Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «МАГІСТР»

РОЗРОБКА ТА ДОСЛІДЖЕННЯ ІНТЕРФЕЙСУ ДЛЯ ІНТЕРАКТИВНОГО УПРАВЛІННЯ ПРОЕКТАМИ З KANBAN-ДОШКОЮ

DEVELOPMENT AND RESEARCH OF AN INTERFACE FOR INTERACTIVE PROJECT MANAGEMENT WITH A KANBAN BOARD

спеціальність 121 «Інженерія програмного забезпечення»
освітня програма «Інженерія програмного забезпечення»

Виконав: здобувач вищої освіти
групи ІПЗм-21
Шворак П. А.
Керівник:
к.т.н., доцент
Ліщина Н. М.

Кваліфікаційну роботу
допущено до захисту
«___» _____ 20__ р.
Гарант освітньої програми:
к.т.н., доцент Суринович О. М.
_____

Луцьк – 2025 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *інженерії програмного забезпечення*
Ступінь вищої освіти *магістр*
Галузь знань: *12 «Інформаційні технології»*
Спеціальність: *121 «Інженерія програмного забезпечення»*
Освітня програма: «*Інженерія програмного забезпечення»*

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____

«___» _____ 202__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА

ДРУГОГО (МАГІСТЕРСЬКОГО) РІВНЯ ВИЩОЇ ОСВІТИ

Швораку Павлу Анатолійовичу

1. Тема кваліфікаційної роботи: Розробка та дослідження інтерфейсу для інтерактивного управління проектами з kanban-дошкою
Керівник роботи: Ліщина Наталія Миколаївна, доцент, к.т.н.
затверджені наказом закладу вищої освіти від «29» березня 2025 року № 190/01-02
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи: 04 грудня 2025 р.
3. Вихідні дані до роботи *технічне та програмне забезпечення ЕОМ*
4. Зміст розрахунково-пояснювальної записки: *аналіз проблематики прогнозування попиту та вибір методів дослідження, обґрунтування технологій і реалізацію вебсистеми на основі регресійного аналізу, експериментальне дослідження результативності програмного забезпечення*
5. Перелік графічного матеріалу 15 рисунків, 5 таблиці, 6 лістинги коду

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз проблеми за темою роботи та постановка завдань дослідження	*Ліщина Н. М.*		
Теоретичне дослідження та практична реалізація	*Ліщина Н. М.*		
Експериментальне дослідження системи	*Ліщина Н. М.*		
Нормоконтроль	*Повстяна Ю. С.*		
Гарант ОП	*Андрущак І. Є.*		
Показник запозичень тексту	___%		
Академічна доброчесність	*Ліщина Н. М.*		

7. Дата видачі завдання *«02 квітня 2025 р.*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи магістра	Строк виконання етапів роботи	Примітка
1	Провести огляд літературних джерел по темі кваліфікаційної роботи	02.05.2025	
2	Провести аналіз загальної проблеми і вибір напрямків дослідження	24.09.2025	
3	Розробити функціональну модель та архітектуру системи	01.11.2025	
4	Описати засоби розробки об'єкта проектування	19.11.2025	
5	Практична реалізація об'єкта проектування	26.11.2025	
6	Розробити методику для проведення експерименту	05.11.2025	
7	Провести аналіз результатів експерименту	15.11.2025	
8	Здача чистового варіанту кваліфікаційної роботи на кафедру	04.12.2025	

Здобувач вищої освіти _____ Шворак П. А.

Керівник кваліфікаційної роботи _____ Ліщина Н. М.

АНОТАЦІЯ

Шворак П. А. Розробка інтерфейсу для інтерактивного управління проектами з Kanban-дошкою. Рукопис.

Кваліфікаційна робота магістра ОП «Інженерія програмного забезпечення» спеціальності 121 Інженерія програмного забезпечення. Луцький національний технічний університет. Луцьк, 2025.

Кваліфікаційна робота магістра складається зі вступу, трьох розділів, висновків, списку використаних джерел.

Кваліфікаційна робота магістра присвячена розробці інтерфейсу для інтерактивного управління проектами з Kanban-дошкою.

Основна частина містить детальний опис всіх етапів розробки, також містить частини коду, знімки екрану, опис процесу створення бази даних, розробки сторінок та їх стилізації з використанням JavaScript, TypeScript, React, Vite та SCSS.

Результати розробки є загальними і можуть бути використані при розробці веб-сайтів усіх типів.

Ключові слова: Kanban-дошка, управління проєктами, інтерфейс, веб-система, програмне забезпечення, React, TypeScript, JavaScript, SCSS.

ABSTRACT

Shvorak P. A. Development and research of an interface for interactive project management with a kanban-board. Manuscript.

Master's thesis of the Educational Program «Software Engineering», specialty 121 «Software Engineering». Lutsk National Technical University. Lutsk, 2025.

The master's thesis consists of an introduction, three chapters, conclusions, and a list of references (according to the structure of qualification works approved by the department).

The master's thesis is devoted to the development of an interface for interactive project management using a Kanban board. The work focuses on creating a modern software system that enables visualization of task states, convenient organization of workflows, and interactive manipulation of task cards within a multi-board environment.

The main part contains a detailed description of all stages of development and includes code fragments, screenshots, an explanation of the interface design process, and styling of application components using JavaScript, TypeScript, React, Vite, and SCSS.

The results of the development are general in nature and can be applied in the creation of web systems of various types.

Keywords: Kanban board, project management, interface, web system, software, React, TypeScript, JavaScript, SCSS.

ЗМІСТ

ВСТУП

Актуальність теми. У сучасному світі цифрові технології стають основою більшості бізнес-процесів, а ефективне управління проєктами є ключовим чинником успішної діяльності компаній.

