID | Зовнішній ключ на tasks до якого належить коментар |
| parent\_id | UUID | Зовнішній ключ на comments, для організації ланцюжків відповідей |
| created\_at | timestamp | Час створення |

Отже, даталогічна модель охоплює всі необхідні сутності й зв’язки для реалізації ключового функціоналу автентифікації та авторизації користувачів, створення та координації командної роботи, ведення завдань, обміну файлами і проведення дискусій. Що гарантує цілісність даних, ефективність запитів і готовність до подальшого розширення системи, побудована модель забезпечує не лише підтримку поточних процесів, а й слугує надійним фундаментом для подальшого розвитку платформи.

2.3 Веб-технології та засоби розробки інформаційної системи

Правильний підбір інструментів і технологій на етапі проєктування визначає ефективність реалізації бізнес-логіки, рівень безпеки та здатність до масштабування інформаційної системи. Тому фокус підбору засобів розробики було зосереджено на створенні єдиного середовища розробки, яке забезпечує узгодженість клієнтської та серверної частин, спрощує супровід і дозволить гнучко реагувати на зміну вимог.

Сучасні вебзастосунки зазвичай будуються як односторінкові інтерфейси (SPA) або за допомогою серверного рендерингу. Серверний рендеринг із шаблонізаторами підходить для статичних сайтів, але при інтенсивній взаємодії з користувачем поступається SPA у швидкодії та інтерфейсі. Серед SPA-фреймворків Angular вирізняється двосторонньою прив’язкою даних і потужним CLI, але потребує тривалого навчання та має великий розмір бандлу. Vue.js поєднує простоту з гнучкістю api, проте менш поширений у великих корпоративних проєктах [14]. React пропонує декларативну компонентну модель, багату екосистему й можливості для оптимізації в реальному часі. З огляду на необхідність динамічного оновлення статусів завдань і широкого набору готових бібліотек для керування станом та маршрутизації, обрано React із функціональними компонентами та хуками.

На рівні серверної логіки увагу приділено порівнянню провідних фреймворків, які по-різному підходять до обробки запитів, масштабування та підтримки сучасних веб-додатків. Django і Rails забезпечують багатий набір вбудованих засобів для аутентифікації та ORM, але горизонтальне масштабування таких рішень часто ускладнене. Laravel приваблює PHP-спеціалістів, проте менш ефективний при обробці великої кількості одночасних з’єднань. Асинхронна, подієва модель Node.js із фреймворком Express дозволяє використовувати одну мову програмування на всіх рівнях, інтегрувати WebSocket-сервери та реалізувати високопродуктивні API з мінімальними накладними витратами. Саме це поєднання забезпечує оптимальну продуктивність та швидкість розробки.

Для реалізації авторизації було розглянуто власні JWT-механізми, сторонні сервіси Auth0/Okta та Firebase Authentication. Власна JWT-система потребує постійного аудиту й оновлення з огляду на ризики безпеки. Auth0 пропонує розширений функціонал, але пов’язаний із ліцензійними витратами й конфігураційною складністю. Firebase Authentication забезпечує готові SDK для веба й мобільних платформ з підтримкою соціальних мереж, email-паролів і багатофакторної автентифікації, мінімізуючи навантаження на серверну частину й скорочуючи час впровадження [15].

Реляційні СУБД MySQL, MariaDB і PostgreSQL мають широкий спектр застосувань. MySQL відзначається простотою налаштування, MariaDB продуктивністю та відкритою ліцензією, а PostgreSQL розширюваністю, надійною підтримкою транзакцій і повнотекстового пошуку. Для забезпечення стабільної роботи при високих навантаженнях і проведення аналітичних запитів винесено вибір на користь PostgreSQL, що дозволяє використовувати складні типи даних та створювати власні розширення [16].

Обрана архітектура спирається на патерн MVC із RESTful API. Компоненти React відповідають за вигляд (View) та взаємодію із користувачем, серверні маршрути Express додатку реалізують контролери (Controller), які делегують операції на рівень моделей (Model) через ORM Sequelize. Поділ на окремі шари забезпечує високу тестованість, можливість впровадження кешування, черг повідомлень і подальшого переходу до мікросервісного дизайну [17].

Отже, поєднання React, Node.js/Express, Firebase Authentication і PostgreSQL формує стек, що забезпечує швидкість розробки, високу продуктивність та надійний захист даних. Використання єдиної мови JavaScript скорочує складність комунікації між командами, а готові сервіси Firebase знижують експлуатаційні витрати на підтримку авторизації. PostgreSQL гарантує стійкість та гнучкість при роботі з об’ємними та складними запитами. Такий набір технологій закладає міцну основу для подальшого розширення й масштабування платформи управління завданнями.

# Розділ 3

Програмна реалізація інформаційної системи

3.1 Структура та функціональне призначення складових системи

Сучасна вебплатформа для командної роботи та обміну завданнями має відрізняється високим рівнем надійності та гнучкості завдяки чітко виокремленим рівням відповідальності й взаємодії між компонентами. В данному проєкті архітектура реалізована за принципом трирівневої моделі. Верхній шар контролерів відповідає за обробку HTTP-запитів, первинну валідацію та маршрутизацію даних до бізнес-логіки, проміжний шар сервісів керує всіма алгоритмами створення, оновлення та видалення ключових сутностей, а саме користувачів, команд, завдань, файлів і коментарів. Нижній шар ORM-моделей відображає фізичну структуру реляційної бази даних і виконує безпосередню роботу зі збереженням інформації [18].

Така багаторівнева організація гарантує ізоляцію логіки від деталей транспортування та зберігання даних, що сприяє зручності тестування, швидкому впровадженню нових модулів та стійкості системи під час зростання навантаження. Виходячи з вимог до вебсистеми для управління проєктами та завданнями, було створено детальну схему взаємозв’язків між п’ятьма ключовими сутностями: Users, Teams, Tasks, Files та Comments. Для кожної сутності передбачено окремий контролер із багатофункціональним набором методів, які забезпечують повний цикл роботи, а також додаткові дії, аутентифікацію та авторизацію в UserController, координацію командних налаштувань і ролей у TeamController, керування життєвим циклом завдань у TaskController, обробку файлів у FileController та підтримку коментарів із можливістю реакцій у CommentController.

Модель взаємодій встановлює канали обміну даними між шарами й гарантує цілісність інформації на кожному етапі її трансформації. Вибране архітектурне рішення створює надійну основу для подальшого масштабування, підвищення продуктивності та розвитку. Для візуалізації архітектури системи розроблено схему, яка демонструє зв’язки між контролерами та ORM-моделями (рисунок 3.1).

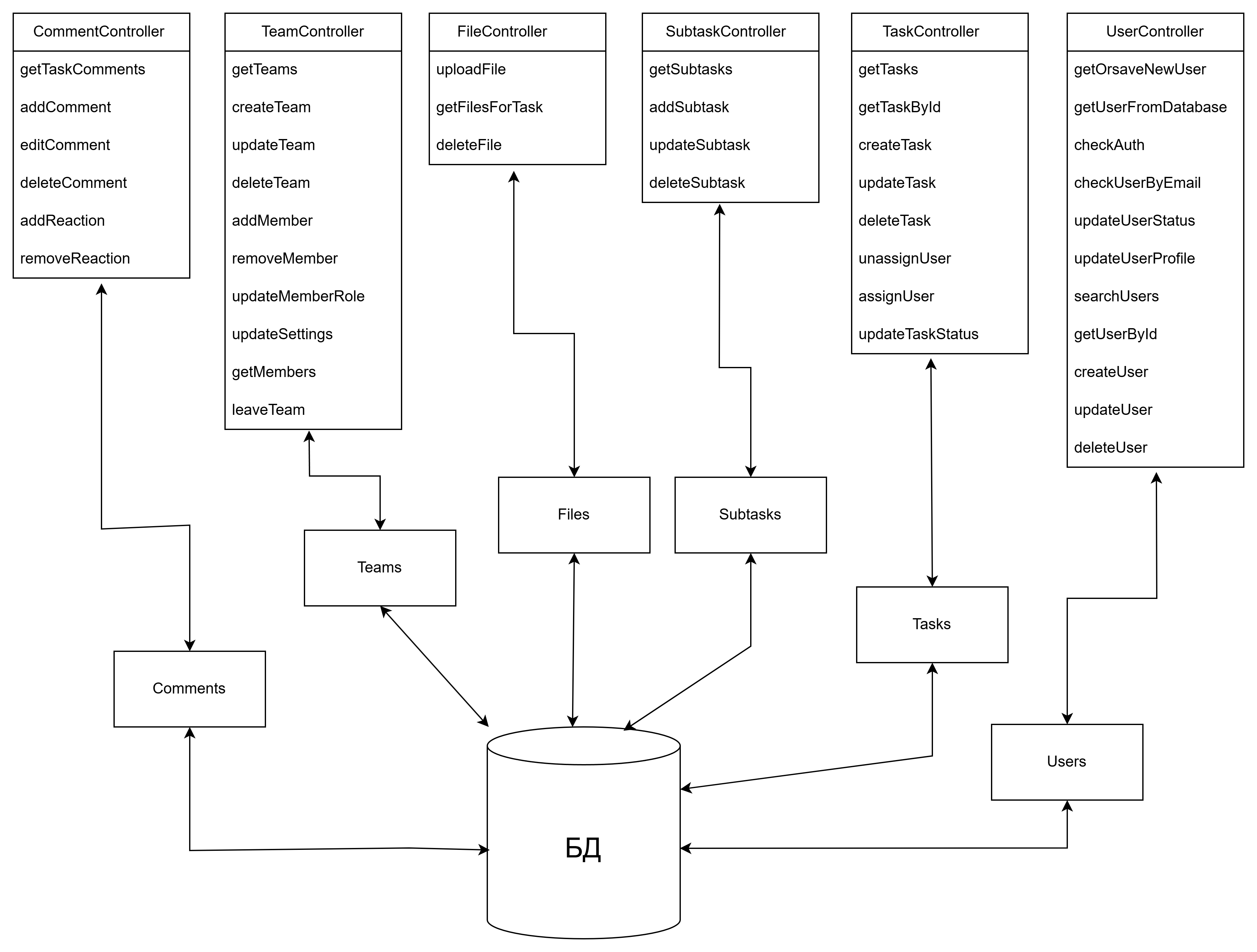


Рисунок 3.1 – Схема взаємозв’язку контролерів та ORM-моделей системи управління завданнями

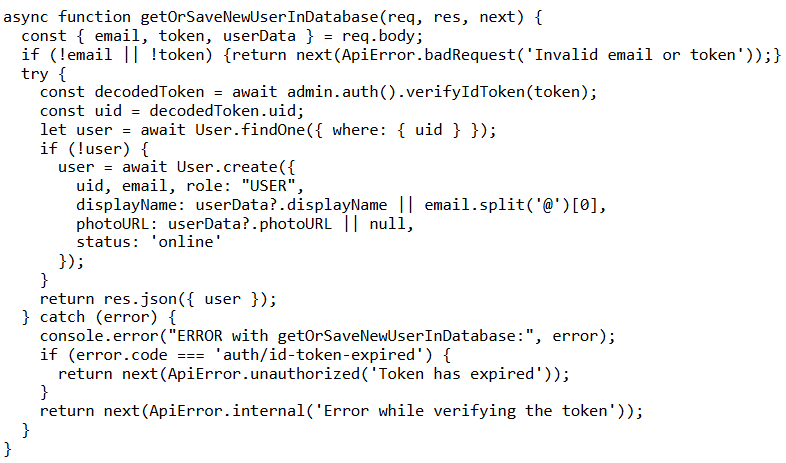
Контролер TeamController надає інтерфейс для роботи з командами та їх учасниками, зокрема створює і видаляє команди, підтримує управління правами та налаштуваннями. Таблиця Teams зберігає основні метадані про кожну команду, включно з властивістю ownerId, яка є зовнішнім ключем до таблиці Users і визначає власника. Зв’язна таблиця TeamMembers реалізує відношення «багато-до-багатьох» між Users і Teams, де роль учасника дозволяє гнучко налаштовувати рівні доступу. Таблиця Tasks прив’язана до Teams через зовнішній ключ teamId і до автора завдання через creatorId, забезпечуючи можливість виконання завдань в межах командних проєктів. Паралельно FileController обслуговує ресурси в таблиці Files, а CommentController для обговорення в таблиці Comments, що дозволяє зберігати коментарі.

На основі схеми, можна зробити висновок, що саме такі зв’язки гарантують цілісність даних і дозволяють будувати надійну логіку через контролери, які модифікують відповідні моделі, та відтворюють сутності в базі даних, а зовнішні ключі оберігають від розривів у зв’язках. А незалежна логіка кожного компонента спрощує написання тестів і дозволяє виконувати масштабування окремих модулів без ризику порушення цілісності всієї системи.

Отже, структурна схема демонструє взаємодію між контролерами, моделями й таблицями бази даних, що є ключовою умовою для успішного функціонування вебсайту системи управління завданнями та подальшого розвитку функціоналу.

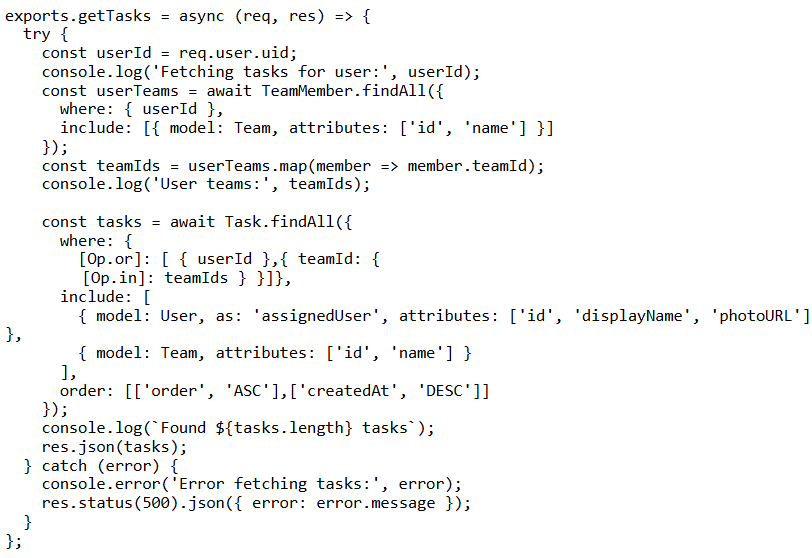
3.2 Особливості реалізації програмних складових системи

В наступних прикладах буде розглянуто реалізацію ключових програмних компонентів вебсистеми керування завданнями, починаючи від механізму автентифікації користувачів і закінчуючи обробкою подій у реальному часі. Аутентифікації користувачів реалізована наступним чином, застосовано пакет Firebase Admin SDK, який дозволяє безшовно інтегрувати зовнішній механізм OAuth у внутрішню модель користувачів. За допомогою асинхронної функції відбувається верифікація ID-токена, пошук існуючого запису в базі даних за uid та, за необхідності, створення нового користувача з виставленими за замовчуванням роллю, іменем, URL-адресою аватара й оновленим статусом online, лістинг функції наведено нижче:



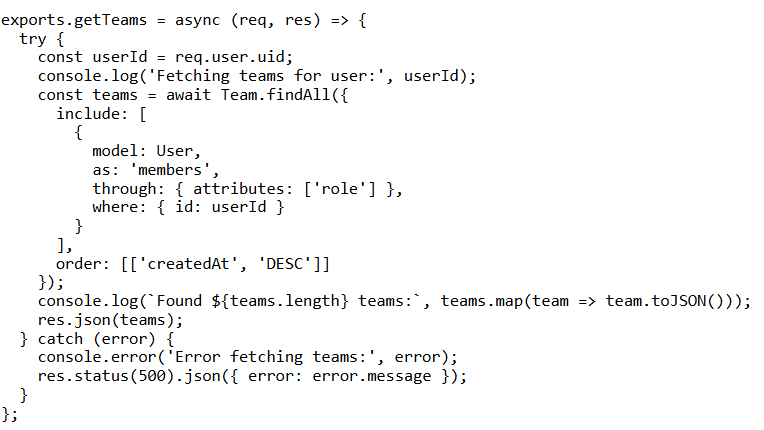
Після перевірки наявності обов’язкових полів email та token виконується асинхронний виклик verifyIdToken, що підтверджує достовірність користувача. У разі успішної валідації система шукає запис з таким uid, а якщо його немає створює новий із заданими атрибутами. Обробка помилок передбачає повернення відповідних HTTP-кодів при протермінованому токені або внутрішніх збоях сервісу.

Контролер завдань використовує ORM Sequelize для одночасного отримання як особистих, так і групових завдань поточного користувача. Спочатку витягується список команд, у складі яких він перебуває, формуються teamIds, після чого основним запитом вибираються всі завдання, що належать або безпосередньо користувачу, або його командам. Асоціації дозволяють включити в результат ім’я та фото відповідального виконавця, а також назву команди, лістинг коду:



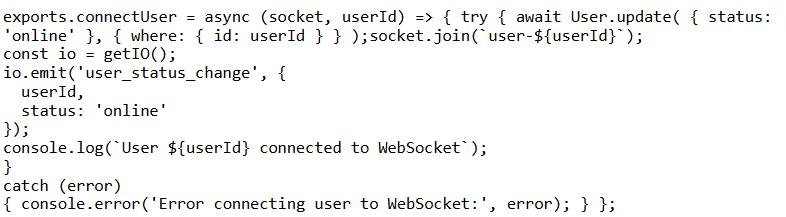
Першим кроком виконується запит до таблиці TeamMember з включенням моделі Team, щоб дізнатися, до яких команд належить користувач. Отримані ідентифікатори використовуються в умовах where запиту до таблиці Task, де за допомогою оператора Op.or комбінуються особисті та командні завдання. Параметр order гарантує спочатку сортування за пріоритетом, а потім за датою створення.

Для формування списку команд поточного користувача відбувається запит до моделі Team з включенням через асоціативну таблицю TeamMember, що дозволяє миттєво дізнатися роль користувача в кожній команді та відсортувати результат за датою створення, лістинг функції подано нижче:



Метод findAll з параметром include дозволяє через проміжну таблицю отримати роль кожного учасника. Сортування за полем createdAt робить відображення найновіших команд першими, що відповідає очікуванням користувача.

Для забезпечення реального часу у взаємодії користувачів із системою використано WebSocket. При підключенні клієнта автоматично оновлюється поле status в таблиці користувачів на online, користувач приєднується до своєї кімнати й усім активним клієнтам розсилається подія user\_status\_change з відповідним userId. Лістинг відповідної функції:



Підхід гарантує, що кожен клієнт отримує лише релевантні повідомлення без зайвого навантаження мережі. Оновлення статусу в базі даних забезпечує узгодженість інформації між HTTP та WebSocket-з’єднаннями, що особливо важливо при масштабуванні.

Отже, програмні модулі забезпечують реалізацію ключових сценаріїв роботи користувача від реєстрації та авторизації до управління завданнями, командами та відстеження статусу в реальному часі. Це дозволяє не лише швидко ідентифікувати проблемні місця під час розробки чи підтримки, але й гарантує коректне реагування клієнтської частини на різноманітні помилки, включаючи прострочені токени чи недоступність зовнішніх сервісів.

3.3 Тестування інформаційної системи

Тестування інформаційної системи є ключовим етапом гарантування якості, адже саме воно дозволяє переконатися в безперебійній роботі кожного з компонентів під навантаженням та в різних умовах використання. В ході розробки було визначено, що особливу увагу слід приділити механізмам контролю доступу та авторизації, коректності обробки та зберігання даних завдань, а також управлінню командами та взаємодії з користувачами через файли та коментарі. Для кожного із цих напрямків були сформовані тест-кейси, які описують реальні сценарії роботи системи та дозволяють відтворити ключові бізнес-процеси.

Авторизація в системі виступає фундаментальним компонентом, без якого неможливо гарантувати захищений доступ до особистого кабінету та ресурсів. Тому перед виконанням інших перевірок проводилося тестування логіну, користувач вводить свій email і пароль, система робить запит до бекенду, перевіряє облікові дані, генерує JWT-токен і повертає його клієнту. Збереження цього токена у локальному сховищі браузера слугує доказом успішної автентифікації.

У таблиці 3.1 наведено покроковий опис цього процесу разом із очікуваним результатом. Після виконання тест-кейсу користувач без затримок потрапляє на дашборд, а токен можна побачити в сховищі, що свідчить про стабільну роботу механізму авторизації.

Таблиця 3.1 – Тест-кейс SU0428

|  |  |  |  |
| --- | --- | --- | --- |
| **Тест-кейс: SU0428** | **Пріоритет:** 1 | **Створено:** 25.05.2025 | |
| **Назва:** Перевірка автентифікації користувача (логін)  **Передумови:** Користувач зареєстрований та має дійсні email i пароль, відкрито сторінку входу в систему | | | |
| **Кроки** | | | **Очікуваний результат** |
| 1. Ввести у поле «Email» дійсний email зареєстрованого користувача;  2. Ввести у поле «Пароль» коректний пароль;  3. Натиснути кнопку «Увійти»;  4. Отримати JWT у відповіді, збережений у локальному сховищі браузера;  5. Перейти на сторінку профілю або дашборду. | | | Поля для входу заповненні коректно, після чого відбувається запит на сервер та отримання токена. Після чого відображається головна сторінка. |
| **Результат виконання тест-кейсу:** пройдено успішно | | | |

Після виконання кроків, описаних у таблиці вище, було отримано результат, який відображено на рисунку 3.2. Користувач успішно пройшов валідацію даних, у локальному сховищі з’явився JWT-токен, а інтерфейс автоматично перенаправив на дашборд.

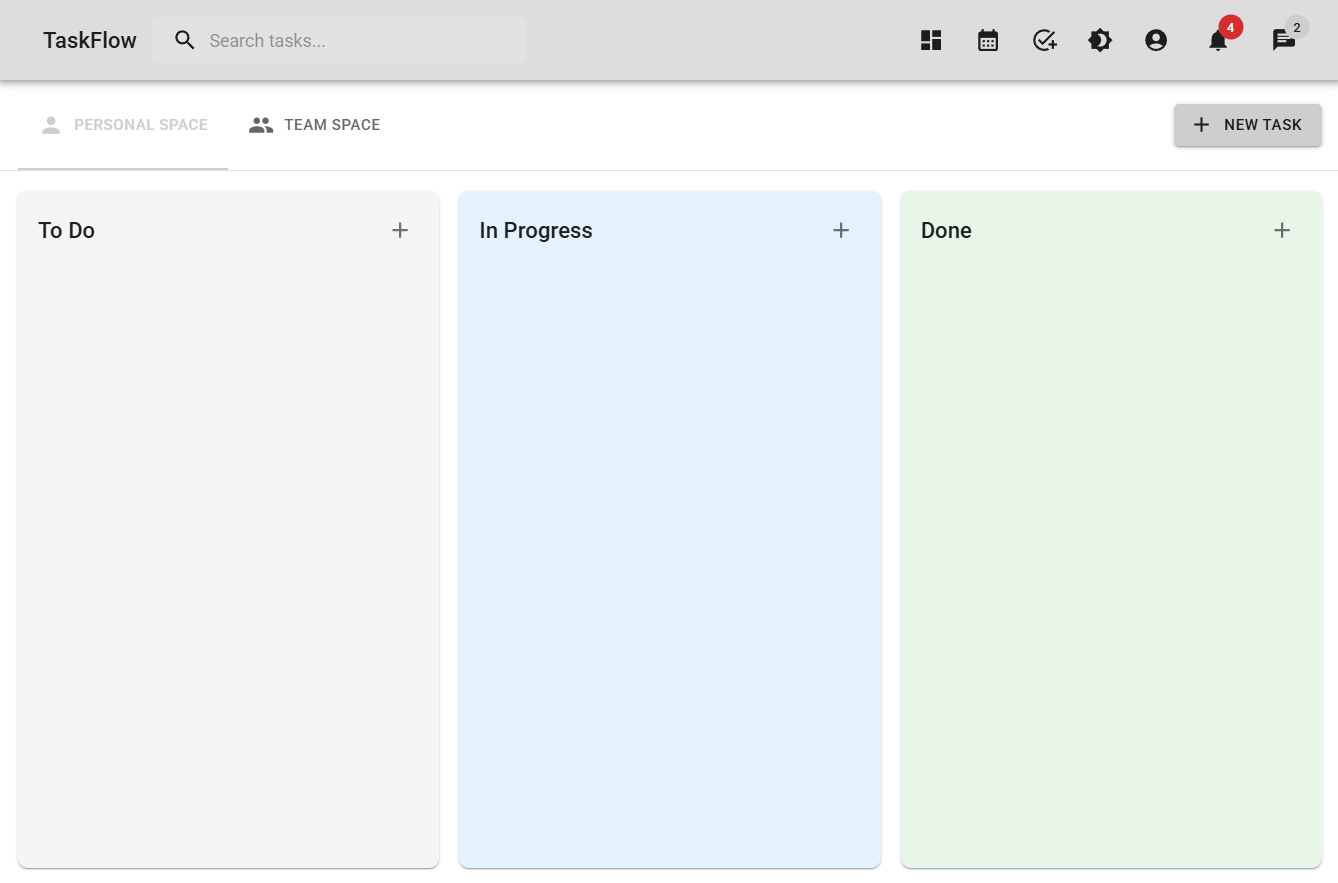


Рисунок 3.2 – Головна сторінка вебсайту системи управління завданнями

Перенаправлення на головну сторінку свідчить те що користувач успішно авторизувався в системі та сервер повернув дані які користувач створював, а саме завдання в особистому просторі.