З розвитком гнучких методологій, таких як Agile, Scrum і Kanban, все більшої популярності набувають інтерактивні інструменти, які дозволяють командам планувати, координувати й контролювати виконання завдань у зручній візуальній формі. Зокрема, Kanban-дошки дають змогу відображати стан кожного завдання, спостерігати за прогресом і своєчасно реагувати на затримки у виконанні. Актуальність теми дослідження зумовлена потребою у створенні сучасних веб-застосунків, що поєднують функціональність управління проєктами з інтуїтивним, естетичним і адаптивним інтерфейсом. У багатьох існуючих рішеннях (Trello, Jira, Asana) користувачі стикаються з проблемами перевантаженості інтерфейсу, складності навігації або обмеженої персоналізації. Тому важливим завданням є розробка системи, яка забезпечить інтерактивність, зручність і швидку взаємодію користувача із завданнями без втрати продуктивності.

Розвиток технологій фронтенду (React, Vue.js, Angular), поява систем реального часу (WebSocket, Firebase) та інструментів для спільної роботи створюють передумови для побудови таких рішень. Окрім технічних аспектів, велике значення має грамотний дизайн користувацького досвіду (UX) та інтерфейсу (UI), що забезпечує інтуїтивне сприйняття функціональності, чітку ієрархію елементів і приємну візуальну взаємодію.

Об’єктом дослідження є процес розробки інтерфейсу систем управління проєктами з Kanban-дошкою.

Предметом дослідження є методи, технології та інструменти побудови інтерактивних інтерфейсів для управління проєктами в реальному часі.

Метою дослідження є створення та всебічне дослідження інтерфейсу для інтерактивного управління проєктами, який поєднує простоту використання,

високу ефективність та привабливий візуальний стиль. Такий інтерфейс має забезпечувати інтуїтивну взаємодію користувачів, підтримувати швидку організацію робочих процесів, візуалізацію стану задач та зручну координацію дій у командному середовищі. Дослідження спрямоване на формування сучасного, адаптивного та функціонального рішення, здатного оптимізувати командну роботу, покращити продуктивність і підвищити якість управління проєктами на основі Kanban-підходу.

Завдання дослідження полягають у необхідності:

– виконати системний аналіз предметної області та визначити функціональні, технічні та UX-вимоги до інтерактивної Kanban-системи;

– проведення експериментального тестування продуктивності інтерфейсу;

– спроектувати користувацький інтерфейс, який мінімізує когнітивне навантаження та забезпечує ефективну візуалізацію задач і робочих процесів;

– створити архітектурну модель веб-застосунку на основі сучасних фреймворків (React/Vue/Angular) із підтримкою динамічної взаємодії та drag-and-drop механізмів;

– реалізації механізмів локального збереження даних, фільтрації та пошуку задач;

– сформулювати рекомендації щодо подальшого вдосконалення системи та аналіз перспектив масштабування розробленого рішення.

Практична цінність роботи полягає у можливості використання розробленої системи для організації командної діяльності в IT-компаніях, навчальних установах та інших сферах, де необхідне планування і контроль задач. Розроблений веб-застосунок може виступати як самостійний інструмент або частина комплексного програмного забезпечення, має високу продуктивність, простоту розширення та можливість інтеграції з сервісами реального часу.

Наукова новизна роботи полягає у поєднанні адаптивного інтерфейсу, сучасних анімаційних технологій Framer Motion, локального збереження даних і

оптимізованих механізмів drag-and-drop для створення високопродуктивної Kanban-системи. Запропоновані підходи забезпечують стабільну швидкодію при масштабуванні і перевершують наявні аналоги за показниками відгуку, інтуїтивності та ефективності взаємодії користувача зі системою.

Запропоновані у кваліфікаційній роботі програмні рішення інтерактивної Kanban-системи мають високий ступінь готовності до практичного впровадження, що підтверджено їх експериментальною перевіркою у реальному веб-середовищі. Основні положення та результати дослідження пройшли апробацію шляхом представлення тези на V Міжнародній науково-практичній конференції «Research in Science, Technology and Economics», яка відбулася у місті Люксембург, Люксембург, 10-12 грудня 2025 року [1].

РОЗДІЛ 1
АНАЛІЗ ПРОБЛЕМАТИКИ ІНТЕРФЕЙСІВ ДЛЯ УПРАВЛІННЯ ПРОЕКТАМИ ТА ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ

1.1 Огляд і аналіз предметної області проблеми, результатів існуючих теоретичних та експериментальних досліджень

У сучасному інформаційному суспільстві, де цифрові технології визначають практично всі аспекти бізнес-процесів, а IT-проєкти стають багаторівневими і багатокомпонентними, питання ефективного управління задачами виходить на передній план. Традиційні методи управління – ті, що передбачають жорстку послідовність етапів, – дедалі частіше не відповідають умовам швидкоплинного ринку, змін вимог або частих переглядів пріоритетів. У такому контексті гнучкі методології проявляють себе як не просто альтернатива, а як необхідність: серед них особливо виділяється Kanban-методологія – гнучкий, візуальний та адаптивний підхід до організації потоку робіт.

Методологія Kanban [2] бере свій початок із виробничих систем компанії Toyota у середині XX століття. Інженери Тайїті Оно (Taiichi Ohno) і Сігео Сінго (Shigeo Shingo) впровадили систему «Just-in-Time», мета якої полягала у оптимізації виробничого потоку, мінімізації запасів і зменшенні втрат. Вони застосували візуальні картки (канбан-карти), які сигналізували про потребу в переміщенні, заміні або поповненні запасів, що дозволяло координувати виробничі операції без надлишкових запасів чи затримок. Особливістю цієї системи була чітка регламентація черг, обмеження незавершених робіт (WIP) і принцип «витягування» задач лише тоді, коли для них є ресурс – усе це створювало основу для гнучкої системи управління потоком.

Згодом поняття Kanban вийшло за межі фізичного виробництва. У 2000-х роках Девід Дж. Андерсон (David J. Anderson) адаптував філософію Kanban до розробки програмного забезпечення [3]. У його фундаментальній праці з управління IT-проєктами сформульовані базові принципи: візуалізація потоку завдань, WIP-обмеження, управління потоком на базі метрик, публічне

відображення станів задач, плавні зміни процесів через Kaizen, вимірювання Cycle Time / Lead Time / Throughput [4]. Ці ідеї перетворили Kanban на гнучку, живу систему, здатну адаптуватися під специфіку IT-команд, зберігаючи дисципліну та передбачуваність процесів.