Наступним критичним етапом є створення нових завдань. Будь-яка затримка або некоректне збереження полів завдання може призвести до порушення робочих процесів. У таблиці 3.2 описано послідовність дій користувача, який заповнює форму створення завдання, обирає відповідні параметри та відправляє форму на сервер.

Таблиця 3.2 – Тест-кейс EI7432

|  |  |  |  |
| --- | --- | --- | --- |
| **Тест-кейс: EI7432** | **Пріоритет:** 1 | **Створено:** 25.05.2025 | |
| **Назва:** Створення нового завдання  **Передумови:** Користувач пройшов автентифікацію та перебуває на дашборді | | | |
| **Кроки** | | | **Очікуваний результат** |
| 1. Натиснути кнопку «Add task» на дашборді;  2. Заповнити поле «Назва»;  3. Натиснути «Створити»;  4. Натиснути кнопку «Edit» на створеному завданні та перевірити можливість редагування та внесення змін. | | | 1. Відкривається форма створення завдання;  2. Поле заповнене без помилок;  3. Виконується запит на сервер, після чого у списку відображається нове завдання;  4. Відкривається вікно завдання з можливостями зміни та модифікації. |
| **Результат виконання тест-кейсу:** пройдено успішно | | | |

Після виконання всіх дій, система створила нове завдання з усіма заповненими полями та відобразила його у списку завдань без необхідності ручного оновлення сторінки. На рисунку 3.3 показано, головну сторінку з створеним завданням.

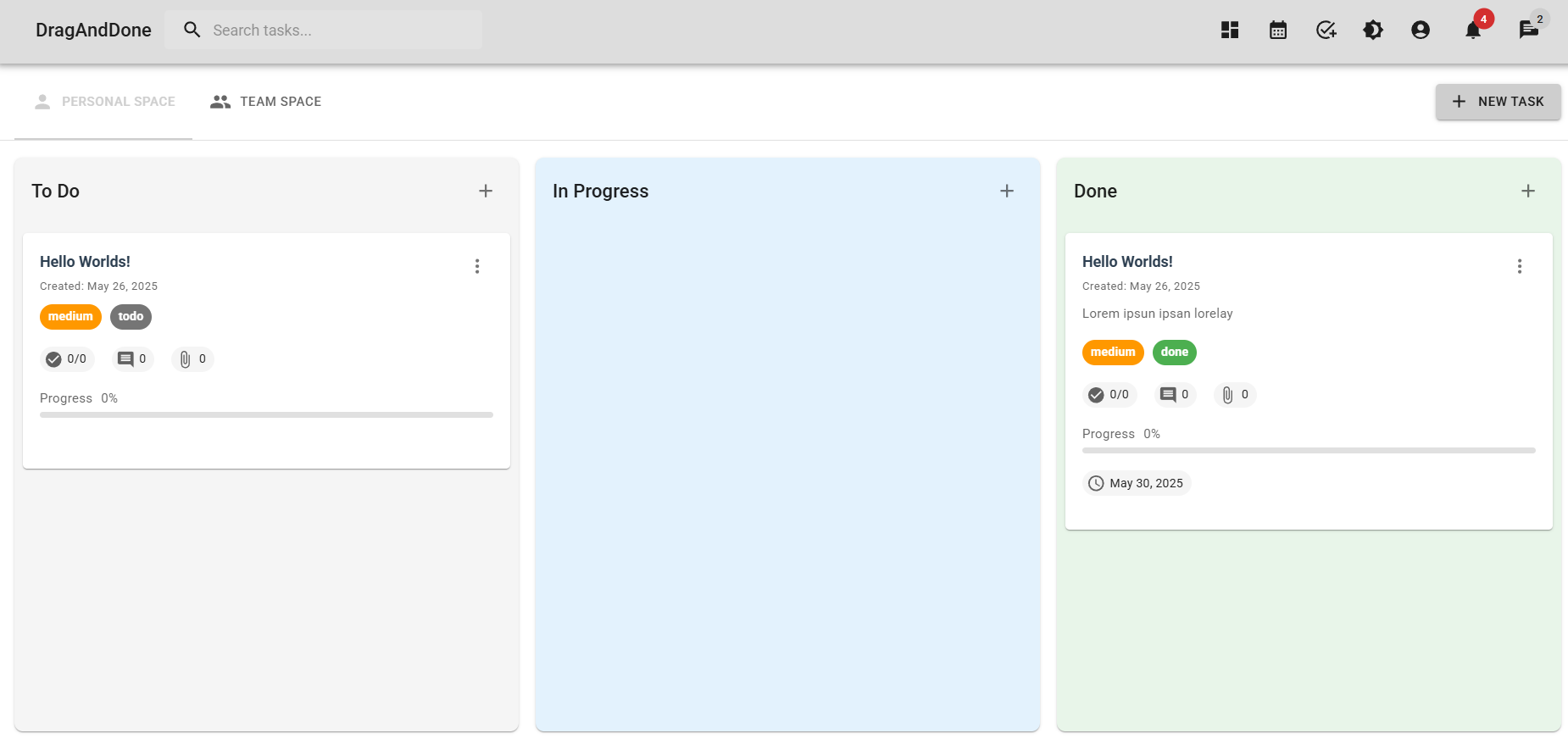


Рисунок 3.3 – Сторінка головного екрану з створеним завданням

Ключовим елементом системи є інструмент коментування завдань, який сприяє оперативній взаємодії членів команди та зберігає історію дискусій у контексті кожного завдання. Розроблений тест-кейс, представлений у таблиці 3.3, метою якого є перевірка функціональності додавання нового коментаря, від відправки тексту користувачем до відображення його в секції коментарів та точним часом створення. Це гарантує, що всі повідомлення фіксуються коректно і стають доступними для огляду.

Таблиця 3.3 – Тест-кейс ME8265

|  |  |  |  |
| --- | --- | --- | --- |
| **Тест-кейс: ME8265** | **Пріоритет:** 1 | **Створено:** 25.05.2025 | |
| **Назва:** Додавання коментаря до завдання  **Передумови:** Користувач пройшов автентифікацію та знаходиться на сторінці конкретного завдання | | | |
| **Кроки** | | | **Очікуваний результат** |
| 1. Натиснути на секцію «Коментарі»;  2. У текстове поле ввести текст, наприклад «Hello world»;  3. Натиснути кнопку «Додати коментар»;  4. Переконатися, що коментар додався та зазначено час створення. | | | 1. Відображається поле для введення нового коментаря;  2. Текст відображається;  3. Виконується запит на сервер;  4. Новий коментар успішно доданий із часом створення. |
| **Результат виконання тест-кейсу:** пройдено успішно | | | |

Після виконання кроків тест-кейсу система зберегла новий коментар у базі даних та відобразила його у секції коментарів без перезавантаження сторінки (рисунок 3.4).

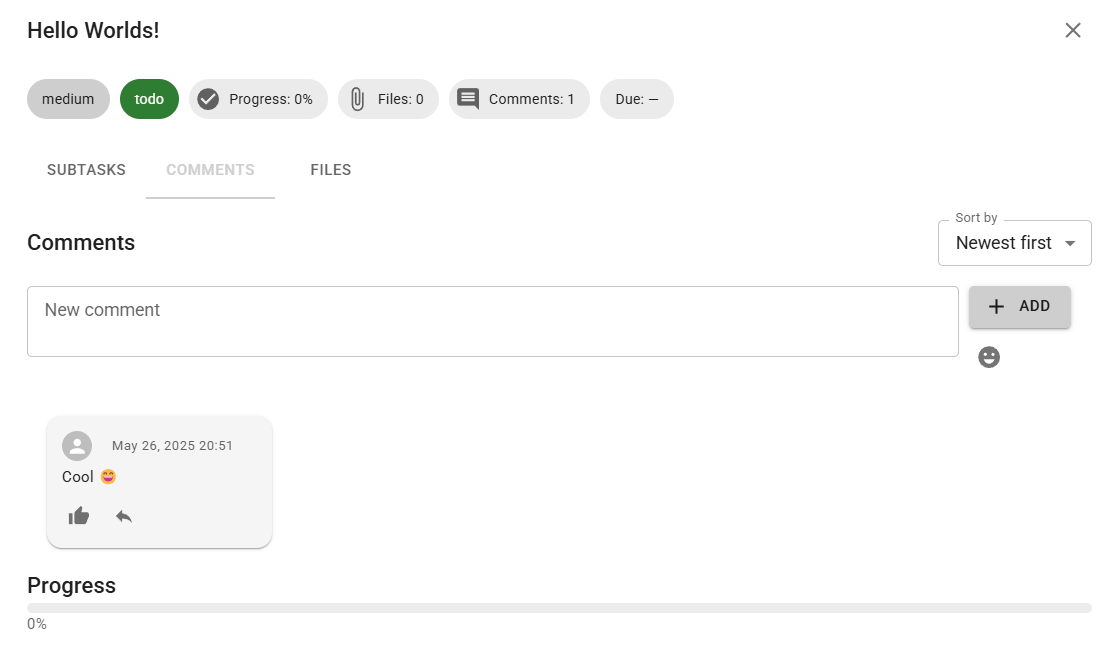


Рисунок 3.4 – Вікно секції коментарів у конкретному завданні

Після виконання всіх тест-кейсів система продемонструвала стабільну роботу критичних модулів. Авторизація підтвердила видачу дійсного JWT-токена та правильний редирект на дашборд, створення завдань забезпечило коректну передачу й відображення всіх полів.

Отже, проведене тестування засвідчило, що базова функціональність платформи управління завданнями реалізована коректно і готова до масштабування та подальшого розширення можливостей. На основі отриманих результатів можна покращувати існуючі можливості та розробляти додаткові сервіси, інтеграцій та вдосконалення інтерфейсу.

3.4 Інструкція користувача

Після переходу на головну сторінку вебзастосунку користувач побачить вікно автентифікації, де необхідно ввести електронну пошту та пароль і натиснути кнопку увійти або вибрати увійти за допомогою інших сервісів, таких як google або github (рисунок 3.5).

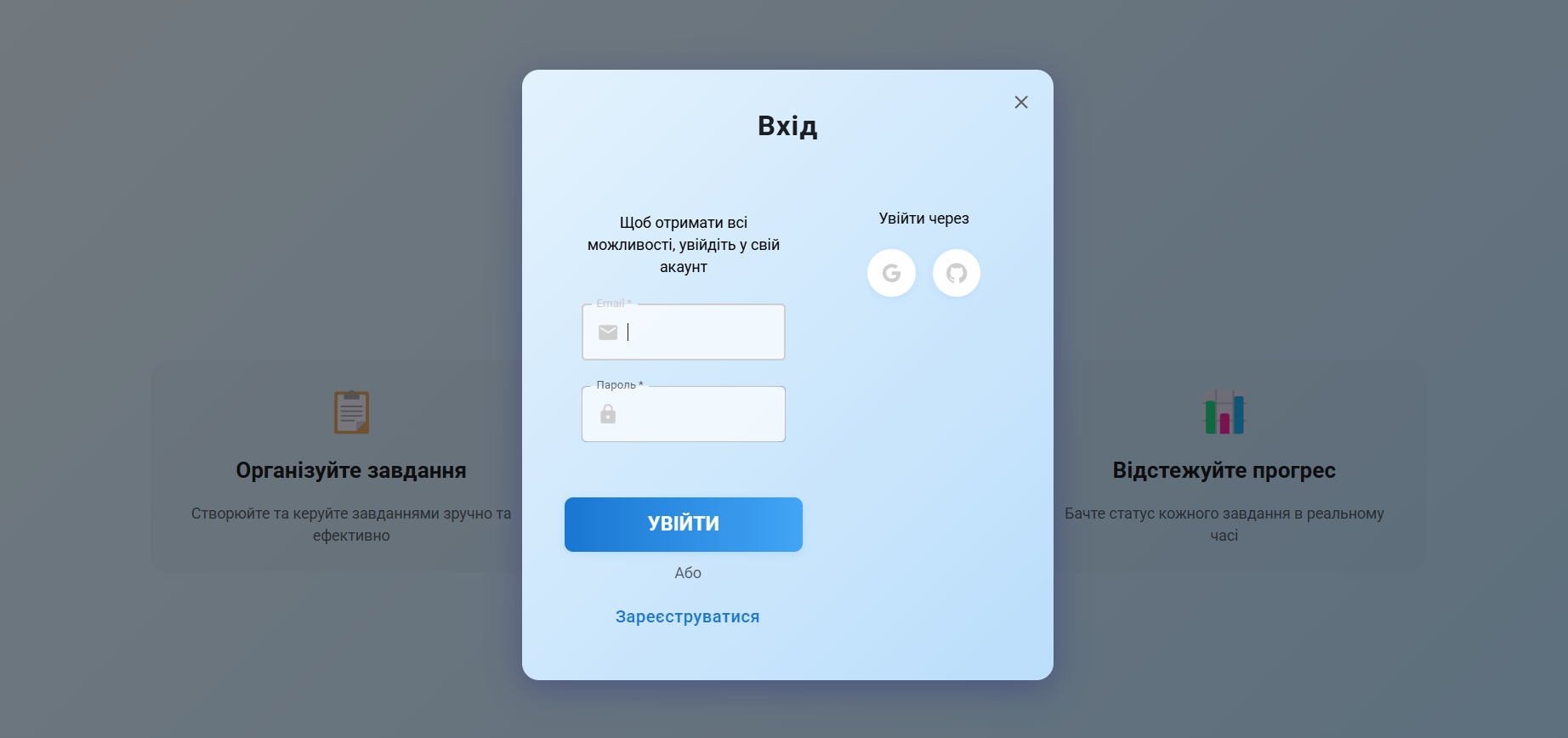


Рисунок 3.5 – Вікно автентифікації користувача

Після успішного входу автоматично відкривається головне вікно дашборд із трьома колонками для завдань: To Do, In Progress та Done, а також панеллю швидкого доступу до створення завдання, команди, аналітики та профілю, що зображено на рисунку 3.6.

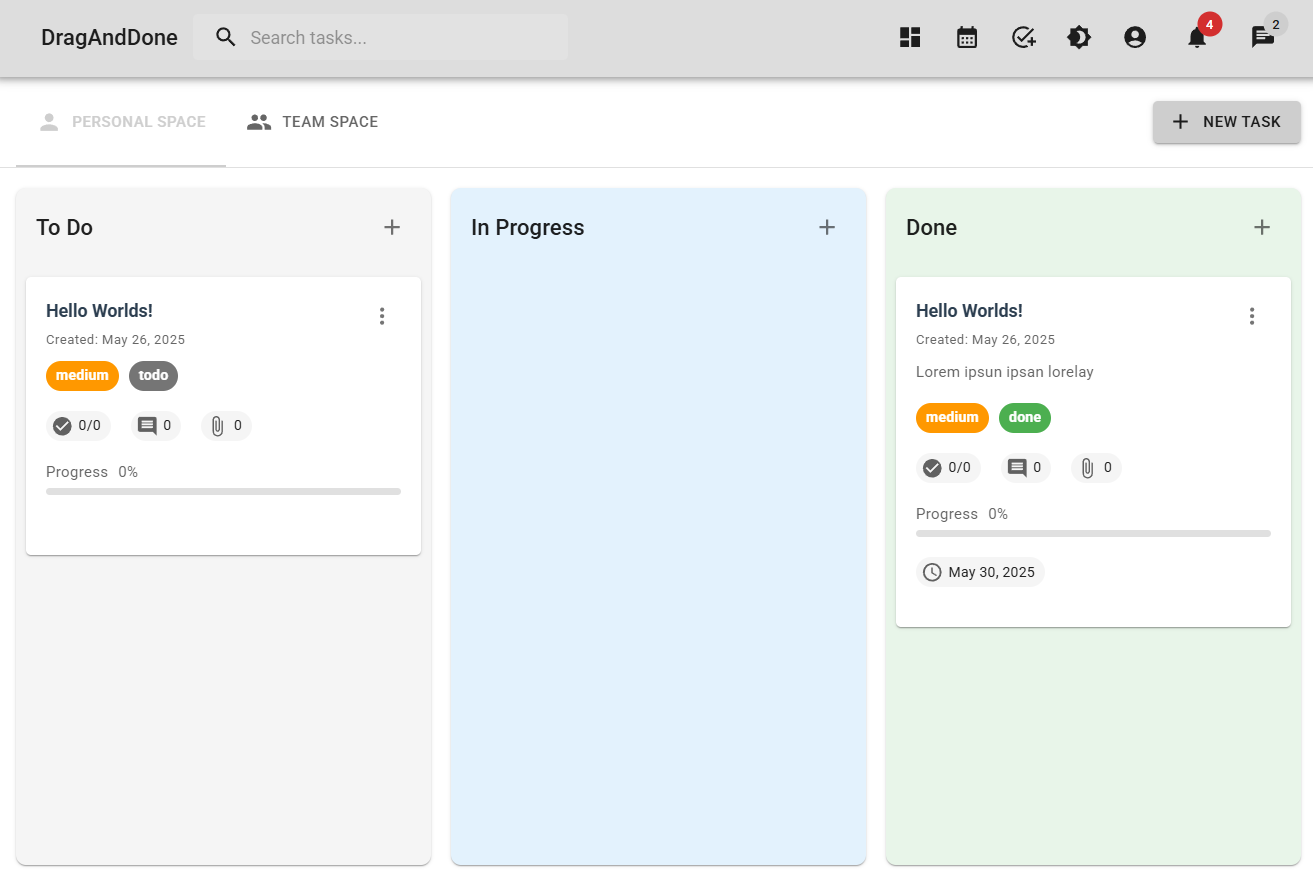


Рисунок 3.6 – Головне вікно зі списками завдань та навігацією

Для створення нового завдання слід натиснути кнопку “Нове завдання” та ввести назву після чого завдання автоматично з’явиться в колонці To Do (рисунок 3.7).

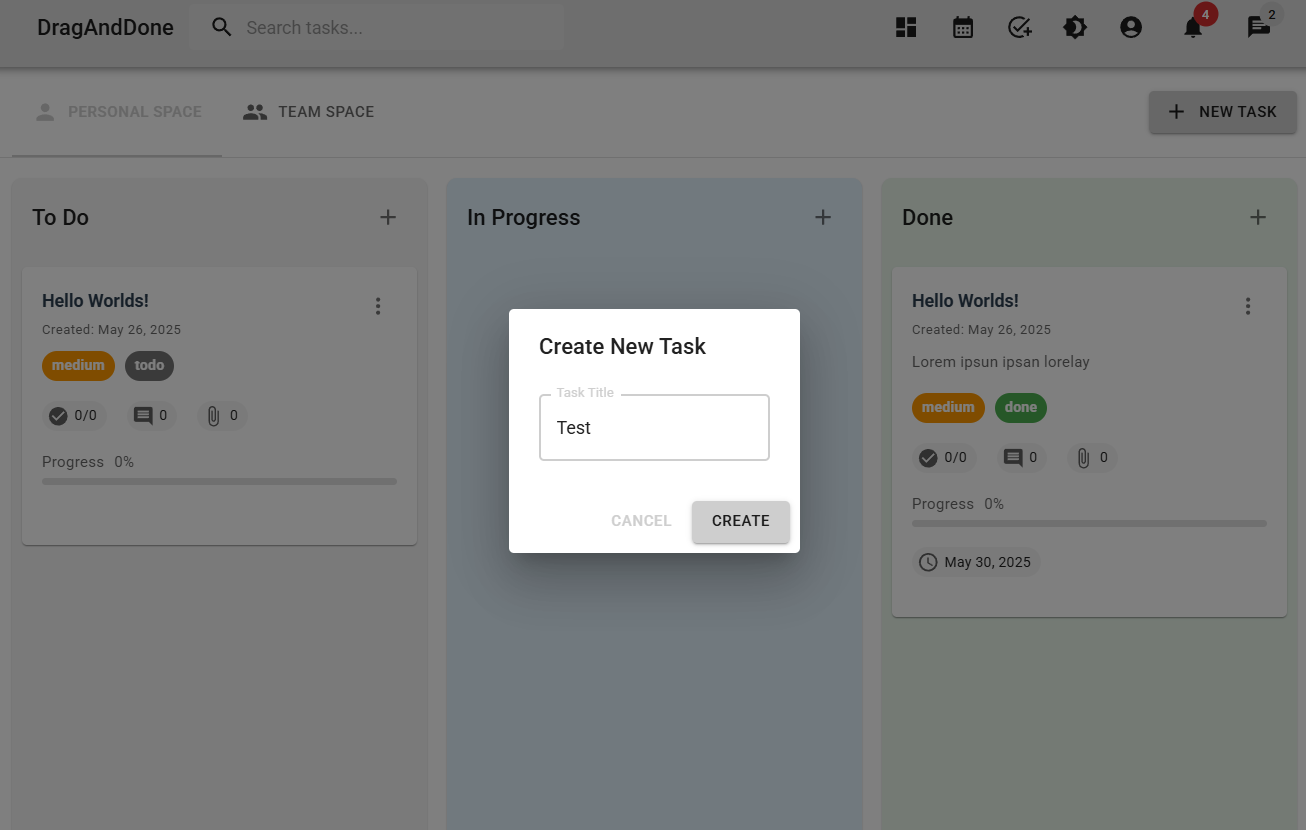


Рисунок 3.7 – Форма створення нового завдання

Щоб змінити параметри наявного завдання, необхідно навести курсор на його картку та натиснути іконку редагування, після чого в діалоговому вікні можна дописати детальний опис, залишити коментар або додати підзавдання (subtasks), а також змінити термін чи пріоритет, після чого збережені зміни миттєво оновлять картку (рисунок 3.8).

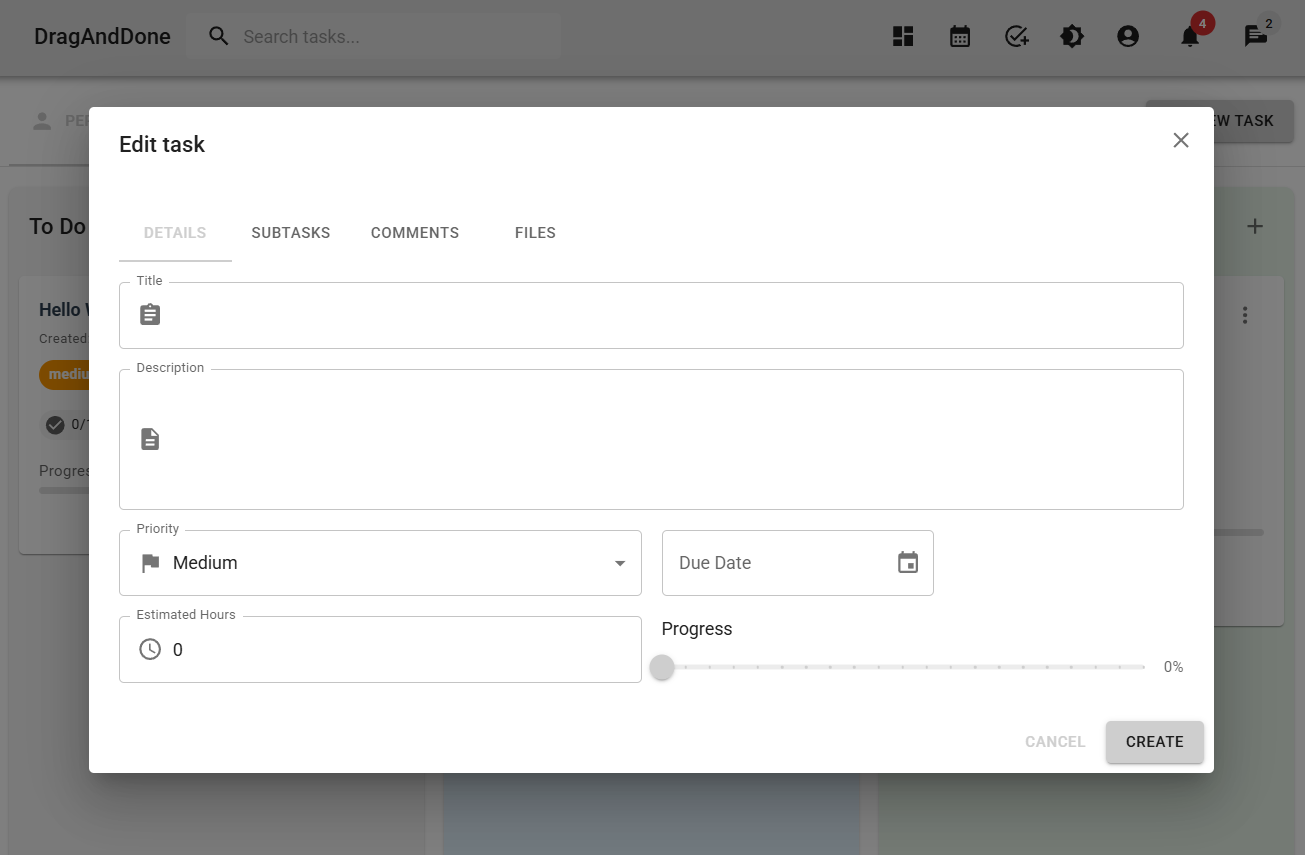


Рисунок 3.8 – Вікно редагування завдання з полями для коментаря та підзавдань

Перегляд повних деталей виконується кліком по картці завдання: у вікні деталей відображаються вся інформація про завдання, перелік підзавдань із чекбоксами та коментарі, а також є можливість швидко переключити статус чи додати новий коментар (рисунок 3.9).