Подальший розвиток концепції представлений роботами таких фахівців, як Клаус Леопольд (Klaus Leopold) – автор концепції багаторівневого Kanban (Flight Levels), яка дозволяє масштабувати Kanban на рівень організації, синхронізуючи роботу окремих команд з бізнес-цілями; Юрген Аппело (Jurgen Appelo) – теоретик agile-організації та lean-мислення, який досліджує самоорганізацію команд, адаптивне управління та стійкі системи продуктивності [5].

Крім того, теоретичну обґрунтованість Kanban забезпечує класична теорія масового обслуговування та черг. Основний математичний інструмент – закон Літтла, сформульований американським математиком Джоном Д. К. Літтлом (John D. C. Little) – описує кореляцію між середнім часом обходження задачі, кількістю задач у системі та пропускною здатністю. Ця формула дає змогу прогнозувати ефективність, оцінювати WIP-обмеження, планувати навантаження і оптимізувати потоки, що надає Kanban науково обґрунтований характер, а не лише «світогляд» або набір практик.

Kanban – це не лише управлінська методологія, а й контекст, у якому інтерфейс має вирішальне значення. Саме якість UI/UX визначає, наскільки легко команда сприйме карткову систему, чи буде вона використовувати її постійно, чи повернеться до старих практик. У цьому сенсі вагомий внесок зробив Дональд Норман (Donald Norman) – його праці з когнітивного дизайну, зокрема принципи видимості, зворотного зв'язку, простоти і відповідності ментальним моделям користувача, стали базою для сучасного UX-дизайну. Якоб Нільсен (Jakob Nielsen) та інші дослідники в галузі usability розкрили, як складні, перевантажені інтерфейси ведуть до помилок, втоми, зниження продуктивності, що особливо критично для систем з великою кількістю елементів та частими операціями – саме таким є Kanban-інтерфейс.

У сучасних дослідженнях інтерфейсних систем Kanban підкреслюється кілька ключових вимог: мінімізація зайвих дій, інтуїтивна навігація, чітка візуальна ієрархія, адаптивність під різні розміри екранів, плавні анімації для drag-and-drop, миттєвий зворотний зв'язок при зміні станів, а також можливість швидкого фільтрування і пошуку задач. Саме такі характеристики дозволяють реалізувати потенціал Kanban на практиці, підвищуючи комфорт, швидкість і точність роботи.

У 2020-х роках активне розвиток отримали веб-інструменти та SPA-застосунки, які реалізують Kanban: фронтенд-фреймворки на кшталт React, Vue, Angular надали можливість створювати реактивні, компонентні UI з підтримкою drag-and-drop, динамічного рендерингу, адаптивності під мобільні пристрої [6]. Також стали доступні бібліотеки для анімації та покращення UX – Framer Motion, GSAP, що дозволяють реалізувати плавні переходи, анімоване переміщення карток, візуальні підказки під час перетягування, покращують сприйняття та зменшують навантаження на користувача.

Однією з головних проблем, на які звертають увагу наукові та практичні дослідження, є продуктивність при великій кількості задач. Якщо інтерфейс не оптимізований, при сотнях чи тисячах елементів можливі лаги, «гальма», зростання затримок, падіння FPS, що робить систему непридатною для масштабних проєктів. У відповідь на це дослідники пропонують: оптимізовану компонентну архітектуру, віртуалізацію списків, системи кешування, мінімізацію рендерингу, lazy-loading, efficient diff-алгоритми, а також продумане управління даними (LocalStorage, IndexedDB, Web Storage API).

Крім того, для корпоративних або розподілених команд досліджується інтеграція з бекендом або realtime-сервісами через WebSocket, Firebase, SignalR чи інші технології, що забезпечують синхронізацію змін між користувачами, зберігання історії змін, контроль версій задач, автентифікацію, рольове управління. Це означає, що сучасний Kanban-інструмент – це не просто UI-дошка, а повноцінна платформа з багаторівневою архітектурою, що поєднує фронтенд, бекенд, бази даних, UI/UX, аналітику та DevOps [7].

Серед найбільш успішних та масових реалізацій Kanban – такі системи як Trello, Jira Software, Azure DevOps Boards, ClickUp, Notion, Taiga, Kanboard та ін. – вони демонструють, як теоретичні принципи Kanban перетворюються на продукт, що використовують тисячі команд по всьому світу. Аналіз їх функціональних можливостей показує сильні і слабкі сторони: Trello пропонує простоту та зручність, але обмежену аналітику; Jira надає потужні засоби для складних проєктів, проте має високу криву навчання та перевантажений інтерфейс; Azure DevOps інтегрується з CI/CD, але має корпоративну орієнтацію; Notion/ClickUp пропонують універсальність, але можуть втрачати продуктивність при масштабі.

Наукові дослідження, що оцінюють ці системи, вказують, що переваги Kanban реалізуються лише тоді, коли UI/UX відповідає вимогам: швидкий відгук, мінімальне когнітивне навантаження, можливість кастомізації, чітке відображення станів, прозорий потік задач, ефективне управління пріоритетами та WIP-обмеженнями, масштабованість під великі команди. У роботах, що досліджують реальну ефективність, зазначено: зменшення часу циклу на 20–40%, підвищення передбачуваності доставки, зменшення кількості контекстних перемикань і помилок, підвищення задоволеності користувачів системою.

Поглиблений аналіз теоретичних основ, історичних витоків, математичного моделювання, UX-досліджень і технологічного контексту підтверджує, що предметна область Kanban – це комплексна, багатовимірна дисципліна, яка поєднує управління потоками задач, аналітику, дизайн інтерфейсів та розробку веб-систем. Наукове та практичне вивчення Kanban продовжується: розробники, дослідники, компанії – усі прагнуть покращити як продуктивність команд, так і комфорт і ефективність взаємодії.