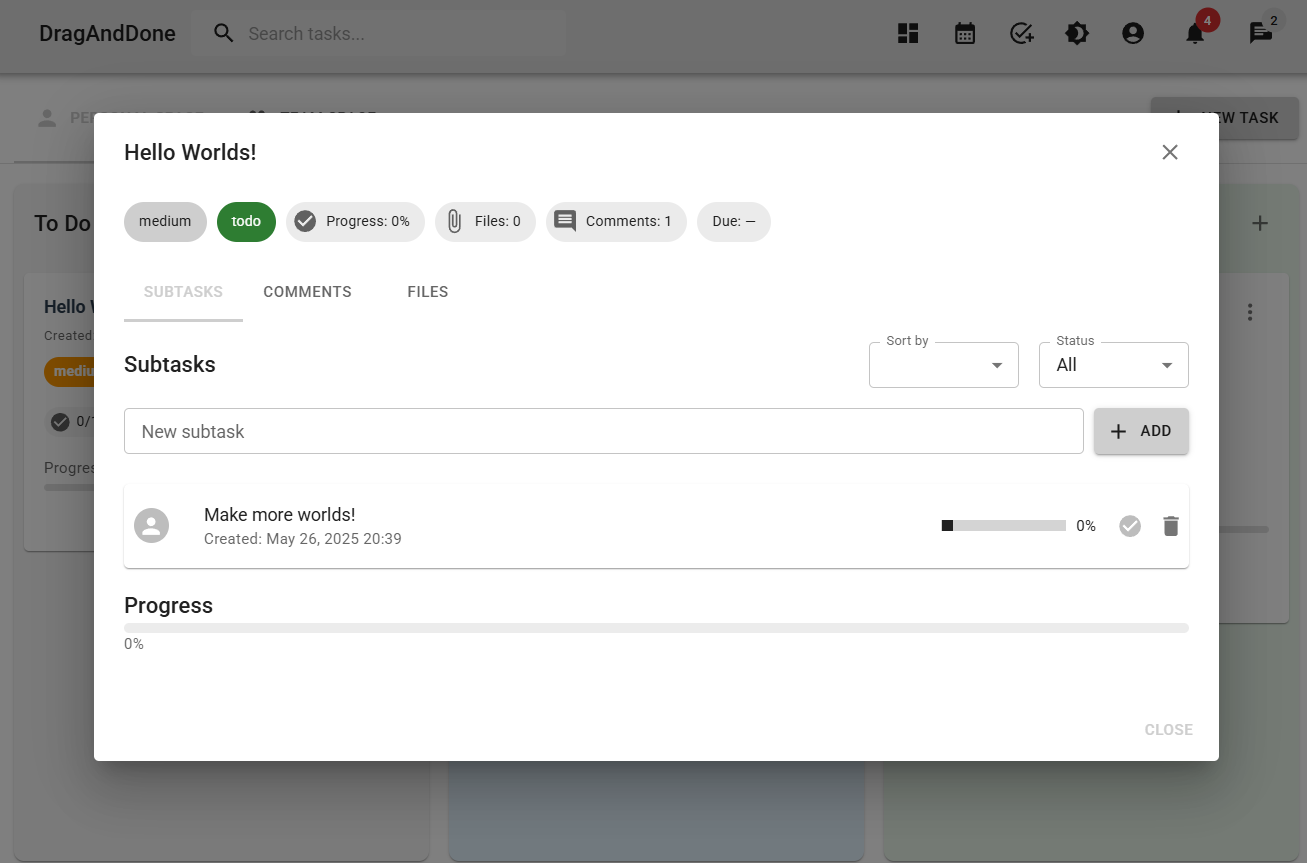


Рисунок 3.9 – Вікно деталей завдання

На рисунку 3.10 зображено зміну статусу завдання, яке здійснюється за допомогою drag and drop: достатньо перетягнути картку з однієї колонки в іншу, і платформа автоматично збереже новий стан.

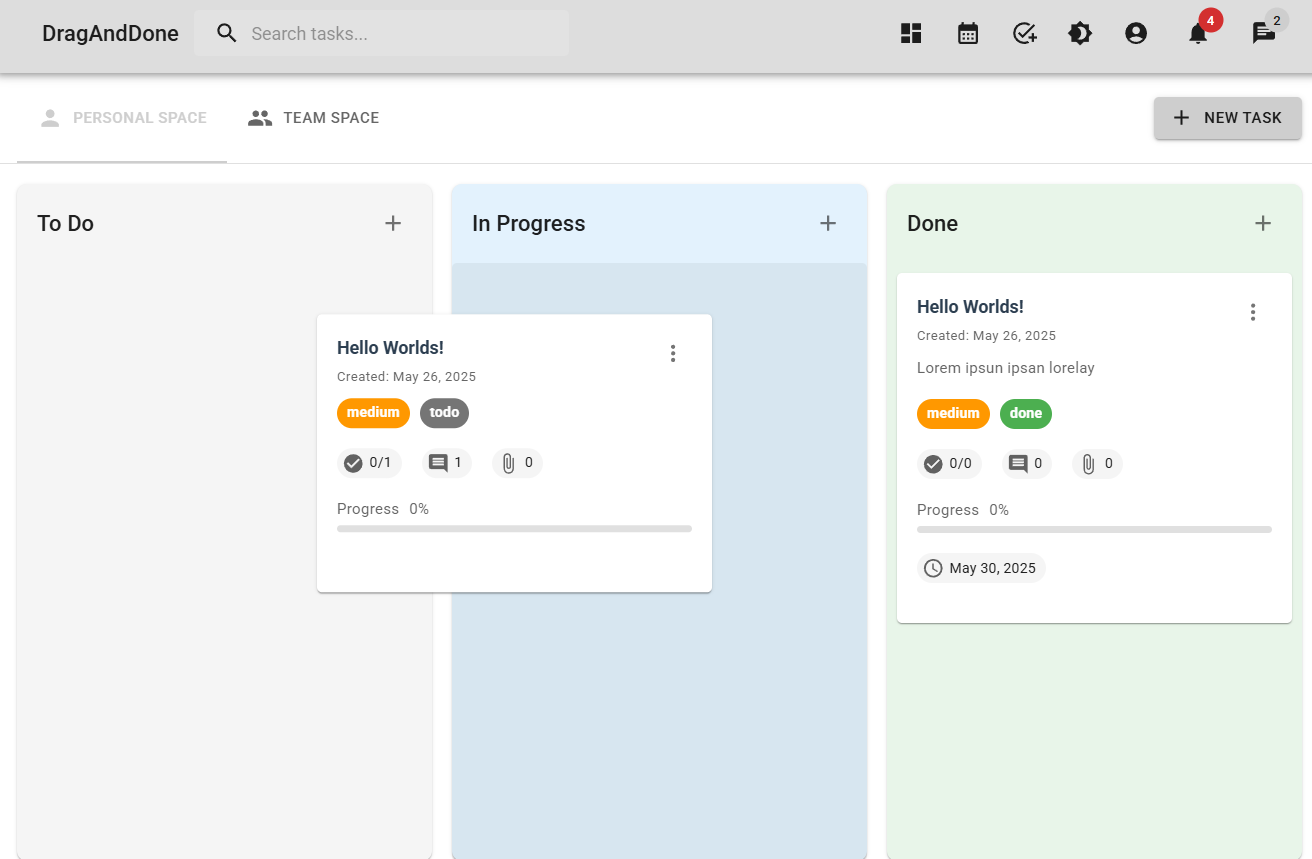


Рисунок 3.10 – Перетягування завдання між колонками методом drag and drop

Для командної роботи слід перейти на вкладку “Команда”, створити нову групу, ввести її назву та опис, після чого можна додавати учасників за електронною поштою й спільно створювати та редагувати завдання (рисунок 3.11).

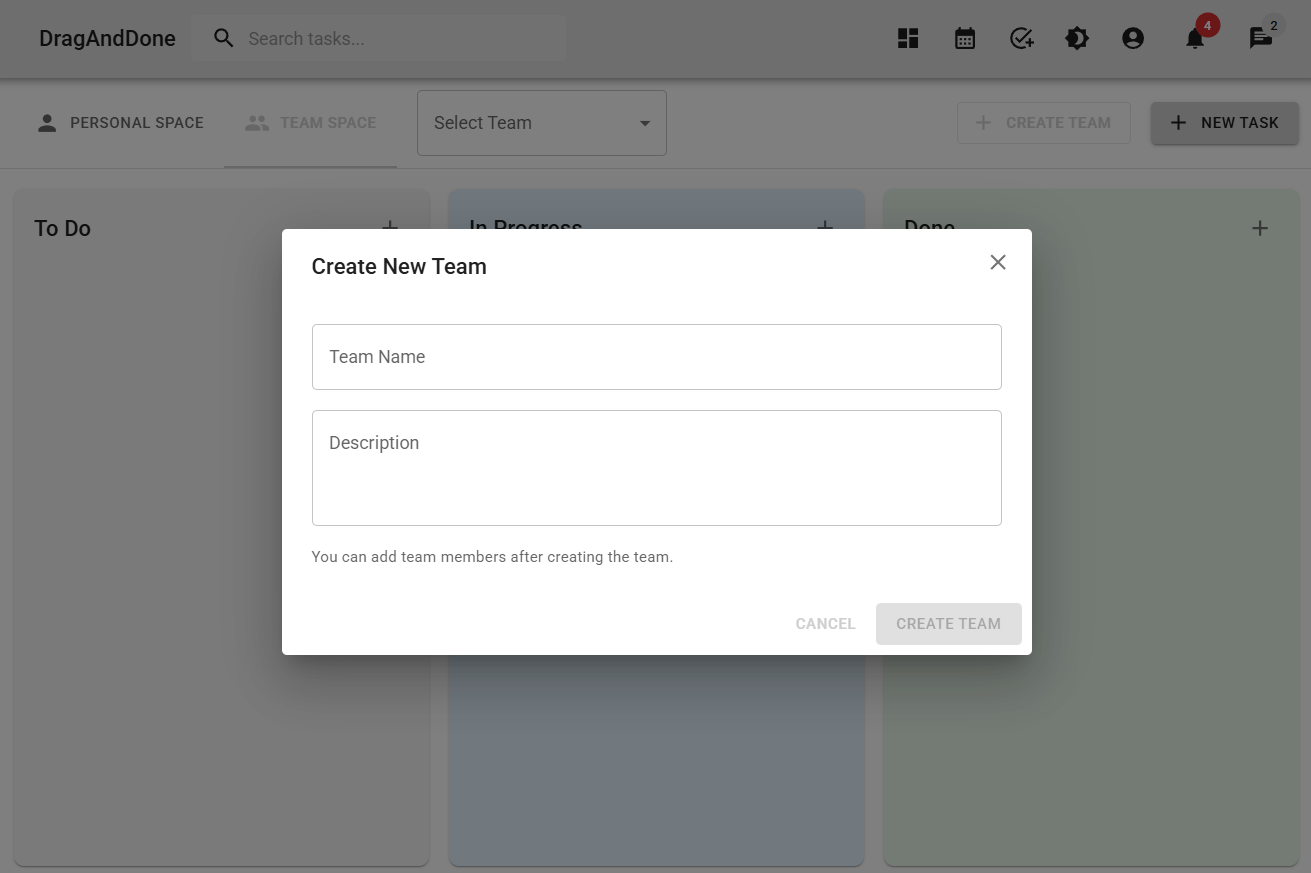


Рисунок 3.11 – Вкладка команди з формою створення нової групи

Налаштування особистих даних здійснюється через розділ “Профіль” у верхньому правому куті інтерфейсу, де можна змінити ім’я, фото та контактну інформацію, видалити створені команди та інші можливості (рисунок 3.12).

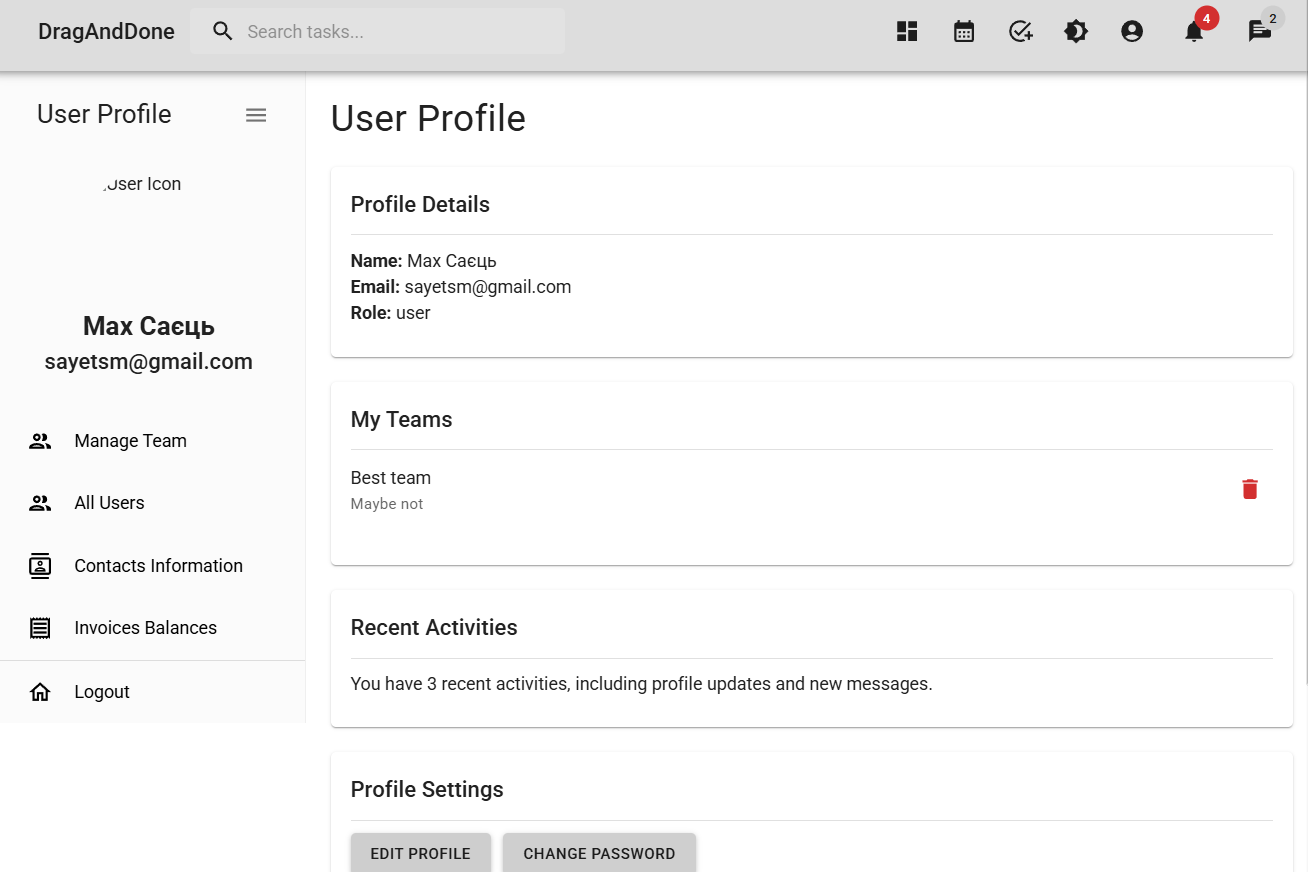


Рисунок 3.12 – Розділ профілю користувача

Аналітика доступна на окремій вкладці й містить графіки, що відображають кількість завдань у кожному статусі, середній час виконання та активність команди з можливістю фільтрації (рисунок 3.13).

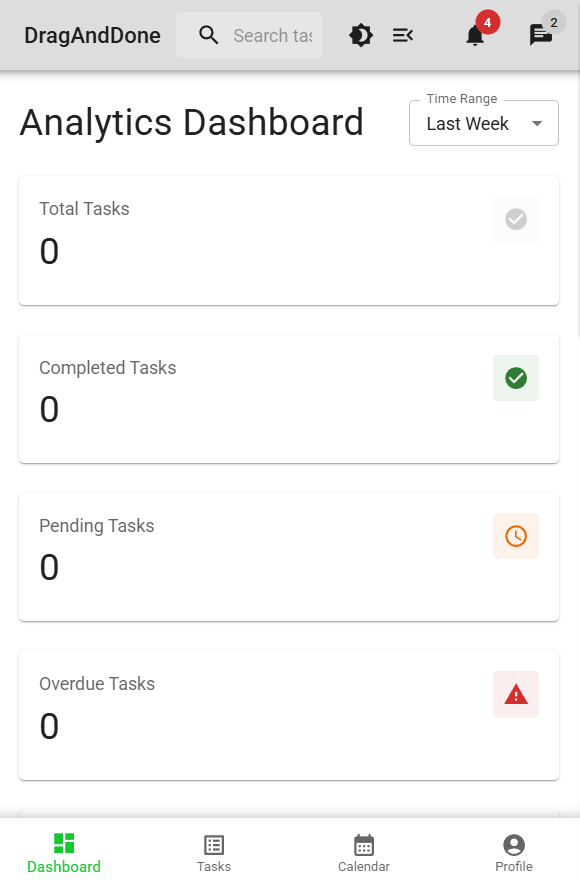


Рисунок 3.13 – Інтерфейс аналітики

Отже, зареєстрований користувач отримує повноцінний та інтуїтивно зрозумілий інструментарій для організації й контролю виконання завдань: від авторизації та переходу на дашборд до створення, редагування й детального перегляду завдань, перетягування їх між статусами за допомогою drag and drop, а також спільної роботи в межах команд. Крім того, налаштування профілю та доступ до розширеної аналітики дозволяють персоналізувати досвід користувача й отримувати завжди актуальну інформацію про продуктивність роботи, що сприяє підвищенню ефективності процесів та взаємодії всередині команди.

3.5 Вимоги до розгортання інформаційної системи

Успішне розгортання системи керування завданнями вимагає налаштування двох складових серверного оточення та клієнтської платформи. З боку сервера необхідно забезпечити наявність інтерпретатора Node.js версії не менше ніж 18.x із менеджером пакетів NPM або Yarn, оскільки саме на цьому середовищі працюють асинхронні контролери, WebSocket-сервер та ORM Sequelize. Також обов’язково встановити PostgreSQL версії 14.0 або новішої для зберігання пов’язаних таблиць, а також налаштувати файл .env із змінними оточення.

Система потребує мінімум 2 ГБ оперативної пам’яті та двоядерного процесора на етапі тестування, а для промислового навантаження бажано мати принаймні 4 ГБ ОЗП, SSL-сертифікат для захищеного HTTPS-з’єднання та безперервний доступ до мережі Інтернет для верифікації токенів.

З боку клієнта користувачеві достатньо скористатися будь-яким сучасним браузером із підтримкою ECMAScript 2020, WebSocket, Fetch API та CSS Grid/Flexbox, з увімкненим JavaScript і дозволом спливаючих вікон.

Мобільним користувачам рекомендовано використовувати найновішу версію Chrome чи Safari, які гарантують коректну роботу drag-and-drop інтерфейсу, динамічної підвантажувальної фільтрації та відображення інтерактивних дашбордів. Завдяки дотриманню цих вимог забезпечується стабільна робота всіх модулів системи, висока швидкість обробки запитів і безперервна взаємодія в реальному часі.

Висновки

Виконання курсового проєкту дало змогу створити високофункціональну веб-платформу для управління проєктами, що забезпечує прозору взаємодію в команді на всіх етапах від планування до моніторингу завдань.

В результаті реалізовано:

* зручну реєстрацію та авторизацію з Firebase Admin SDK;
* модулі управління командами й завданнями з підтримкою асинхронної обробки запитів;
* систему коментування;
* дашборди аналітики для оцінки продуктивності;
* адаптивний інтерфейс, який однаково комфортно працює на десктопі й мобільних пристроях.

Також особливу увагу приділено обробці помилок і логуванню, що гарантує стабільність роботи під час пікових навантажень, а також масштабованості завдяки делегуванню очікуваних функцій зовнішнім сервісам. Проведене комплексне тестування ключових модулів підтвердило коректність функціонування і зручність користувацького досвіду.

Отже, реалізована веб-платформа повністю відповідає поставленим цілям, забезпечуючи ефективний контроль, злагоджену комунікацію та підвищену продуктивність командної роботи.

Перелік посилань

1. lemon.school. Управління IT проектами - як це відбувається на приктиці. URL: https://lemon.school/blog/upravlinnya-it-proyektamy-yak-tse-vidbuvayetsya-na-praktytsi <https://help.asana.com/s/article/navigating-asana?language=en_US> https://www.atlassian.com/software/jira/guides/advanced-roadmaps/overview#how-to-get-started
2. Wikipedia. Управління проектами. URL: https://uk.wikipedia.org/wiki/Управління\_проєктами
3. stfalcon. 5 фаз життєвого циклу управління проектами. URL: https://stfalcon.com/uk/blog/post/project-management-life-cycle
4. dou.ua. Процес моделювання предметної області. URL: https://dou.ua/forums/topic/42366/
5. trello.com. Що таке Trello. URL: https://trello.com/uk/tour
6. [trello.com](https://trello.com/uk/guide/trello-101). URL: <https://trello.com/uk/guide/trello-101>
7. [help.asana.com](https://help.asana.com/s/article/navigating-asana?language=en_US). URL: <https://help.asana.com/s/article/navigating-asana?language=en_US>
8. atlassian.com. URL: https://www.atlassian.com/software/jira/guides/advanced-roadmaps/overview#how-to-get-started
9. Wikipedia. Діаграма прецедентів. URL: https://uk.wikipedia.org/wiki/Діаграма\_прецедентів
10. whileweb.com. Веб-дизайн та UX/UI: як створити зручний інтерфейс? URL: https://whileweb.com/uk/blog/veb-dizajn-ta-uxui-yak-stvoriti-zruchnij-interfejs/
11. dashthis.com. Drag and Drop Widgets to your Web Dashboard. URL: https://dashthis.com/blog/drag-and-drop-widgets-to-your-web-dashboard/
12. wikipedia. Моделі баз даних. URL: [https://uk.wikipedia.org/wiki/Моделі\_баз\_даних](https://uk.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D0%B5%D0%BB%D1%96_%D0%B1%D0%B0%D0%B7_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85)
13. dou.ua. Типи баз даних: особливості, відмінності та приклади. URL: https://dou.ua/lenta/articles/types-of-databases/
14. dou.ua. Багатосторінкові та односторінкові додатки, їх переваги та недоліки. URL: https://dou.ua/forums/topic/25444/
15. wikipedia. Firebase. URL: https://uk.wikipedia.org/wiki/Firebase
16. wikipedia. PostreSQL. URL: https://uk.wikipedia.org/wiki/PostgreSQL
17. wikipedia. MVC. URL: https://uk.wikipedia.org/wiki/Модель-вид-контролер
18. deltahost.ua. Типи мережевих протоколів. URL: https://deltahost.ua/ua/tipi-merezhevix-protokoliv-i-ih-priznachennya-http-ip-ssh-ftp-pop3-mac.html?srsltid=AfmBOoqVox-BaFVMI\_hpbSWDDhiesjRaO9xcWNdbCXcnNvKyJhgyoWP5

**ДОДАТКИ**

Додаток А

Програмні коди

Лістинг App.js:

import React, { useContext, useEffect, useCallback } from "react";

import { BrowserRouter, Route, Routes } from "react-router-dom";

import { HashRouter } from 'react-router-dom';

import AppRouter from "./components/Routers/AppRouter";

import { observer } from "mobx-react-lite";

import { RootStoreContext } from "./store/RootStoreProvider";

import NavBar from "./components/NavBar/NavBar";

import LandingPage from "./components/LandingPage/LandingPage";

import { ColorModeContext, useMode} from "./theme";

import { ThemeProvider, CssBaseline, CircularProgress, Box } from "@mui/material";

import { Box as MuiBox } from "@mui/system";

import "./reset.css";

import TaskForm from './components/TaskForm/TaskForm';

import MobileBottomMenu from './components/MobileBottomMenu';

import { useMediaQuery } from 'react-responsive';

import { socketService } from './services/socketService';

const App = observer(() => {

const [theme, colorMode, globalStyles] = useMode();

const { userStore, taskStore } = useContext(RootStoreContext);

const isMobile = useMediaQuery({ maxWidth: 768 });

useEffect(() => {

const fetchInitialData = async () => {

try {

await userStore.checkAuth();