1.2 Огляд і аналіз методів та засобів розробки інтерфейсу для інтерактивного управління проєктами для вирішення проблеми дослідження

Розробка інтерфейсу для інтерактивного управління проєктами, зокрема на основі Kanban-підходу, вимагає застосування сучасних методів проектування користувацького досвіду та інструментів веб-розробки, які забезпечують динамічність, адаптивність, масштабованість і високу продуктивність системи. Наукові дослідження у сфері людино-комп'ютерної взаємодії (HCI) свідчать, що ефективний інтерфейс є критичним чинником успішності будь-якої інтерактивної платформи, оскільки визначає швидкість взаємодії користувача з інформацією, знижує когнітивне навантаження та забезпечує точність і передбачуваність робочих процесів.

У контексті Kanban-систем проектування інтерфейсу передбачає застосування методів UX-дизайну, які включають аналіз користувацьких сценаріїв, побудову інформаційної архітектури, визначення ієрархії елементів та оптимізацію візуальної структури дошки. Одним із найпоширеніших методів є проєктування на основі когнітивних моделей, зокрема моделі GOMS та теорії ментальних моделей Нормана, що дозволяє оцінити складність взаємодії та мінімізувати кількість дій, необхідних для досягнення користувацьких цілей. Значну роль відіграє принцип візуального групування, контрастності та просторової організації, який забезпечує швидке сприйняття ключових елементів інтерфейсу, таких як статуси задач, WIP-обмеження, маркери блокування та індикатори критичного навантаження.

У технічній площині розробка інтерфейсу Kanban-системи здебільшого базується на сучасних фронтенд-фреймворках, які реалізують компонентно-орієнтовану архітектуру та підтримують реактивне оновлення даних. Найпоширенішими інструментами є React, Vue.js та Angular. React, зокрема, забезпечує високу продуктивність завдяки використанню віртуального DOM, ефективному керуванню станом та можливості створювати повторно

використовувані компоненти, що значно спрощує побудову складних інтерфейсів, таких як Kanban-дошка з drag-and-drop механізмами. Vue.js вирізняється низьким порогом входу та зручною реактивністю, що робить його оптимальним для швидкого прототипування. Angular забезпечує строгість архітектури та масштабованість, що є важливим для корпоративних систем з великою кількістю користувачів. Для реалізації інтерактивних елементів Kanban-дошки, таких як перетягування карток між колонками, найчастіше застосовуються бібліотеки React Beautiful DnD, SortableJS або Dragula, які базуються на стандартах HTML5 Drag and Drop API та забезпечують коректну поведінку елементів при переміщенні. У системах, де необхідна висока точність візуалізації та складні анімації, використовують спеціалізовані інструменти, такі як Framer Motion або GreenSock, які дозволяють підвищити якість взаємодії без втрати продуктивності. Важливою складовою інтерактивного інтерфейсу є забезпечення синхронності даних між різними користувачами, що працюють із системою одночасно. Для цього застосовуються WebSocket-протоколи, технології Firebase Realtime Database, Socket.IO або GraphQL Subscriptions, які гарантують миттєве оновлення статусу задач та запобігають конфліктам доступу. Крім того, сучасні Kanban-платформи інтегруються з REST API або GraphQL-сервісами для забезпечення обміну даними зі сторонніми системами, зокрема CRM, системами аналітики, сховищами файлів чи DevOps-інструментами.

У рамках методів забезпечення якості інтерфейсу особливого поширення набули тестування юзабіліті, дослідження реакцій користувачів, A/B-тестування інтерфейсних елементів, а також метрики оцінювання уваги, серед яких eye-tracking, аналіз теплових карт та вимірювання часу до виконання задачі (task completion time). Ці підходи дозволяють обґрунтовано оцінити ефективність візуальних рішень та виявити інтерфейсні бар'єри ще на етапі розробки. У сфері дизайну інтерфейсів для складних систем проектного управління також застосовуються дизайн-системи, такі як Material Design, Ant Design, Carbon Design System або Fluent UI, які забезпечують стандартизовані компоненти,

чітку візуальну структуру та єдині принципи взаємодії, що дозволяє уникнути хаотичності та невідповідності елементів у масштабних застосунках. У наукових дослідженнях доведено, що застосування дизайн-систем суттєво зменшує час розробки, покращує доступність інтерфейсу та забезпечує узгодженість користувацького досвіду, що є особливо актуальним для Kanban-середовищ з великою кількістю візуальних компонентів. При створенні інтерфейсів для управління проєктами також критично важливими є засоби адаптивної верстки – CSS Grid, Flexbox, сучасні підходи Mobile First, а також фреймворки Bootstrap або Tailwind CSS, які гарантують коректне відображення Kanban-дошки на мобільних пристроях, планшетах та широкоформатних екранах. Адаптивність відіграє значну роль у процесі командної роботи, оскільки користувачі часто взаємодіють із системою в різних умовах, включаючи віддалений доступ. Окрему увагу в дослідженні методів розробки слід приділити аспектам доступності (accessibility), оскільки сучасні стандарти WCAG передбачають можливість роботи зі складними інтерфейсами для людей із порушеннями зору, моторики або когнітивних функцій. Використання ARIA-атрибутів, коректна семантика HTML, достатній контраст та підтримка клавіатурної навігації є обов'язковими для якісного інтерфейсу Kanban.

Узагальнюючи, можна стверджувати, що сучасні методи та засоби розробки інтерфейсу для інтерактивного управління проєктами формують комплексний міждисциплінарний підхід, який об'єднує принципи UX-дизайну, веб-інженерії, теорії когнітивного навантаження та сучасних технологій інтерактивності. Їхнє систематичне застосування створює передумови для формування ефективного, інтуїтивного та продуктивного інтерфейсу Kanban-системи, що забезпечує високу якість взаємодії, оптимізацію робочих процесів і підтримку стратегічних цілей проектної діяльності.

1.3 Постановка завдання на кваліфікаційну роботу магістра

Враховуючи результати огляду предметної області, проведений аналіз теоретичних і практичних досліджень у сфері інтерактивного управління проєктами, а також дослідження сучасних методів і засобів розробки користувацьких інтерфейсів, постає необхідність створення високоефективної веб-орієнтованої Kanban-системи, здатної забезпечити оптимальну візуалізацію робочих процесів, інтуїтивність взаємодії, оперативну синхронізацію даних та підтримку сучасних вимог до адаптивності, масштабованості й продуктивності.