} catch (error) {

console.error('Error checking auth:', error);

}

};

fetchInitialData();

}, []);

useEffect(() => {

let isMounted = true;

let timeoutId = null;

const loadData = async () => {

if (userStore.isAuth && isMounted) {

try {

await taskStore.loadTasks();

} catch (error) {

console.error('Error loading tasks:', error);

}

}

};

if (userStore.isAuth) {

timeoutId = setTimeout(loadData, 500);

}

return () => {

isMounted = false;

if (timeoutId) {

clearTimeout(timeoutId);

}

};

}, [userStore.isAuth]);

useEffect(() => {

let socketTimeoutId = null;

const initializeSocket = () => {

if (userStore.isAuth && !socketService.connected) {

socketTimeoutId = setTimeout(() => {

socketService.connect();

}, 1000);

}

};

initializeSocket();

return () => {

if (socketTimeoutId) {

clearTimeout(socketTimeoutId);

}

socketService.disconnect();

};

}, [userStore.isAuth]);

if (taskStore.loading) {

return (

<MuiBox

display="flex"

justifyContent="center"

alignItems="center"

minHeight="100vh"

>

<CircularProgress />

</MuiBox>

);

}

return (

<HashRouter>

<ColorModeContext.Provider value={colorMode}>

<ThemeProvider theme={theme}>

<CssBaseline />

{globalStyles}

<div className="app">

<main className="content">

{userStore.isAuth && <NavBar />}

<Box

sx={{

maxWidth: "1920px",

margin: "0 auto",

padding: {}

}}

>

<Routes>

{userStore.isAuth ? (

<Route path="/\*" element={<AppRouter />} />

) : (

<Route path="/\*" element={<LandingPage />} />

)}

</Routes>

</Box>

{userStore.isAuth && isMobile && <MobileBottomMenu />}

{userStore.isAuth && <TaskForm />}

</main>

</div>

</ThemeProvider>

</ColorModeContext.Provider>

</HashRouter>

);

});

export default App;

Лістинг UserStore.js;

перерендуваться

import { makeAutoObservable } from "mobx";

import { getAuth, onAuthStateChanged, signOut } from "firebase/auth";

import { getOrsaveUserInDatabase, getUserFromDatabase, checkUserByEmail } from "../http/userApi";

import { loginUserWithEmailAndPassFireBase, registerUserWithEmailAndPassFireBase } from "../utils/fireBase/authFireBaseService";

export default class UserStore {

constructor(){

this.rootStore = null;

this.\_isAuth = false

this.\_user = {}

this.\_isLoading = false

makeAutoObservable(this)

}

setIsAuth(bool){

this.\_isAuth = bool

}

setUser(user){

if (user) {

this.\_user = user;

localStorage.setItem("role", user.role);

} else {

this.\_user = {};

localStorage.removeItem("role");

}

}

setLoading(bool){

this.\_isLoading = bool

}

setRootStore(rootStore) {

this.rootStore = rootStore;

}

get isAuth(){

return this.\_isAuth

}

get user(){

return this.\_user

}

get isLoading(){

return this.\_isLoading

}

clearStore() {

this.\_isAuth = false;

this.\_user = {};

this.\_isLoading = false;

localStorage.removeItem('token');

localStorage.removeItem('role');

}

async login(email, password){

try {

const userFireBase = await loginUserWithEmailAndPassFireBase(email, password);

localStorage.setItem('token', userFireBase.token);

const response = await getUserFromDatabase();

if (response && response.user) {

this.setUser(response.user);

this.setIsAuth(true);

localStorage.setItem('role', response.user.role);

} else {

throw new Error('Failed to get user data');

}

} catch (error) {

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

throw error;

}

}

async registration(email, password){

try {

const userFireBase = await registerUserWithEmailAndPassFireBase(email, password);

localStorage.setItem('token', userFireBase.token);

const response = await this.getUserFromDB(userFireBase.email, userFireBase.token);

} catch (error){

throw error;

}

}

async getUserFromDB(email, token, userData){

try {

const response = await getOrsaveUserInDatabase(email, token, userData);

localStorage.setItem('token', token);

this.setUser(response.user);

this.setIsAuth(true);

} catch (error){

throw error

}

}

async logout(){

try {

const auth = getAuth();

signOut(auth)

.then(() => {

// Sign-out successful.

this.rootStore.clearAllStores();

}).catch((error) => {

console.log(error.message)

}

);

} catch (e) {

console.log(e.response?.message);

}

}

async checkAuth(){

const auth = getAuth();

try {

const userFireBase = await new Promise((resolve, reject) => {

const unsubscribe = onAuthStateChanged(auth, (user) => {

unsubscribe();

resolve(user);

}, reject);

});

if (userFireBase) {

const token = await userFireBase.getIdToken();

localStorage.setItem('token', token);

const response = await getUserFromDatabase(token);

if(response && response.user){

this.setUser(response.user);

this.setIsAuth(true);

localStorage.setItem('role', response.user.role);

} else {

this.setUser(null);

this.setIsAuth(false);

}

} else {

this.setUser(null);

this.setIsAuth(false);

}

} catch (error) {

console.error("Error checking authentication:", error);

this.setUser(null);

this.setIsAuth(false);

}

}

async checkUserExists(email) {

try {

const data = await checkUserByEmail(email);

return data.exists;

} catch (error) {

console.error('Error checking user:', error);

return false;

}

}

}

Лістинг SocketService.js: import { io } from 'socket.io-client';

import { makeAutoObservable } from 'mobx';

import userStore from '../store/UserStore';

class SocketService {

socket = null;

connected = false;

notifications = [];

onlineUsers = new Set();

reconnectAttempts = 0;

maxReconnectAttempts = 3;

connectionTimeout = null;

constructor() {

makeAutoObservable(this);

}

connect() {

if (this.socket?.connected || this.connectionTimeout) {

return;

}

this.disconnect();

this.connectionTimeout = setTimeout(() => {

this.connectionTimeout = null;

if (this.reconnectAttempts >= this.maxReconnectAttempts) {

console.log('Max reconnection attempts reached');

return;

}

}, 5000);

this.socket = io(process.env.REACT\_APP\_API\_URL || '<http://localhost:9000>', {

withCredentials: true,

transports: ['websocket'],

auth: {

userId: userStore.user?.uid

},

reconnection: false,

timeout: 5000

});

this.setupSocketListeners();

}

setupSocketListeners() {

if (!this.socket) return;

this.socket.on('connect', () => {

console.log('Connected to socket server');

this.connected = true;

this.reconnectAttempts = 0;

if (this.connectionTimeout) {

clearTimeout(this.connectionTimeout);

this.connectionTimeout = null;

}

});

this.socket.on('disconnect', () => {

console.log('Disconnected from socket server');

this.connected = false;

});

this.socket.on('connect\_error', (error) => {

console.error('Socket connection error:', error);

this.reconnectAttempts++;

if (this.reconnectAttempts >= this.maxReconnectAttempts) {

console.log('Max reconnection attempts reached');

this.disconnect();

}

});

this.socket.on('userStatusChanged', ({ userId, isOnline }) => {

if (isOnline) {

this.onlineUsers.add(userId);

} else {

this.onlineUsers.delete(userId);

}

});

this.socket.on('notification', (notification) => {

if (notification) {

this.notifications.push(notification);

this.showNotification(notification);

}

});

}

disconnect() {

if (this.connectionTimeout) {

clearTimeout(this.connectionTimeout);

this.connectionTimeout = null;

}

if (this.socket) {

this.socket.removeAllListeners();

this.socket.disconnect();

this.socket = null;

this.connected = false;

this.notifications = [];

this.onlineUsers.clear();

this.reconnectAttempts = 0;

}

}

joinTeam(teamId) {

if (this.socket?.connected) {

this.socket.emit('joinTeam', teamId);

}

}

leaveTeam(teamId) {

if (this.socket?.connected) {

this.socket.emit('leaveTeam', teamId);

}

}

onTaskCreated(callback) {

if (this.socket?.connected) {

this.socket.on('taskCreated', callback);

}

}

onTaskUpdated(callback) {

if (this.socket?.connected) {

this.socket.on('taskUpdated', callback);

}

}

onCommentAdded(callback) {

if (this.socket?.connected) {

this.socket.on('commentAdded', callback);

}

}

emitNewTask(data) {

if (this.socket?.connected) {

this.socket.emit('newTask', data);

}

}

emitUpdateTask(data) {

if (this.socket?.connected) {

this.socket.emit('updateTask', data);

}

}

emitNewComment(data) {

if (this.socket?.connected) {

this.socket.emit('newComment', data);

}

}

isUserOnline(userId) {

return this.onlineUsers.has(userId);

}

showNotification(notification) {

if ('Notification' in window) {

if (Notification.permission === 'granted') {

new Notification(notification.title, {

body: notification.message,

icon: '/logo192.png'

});

} else if (Notification.permission !== 'denied') {

Notification.requestPermission().then(permission => {

if (permission === 'granted') {

new Notification(notification.title, {

body: notification.message,

icon: '/logo192.png'

});

}

});

}

}

}

markNotificationAsRead(notificationId) {

const notification = this.notifications.find(n => n.id === notificationId);

if (notification) {

notification.read = true;

}

}

getUnreadNotifications() {

return this.notifications.filter(n => !n.read);

}

}

export const socketService = new SocketService();

Лістинг AuthPopup.jsx;

import React, { useContext, useState } from 'react';

import Button from '@mui/material/Button';

import TextField from '@mui/material/TextField';

import Dialog from '@mui/material/Dialog';

import DialogActions from '@mui/material/DialogActions';

import DialogContent from '@mui/material/DialogContent';

import DialogContentText from '@mui/material/DialogContentText';

import DialogTitle from '@mui/material/DialogTitle';

import Box from '@mui/material/Box';

import IconButton from '@mui/material/IconButton';

import Fade from '@mui/material/Fade';

import CloseIcon from '@mui/icons-material/Close';

import InputAdornment from '@mui/material/InputAdornment';

import EmailIcon from '@mui/icons-material/Email';

import LockIcon from '@mui/icons-material/Lock';

import Typography from '@mui/material/Typography';

import GoogleIcon from '@mui/icons-material/Google';

import GitHubIcon from '@mui/icons-material/GitHub';

import { RootStoreContext } from "../../../store/RootStoreProvider";

import { handleAuthGoogle, handleAuthGitHub, checkAuth } from '../../../utils/fireBase/authFireBaseService';

import { getErrorMessageFireBasePopup } from '../../../utils/errors/errorsFirebase';

const AuthPopup = ({open, setOpen}) => {

const [logOrReg, setLogOrReg] = useState(true);

const [errorMessage, setErrorMessage] = useState(null);

const {userStore} = useContext(RootStoreContext);

const handleRegisterUserWithEmailAndPass = async (email, password) => {

try {

await checkAuth(userStore);

if (logOrReg) {

await userStore.login(email, password);

} else {

await userStore.registration(email, password);

}

handleClose();

} catch (error) {

const errorMessage = getErrorMessageFireBasePopup(error);

setErrorMessage(errorMessage);

}

};

const handleLogRegWithGoogle = async () => {

try {

await checkAuth(userStore);

const userFireBase = await handleAuthGoogle();

if (userFireBase) {

await userStore.getUserFromDB(userFireBase.email, userFireBase.token, userFireBase.userData);

handleClose();

} else {

console.log('No user data from Firebase');

}

} catch (error) {

const errorMessage = getErrorMessageFireBasePopup(error);

setErrorMessage(errorMessage);

}

};

const handleLogRegWithGitHub = async () => {

try {

await checkAuth(userStore);

const userFireBase = await handleAuthGitHub();

if (userFireBase) {

await userStore.getUserFromDB(userFireBase.email, userFireBase.uid, userFireBase.userData);

handleClose();

} else {

console.log('No user data from Firebase');

}

} catch (error) {

const errorMessage = getErrorMessageFireBasePopup(error);

setErrorMessage(errorMessage);

}

};

const handleLogOrReg = () => {

setLogOrReg(prevLogOrReg => !prevLogOrReg);

};

const handleClose = () => {

setOpen(false);

setErrorMessage(null);

};

return (

<React.Fragment>

<Dialog

open={open}

onClose={handleClose}

TransitionComponent={Fade}

PaperProps={{

component: 'form',

sx: {

borderRadius: 4,

background: 'linear-gradient(135deg, #e3f2fd 0%, #bbdefb 100%)',

boxShadow: '0 8px 32px 0 rgba(31, 38, 135, 0.37)',

p: { xs: 1, md: 2 },

minWidth: { xs: 320, sm: 400 },

maxWidth: 500,

position: 'relative',

},

onSubmit: (event) => {

event.preventDefault();

const formData = new FormData(event.currentTarget);

const formJson = Object.fromEntries(formData.entries());

const email = formJson.email;

const password = formJson.password;

handleRegisterUserWithEmailAndPass(email, password);