Проблема дослідження полягає у тому, що існуючі інструменти, хоча й пропонують широкий функціонал, не завжди забезпечують баланс між простотою інтерфейсу, когнітивною зручністю користувачів та технічною гнучкістю.

Відтак метою кваліфікаційної роботи є обґрунтована розробка архітектури, дизайну та прототипу інтерактивного інтерфейсу Kanban-дошки, що відповідає принципам теорії черг, сучасним UX-підходам і можливостям технологій фронтенд-розробки.

У межах роботи необхідно поєднати наукові принципи побудови потоково-орієнтованих систем із практичними методами проектування, створивши програмне рішення, орієнтоване на підвищення ефективності роботи користувачів, зниження когнітивного навантаження та покращення керованості процесів.

На основі цього в межах кваліфікаційної роботи формуються такі конкретні цілі, яких потрібно досягти:

– виконати системний аналіз предметної області та визначити функціональні, технічні та UX-вимоги до інтерактивної Kanban-системи;

– проведення експериментального тестування продуктивності інтерфейсу;

– спроектувати користувацький інтерфейс, який мінімізує когнітивне навантаження та забезпечує ефективну візуалізацію задач і робочих процесів;

– створити архітектурну модель веб-застосунку на основі сучасних фреймворків (React/Vue/Angular) із підтримкою динамічної взаємодії та drag-and-drop механізмів;

– реалізації механізмів локального збереження даних, фільтрації та пошуку задач;

– сформулювати рекомендації щодо подальшого вдосконалення системи та аналіз перспектив масштабування розробленого рішення.

Висновки до розділу 1

Узагальнюючи результати огляду предметної області та аналізу методів і засобів розробки інтерфейсу для інтерактивного управління проєктами, можна дійти висновку, що сучасні Kanban-системи виступають комплексним інструментом, який поєднує науково обгрунтовані принципи організації робочих процесів, когнітивні моделі взаємодії користувача з інформаційним середовищем та інноваційні технології веб-розробки.

Проведений аналіз показав, що ефективність Kanban підкріплена фундаментальними теоретичними положеннями теорії масового обслуговування, зокрема законом Літтла, який визначає взаємозв'язок між обсягом незавершеної роботи, пропускною здатністю та часом виконання задач, а також емпіричними дослідженнями, що підтверджують здатність методології зменшувати час циклу, підвищувати передбачуваність процесів і покращувати командну взаємодію.

Водночас доведено, що значущу роль у досягненні цих переваг відіграє якість інтерфейсу, оскільки саме інтерфейс визначає когнітивне навантаження, швидкість орієнтації користувачів та точність виконання дій. Аналіз сучасних методів та інструментів інтерфейсної розробки засвідчив, що побудова високоефективної Kanban-системи можлива лише за умови інтеграції UX-підходів, що враховують психологічні особливості сприйняття інформації, разом із передовими веб-технологіями, які забезпечують реактивність,

масштабованість, адаптивність та синхронність даних у режимі реального часу.

Використання компонентних фреймворків (React, Vue.js, Angular), бібліотек для drag-and-drop механізмів, дизайн-систем та протоколів WebSocket дозволяє створити інтерфейс, здатний не лише візуалізувати складні робочі процеси, а й робити це з високою точністю, швидкістю та стійкістю. Таким чином, проведений теоретичний і методологічний аналіз підтверджує доцільність і актуальність створення інтерактивної Kanban-платформи, що спирається на науково-обгрунтовані принципи управління потоками робіт та сучасні технології інтерфейсного проектування.

Сформована база знань дає можливість визначити вимоги до майбутньої системи, окреслити технічні та архітектурні рішення, а також забезпечити методичне підґрунтя для реалізації програмного продукту, який буде ефективним, інтуїтивним і здатним оптимізувати процеси управління проєктами в умовах високої динаміки й складності сучасного інформаційного середовища.

РОЗДІЛ 2
ТЕОРЕТИЧНЕ ДОСЛІДЖЕННЯ ТА ПРАКТИЧНА РЕАЛІЗАЦІЯ ІНТЕРФЕЙСУ ДЛЯ ІНТЕРАКТИВНОГО УПРАВЛІННЯ ПРОЕКТАМИ

2.1. Обґрунтування вибору шляхів, технологій, алгоритмів і засобів вирішення поставленого завдання

Проблему створення веб-системи для інтерактивного управління проєктами можна було вирішити кількома способами:

– за допомогою конструкторів інтерфейсів;
– використовуючи CMS-системи;
– застосовуючи готові SaaS-сервіси управління задачами;
– використовуючи бібліотеки;
– розробляючи застосунок повністю власним кодом.

Було розглянуто найпоширеніші хмарні сервіси створення веб-додатків, такі як Trello та Notion.

Trello [8] – популярний онлайн-сервіс управління проєктами, який використовує Kanban-підхід. Він підтримує drag&drop [9], має мобільні додатки та мінімалістичний інтерфейс. Проте можливість глибокої кастомізації дуже обмежена, а створення нестандартних функцій вимагає використання Power-Ups, більшість яких є платними. Попередній вигляд інтерфейсу наведено на рисунку 2.1.

Рисунок 2.1 – Сайт Trello

Notion – це універсальний інструмент для організації інформації, який поєднує в собі можливості нотатника, баз даних, таск-менеджера та платформи для спільної роботи. Його головна ідея – дати користувачу максимально гнучкий простір, у якому можна створити практично будь-яку структуру даних: від простих нотаток до складних систем управління проєктами. Notion працює за принципом блоків, де кожен фрагмент інформації – це окремий елемент (текст, список, база даних, таблиця, дошка, вкладення тощо), який можна вільно комбінувати, перетягувати, вкладати один в один та організовувати так, як зручно [10]. Вигляд сервісу показано на рисунку 2.2.

Однією з ключових переваг Notion є бази даних, які дозволяють створювати гнучкі системи: списки, таблиці, канбан-дошки, календарі чи галереї. Кожен запис має властивості – теги, дати, формули, статуси, зв'язки з іншими базами – що дає змогу будувати складні взаємозв'язки й автоматизувати частину роботи. Завдяки цьому Notion часто застосовують для управління проектами, контент-планування, CRM-систем, відстеження задач чи навчання. Усе можна кастомізувати під власні потреби без кодування.

Ще одна важлива особливість Notion – можливість спільної роботи. Користувачі можуть редагувати сторінки одночасно, залишати коментарі, згадувати колег, використовувати шаблони та обмінюватися даними. Платформа підходить як для особистих нотаток, так і для командної документації. Notion підтримує вкладення файлів, вбудовування сторонніх сервісів, інтеграції з іншими інструментами та синхронізацію між пристроями. У результаті він стає не просто нотатником, а повноцінною робочою екосистемою.

Популярність Notion пояснюється його простотою, гнучкістю та можливістю створити єдине робоче середовище замість десятків окремих додатків. Користувач сам вирішує, як буде виглядати його робочий простір, і може змінювати його в будь-який момент без обмежень. Саме це робить Notion універсальним інструментом для студентів, команд, бізнесів і творчих людей, яким потрібна система, що підлаштовується під їхні задачі, а не навпаки.

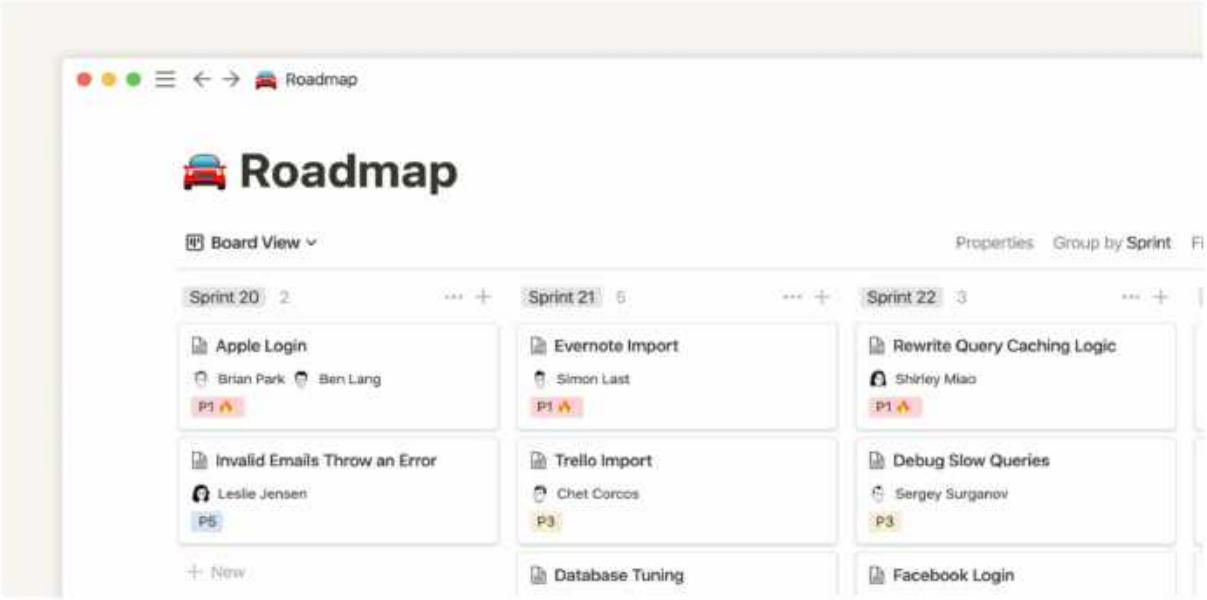

Рисунок 2.2 – Сайт Notion

Серед CMS було розглянуто WordPress та Joomla, однак обидві платформи погано підходять для інтерактивної роботи з великою кількістю змін у режимі реального часу. До того ж, реалізація drag&drop логіки в рамках CMS ускладнюється архітектурними обмеженнями, а підтримка компонентної структури – недостатня.

Було проведено аналіз веб-фреймворків, що можуть бути використані для проектування веб-інтерфейсу, такі як React та Vue.

React – це популярна JavaScript-бібліотека для створення інтерфейсів користувача, розроблена компанією Meta, яка дає змогу будувати динамічні, масштабовані та ефективні веб-додатки на основі компонентного підходу [11]. Основна ідея React полягає в тому, що інтерфейс розбивається на незалежні, багаторазові компоненти, кожен з яких відповідає за свій стан і логіку відображення, що спрощує розробку та підтримку навіть великих проєктів. React використовує віртуальний DOM – легку копію реального DOM, яка дає змогу оновлювати сторінку максимально швидко, порівнюючи зміни та застосовуючи їх точково, а не перерендерюючи всю сторінку. Компоненти можуть бути функціональними або класовими, але сучасний підхід ґрунтується на функціональних із використанням хуків – спеціальних функцій, таких як

useState чи useEffect, що дозволяють працювати зі станом і життєвими циклами без класів. React також підтримує одностороннє управління даними, коли інформація передається від батьківських компонентів до дочірніх через пропси, що робить структуру даних передбачуваною. Завдяки JSX – синтаксису, який змішує JavaScript із HTML-подібними конструкціями – розробка інтерфейсів стає інтуїтивною. React легко інтегрується з різними інструментами, такими як React Router для навігації чи Redux/Zustand/Recoil для складного керування станом, а також працює в рамках екосистеми, що охоплює веб, мобільну розробку (React Native), серверний рендеринг (Next.js) та інші підходи [12]. Його гнучкість, швидкість і масштабованість роблять React одним із найпопулярніших рішень для створення сучасних інтерфейсів у веб-розробці.

З переваг React можна виділити:

– компонентна архітектура, що дозволяє розділити логіку на окремі модулі;

– використання віртуального DOM для підвищення продуктивності;

– широкий набір бібліотек для анімацій, drag&drop та станів;

– велика спільнота та універсальність.