},

}}

>

{/\* Іконка закриття \*/}

<IconButton

onClick={handleClose}

sx={{ position: 'absolute', right: 8, top: 8, zIndex: 10, color: 'grey.700' }}

size="large"

>

<CloseIcon />

</IconButton>

<Box>

<DialogTitle sx={{ textAlign: 'center', fontWeight: 700, fontSize: { xs: 22, md: 26 }, letterSpacing: 1, mb: 1 }}>

{logOrReg ? "Вхід" : "Реєстрація"}

</DialogTitle>

</Box>

<Box sx={{display: "flex", flexDirection: { xs: 'column', sm: 'row' }, alignItems: 'stretch', justifyContent: 'center', gap: 2, p: { xs: 1, md: 2 }}}>

<Box sx={{maxWidth: "350px", flex: 1, minWidth: 240}}>

<DialogContent>

<DialogContentText sx={{color: "text.main", textAlign: 'center', mb: 2}}>

{logOrReg ? "Щоб отримати всі можливості, увійдіть у свій акаунт" :

"Щоб отримати всі можливості, зареєструйте акаунт"}

</DialogContentText>

<TextField

autoFocus

required

margin="dense"

id="name"

name="email"

label="Email"

type="email"

fullWidth

variant="outlined"

InputProps={{

startAdornment: (

<InputAdornment position="start">

<EmailIcon color="primary" />

</InputAdornment>

),

}}

sx={{ mb: 2, background: 'rgba(255,255,255,0.7)', borderRadius: 2 }}

/>

<TextField

required

margin="dense"

id="password"

name="password"

label="Пароль"

type="password"

fullWidth

variant="outlined"

InputProps={{

startAdornment: (

<InputAdornment position="start">

<LockIcon color="primary" />

</InputAdornment>

),

}}

sx={{ mb: 2, background: 'rgba(255,255,255,0.7)', borderRadius: 2 }}

/>

</DialogContent>

<DialogActions sx={{justifyContent: "center", width: "100%",display: "flex", flexDirection: "column", alignItems: "center", gap: "10px"}}>

{errorMessage && (

<Box sx={{color: "red", fontWeight: 600, animation: 'shake 0.3s', borderRadius: 1, px: 1, py: 0.5, background: 'rgba(255,0,0,0.07)'}}>

{errorMessage}

</Box>

)}

<Button type="submit" fullWidth variant="contained" sx={{

background: 'linear-gradient(90deg, #1976d2 0%, #42a5f5 100%)',

color: 'white',

fontWeight: 600,

borderRadius: 2,

boxShadow: '0 2px 8px 0 rgba(33,150,243,0.15)',

py: 1.2,

fontSize: 18,

mt: 1,

'&:hover': {

background: 'linear-gradient(90deg, #1565c0 0%, #1976d2 100%)',

}

}}>{logOrReg ? "Увійти" : "Зареєструватися"}</Button>

{/\* Роздільник \*/}

<Box sx={{ display: 'flex', alignItems: 'center', my: 0 }}>

<Box sx={{ flex: 1, height: 1, background: 'rgba(0,0,0,0.1)' }} />

<Typography sx={{ mx: 2, color: 'text.secondary', fontSize: 14 }}>Або</Typography>

<Box sx={{ flex: 1, height: 1, background: 'rgba(0,0,0,0.1)' }} />

</Box>

<Button onClick={handleLogOrReg} fullWidth sx={{ color: '#1976d2', fontWeight: 500, textTransform: 'none', fontSize: 16 }}>

{logOrReg ? "Зареєструватися" : "Увійти"}

</Button>

</DialogActions>

</Box>

<Box sx={{width: { xs: '100%', sm: '200px' }, display: "flex", flexDirection: "column", justifyContent: "center", alignItems: 'center', textAlign: "center", gap: 2, py: 2}}>

<DialogContent sx={{display: "flex", flexDirection: "column", gap: "10px", alignItems: 'center', p: 0}}>

<DialogContentText sx={{color: "text.main", mb: 1}}>

{logOrReg ? "Увійти через" :

"Реєстрація через"}

</DialogContentText>

<Box sx={{ display: 'flex', gap: 2, justifyContent: 'center' }}>

<IconButton

size="large"

color="primary"

onClick={handleLogRegWithGoogle}

sx={{ background: 'white', boxShadow: '0 2px 8px 0 rgba(33,150,243,0.10)', '&:hover': { background: '#e3f2fd' } }}

>

<GoogleIcon />

</IconButton>

<IconButton

size="large"

color="primary"

onClick={handleLogRegWithGitHub}

sx={{ background: 'white', boxShadow: '0 2px 8px 0 rgba(33,150,243,0.10)', '&:hover': { background: '#e3f2fd' } }}

>

<GitHubIcon />

</IconButton>

</Box>

</DialogContent>

</Box>

</Box>

<style>{`

@keyframes shake {

0% { transform: translateX(0); }

20% { transform: translateX(-8px); }

40% { transform: translateX(8px); }

60% { transform: translateX(-8px); }

80% { transform: translateX(8px); }

100% { transform: translateX(0); }

}

`}</style>

</Dialog>

</React.Fragment>

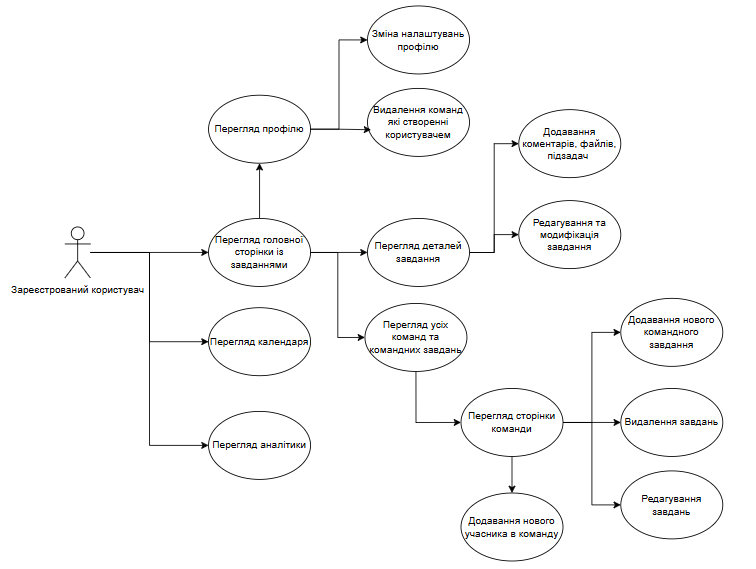
);

};

export default AuthPopup;

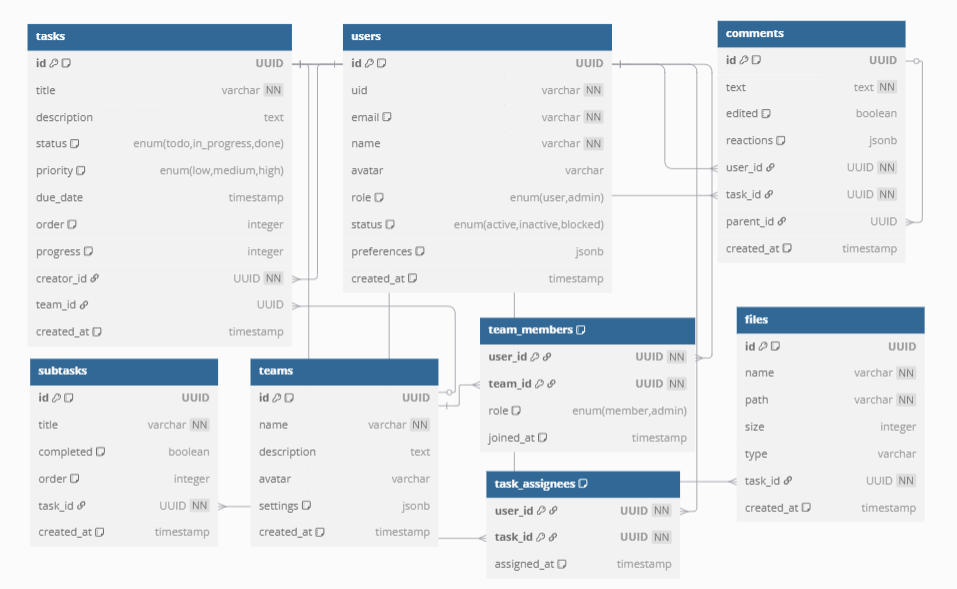
Додаток Б

**Діаграма варіантів використання вебсайта системи для управління завданнями для зареєстрованих користувачів**



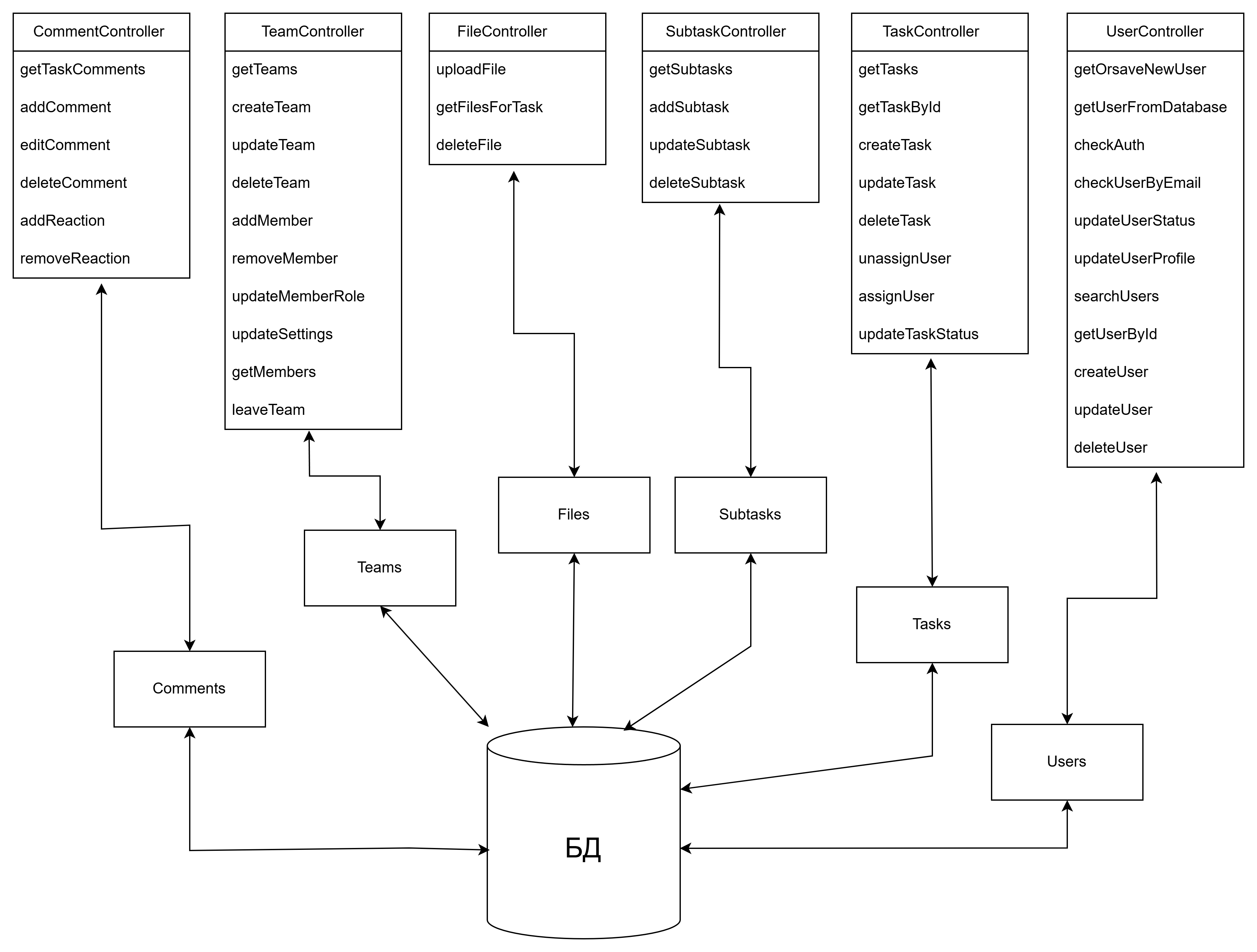
Додаток В

Даталогічна модель БД вебсайта системи управління завданнями



Додаток Г

**Схема взаємозв’язку контролерів та ORM-моделей системи управління завданнями**



Додаток Д

Презентаційний матеріал