Vue – це прогресивний фреймворк для створення інтерфейсів користувача, який поєднує простоту, гнучкість і високу продуктивність. Його розробив Еван Ю. як легку й доступну альтернативу важчим фреймворкам, і Vue швидко став одним із найпопулярніших інструментів фронтенд-розробки. Головна ідея Vue полягає в реактивності: коли змінюються дані, інтерфейс автоматично оновлюється без додаткової ручної логіки, що значно спрощує розробку. Компонентний підхід, на якому заснований Vue, дає змогу розбивати застосунок на незалежні частини з власними шаблонами, логікою та стилями, а файли типу *.vue* об'єднують ці частини в одному місці, роблячи код максимально читабельним і структурованим. Синтаксис Vue виглядає інтуїтивно – зв'язування даних, директиви та шаблони нагадують HTML, тому поріг входу дуже низький навіть для початківців. У Vue 3 з'явилася Composition API – сучасний спосіб організації логіки, який дозволяє групувати

функціональні частини за призначенням, а не за приналежністю до компонента, що робить код більш масштабованим і гнучким у великих проєктах. Водночас зберігається класичний Options API, який зручний для навчання та невеликих проєктів.

Vue добре поєднується з офіційними інструментами: Vue Router для маршрутизації, Pinia або Vuex для керування станом, а також Vite як швидкий сучасний збирач. Завдяки своїй легкості, природній реактивності, зрозумілому синтаксису та сильній екосистемі Vue є чудовим вибором як для простих інтерфейсів, так і для великих масштабованих веб-додатків, забезпечуючи розробнику приємний та ефективний досвід роботи.З переваг Vue можна виділити:

– простіший синтаксис для новачків;
– двостороннє зв'язування даних.

Порівнявши фреймворки та враховуючи потребу у високій продуктивності, інтерактивності та модульності, було обрано саме React у поєднанні з TypeScript.

TypeScript – це надбудова над JavaScript, створена компанією Microsoft, яка додає до мови статичну типізацію та низку інструментів, що роблять розробку надійнішою, передбачуванішою та зручнішою, особливо у великих проєктах. Головна ідея TypeScript полягає в тому, що він дозволяє розробнику явно визначати типи змінних, параметрів, функцій та об'єктів, завдяки чому помилки виявляються ще на етапі компіляції, а не в браузері під час виконання. Це суттєво зменшує кількість непередбачуваних ситуацій і робить роботу з кодом більш контрольованою. При цьому TypeScript не замінює JavaScript: будь-який валідний JavaScript-код є валідним TypeScript-кодом, а результат роботи TypeScript-компілятора – це звичайний JavaScript, який може виконуватися в будь-якому середовищі [13].

Типова система TypeScript дає змогу описувати складні структури даних через інтерфейси й типи, розширювати їх, створювати узагальнені (generic) типи та застосовувати суворі правила до об'єктів, функцій і класів. Завдяки

цьому код стає самодокументованим: редактор розуміє типи й може підказувати автозаповнення, виявляти помилки, пропонувати рефакторинг і значно полегшувати навігацію. Особливо важливим є те, що TypeScript чудово працює з сучасними фреймворками – React, Vue, Angular – і активно використовується у великих командних проєктах, де необхідно впевнено контролювати структуру даних і логіку.

Крім типів, TypeScript підтримує сучасні можливості ECMAScript, класи, модулі, декоратори та інші концепції, які можуть бути недоступні у звичайному JavaScript без додаткового транспілювання. Він також дозволяє поступово впроваджувати типізацію: проєкт може містити як строго типізовані частини, так і нестрогі, що спрощує міграцію зі стилю «чистого» JavaScript на більш структурований підхід. У результаті TypeScript робить код більш безпечним, зручним для підтримки й розширення, а також покращує взаємодію в команді, забезпечуючи чіткі контракти та зрозумілі API між модулями, компонентами чи сервісами.

З переваг TypeScript можна виділити:

- статична типізація;
- покращений автокомпліт у редакторах;
- самодокументованість коду.
- легше масштабування великих проєктів;
- покращена навігація та рефакторинг;
- сумісність із javascript;
- сильна підтримка фреймворків;
- підтримка сучасних можливостей js;
- безпечніший код;
- покращена командна робота;
- інтерфейси та generics;
- підтримка ООП - підходу.

Було також розглянуто інструменти збирання проєктів: Webpack, Parcel та Vite.

З переваг Vite [14] можна виділити:

– дуже швидке збирання і запуск проєкту;
– підтримка гарячого оновлення модулів;
– зручна інтеграція з TypeScript та React.

Для стилізації інтерфейсу було обрано SCSS, який надає можливість створювати змінні тем, міксини, вкладені структури та підтримує масштабування.

SCSS (Sassy CSS) – це розширений синтаксис мови стилів Sass, який є сумісним із CSS і водночас значно розширює його функціональність. Він дозволяє організовувати стилі більш структуровано, логічно та ефективно, зменшуючи дублювання коду та спрощуючи підтримку великих проєктів. Основною перевагою SCSS є використання змінних, що дозволяє зберігати повторювані значення, такі як кольори, розміри чи шрифти, в одному місці та легко змінювати їх у майбутньому. Вкладеність селекторів дає змогу писати стилі у зручній ієрархічній формі, яка відповідає структурі HTML, завдяки чому код стає читабельнішим та легшим у навігації. Міксіни забезпечують можливість створювати багаторазові набори стилів, які можна підключати в будь-яких селекторах, що значно зменшує обсяг повторюваного коду. Розширення (extends) дозволяє успадковувати стилі одного елемента іншим, що теж сприяє оптимізації. Крім того, SCSS підтримує вбудовані та користувацькі функції, які дають змогу виконувати математичні операції, змінювати кольори та працювати з різними параметрами більш гнучко. Усе це робить SCSS зручним інструментом для побудови модульної та масштабованої системи стилів. Ще одна важлива властивість – можливість розділяти стилі на частини та імпортувати їх у головні файли, що полегшує роботу з великими кодовими базами. SCSS не виконується браузером безпосередньо, тому потребує компіляції в звичайний CSS, що здійснюється за допомогою таких інструментів, як Sass, Webpack, Gulp, Vite та різні плагіни. Завдяки своїй гнучкості та розширеним можливостям SCSS широко використовується в сучасній фронтенд-розробці, особливо у великих проєктах, і є одним із найпоширеніших

рішень для написання підтримуваних і організованих стилів [15].

Для мінімальної анімації було вирішено використати бібліотеку Framer Motion.

Framer Motion – це сучасна високорівнева бібліотека для створення анімацій у React-застосунках, яка поєднує простоту декларативного синтаксису з високою продуктивністю та широкими можливостями для реалізації анімацій взаємодії користувача. Розроблена командою Framer, ця технологія набуває все більшої популярності у веб-розробці, оскільки забезпечує плавні, природні та оптимізовані візуальні переходи, що підвищують якість користувацького досвіду.

Основою бібліотеки є об'єкт motion, який містить анімовані елементи (motion.div, motion.button, motion.span тощо), що повністю повторюють структуру стандартних HTML-компонентів, але дозволяють керувати їхнім рухом, прозорістю, трансформаціями та іншими властивостями у динаміці. Framer Motion використовує декларативну модель програмування, завдяки чому анімації описуються через властивості компонента, а основні зміни виконуються шляхом оновлення стану, що забезпечує зручність інтеграції з React-архітектурою.

Однією з ключових переваг Framer Motion є підтримка фізично коректних анімацій, що базуються на моделях пружності, інерції, демпфування та кінематичних параметрах руху. Це дозволяє інтерфейсу природно реагувати на дії користувача, уникати різких і неприродних переходів, характерних для класичних CSS-анімацій. Фреймворк також забезпечує GPU-прискорення, що значно зменшує навантаження на CPU і дозволяє підтримувати стабільну частоту кадрів навіть на мобільних пристроях.

Бібліотека має низку важливих функціональних можливостей, зокрема:

– Variants – гнучкий механізм оголошення множинних станів анімації з подальшим переключенням між ними;

– Animate / Initial / Exit – автоматизація анімацій появи та зникнення елементів, зручна в умовах динамічних інтерфейсів;

– Layout animations – плавні просторові переходи між змінами DOM-структури, що особливо корисно при перетягуванні, додаванні та сортуванні елементів;

– Gestures – підтримка жестів (hover, tap, drag), необхідних для реалізації інтерактивних UI-рішень;

– Scroll-based animations – можливість візуальних ефектів, синхронізованих із прокручуванням сторінки.

Framer Motion перевершує традиційні підходи до анімації веб-інтерфейсів, такі як @keyframes у CSS або бібліотеки типу GSAP, завдяки повній інтеграції з React, прозорому керуванню станами та можливості анімації змін компонентного розташування. У порівнянні з CSS-анімаціями фреймворк пропонує більшу контрольованість, передбачуваність та адаптивність, що робить його ідеальним для реалізації динамічних SPA-застосунків.

Застосування Framer Motion є особливо доцільним у системах, де висока плавність взаємодії є ключовим чинником ефективності інтерфейсу. Такі приклади включають дошки управління задачами, інтерфейси для колаборативної роботи, візуалізацію потоків даних, інтерактивні панелі та складні UI-компоненти. Використання анімованого компонента motion.div дозволяє реалізувати плавне перетягування елементів, їхню появу та зникнення, підсвічування під час наведення, акцентування важливих змін стану.

Практична цінність Framer Motion полягає у здатності зменшувати когнітивне навантаження на користувача, формувати очікувану поведінку елементів інтерфейсу та створювати відчуття цілісності системи. Анімація стає не декоративним елементом, а функціональним засобом підсилення користувацької взаємодії: важливі зміни виділяються плавно, а контекст задач зберігається без втрати фокусу уваги.

Таким чином, Framer Motion є потужним інструментом для розробки сучасних високоякісних інтерфейсів. Він забезпечує розробникам доступ до широкого набору анімаційних можливостей, не ускладнюючи при цьому архітектуру застосунку. Стандарти продуктивності, зручна синтаксична модель і

глибока інтеграція з React роблять Framer Motion оптимальним вибором для реалізації інтерактивних SPA-рішень, де важливу роль відіграють плавність, швидкодія та інтуїтивність взаємодії.

Серед редакторів коду було порівняно Visual Studio Code та WebStorm. WebStorm – потужний інструмент з глибокою інтеграцією для JavaScript-проєктів.

Visual Studio Code – легший, безкоштовний редактор з великою бібліотекою розширень, що робить його оптимальним вибором.

З урахуванням вимог до проєкту було обрано WebStorm.

На підставах вище наведеного аналізу вимог та предметної області, було прийнято рішення розробляти сайт з використанням фреймворку React.

Основою розробки сайту методом написання власного коду – є мова програмування.

В ході аналізу і порівняння методів, інструментів та програмного забезпечення було визначено, що оптимальними інструментами та середовищем розробки для виконання поставленого завдання буде поєднання:

- TypeScript;
- React;
- SCSS;
- Vite.

Після встановлення необхідного ПЗ та налаштування середовища було розпочато розробку системи.

2.2 Практична реалізація об'єкта проектування

Практична реалізація проектованої інтерактивної Kanban-системи передбачає побудову повнофункціонального веб-орієнтованого інтерфейсу, який забезпечує ефективну візуалізацію потоків робіт, підтримку взаємодії користувачів у реальному часі, інтуїтивність взаємодії та відповідність сучасним вимогам до UX/UI-дизайну. Основною метою створюваного

програмного продукту є забезпечення можливості структурованого управління завданнями, контролю станів, пріоритетів і етапів виконання, а також швидкої адаптації робочого процесу відповідно до динаміки проєкту.

Основою програмного продукту є односторінковий веб-застосунок, реалізований на базі бібліотеки React із використанням TypeScript. Структура проєкту організована за компонентним принципом: на верхньому рівні розташовано кореневий компонент KanbanBoard, який відповідає за формування загального каркаса інтерфейсу (шапка з фільтрами, набір колонок, тема оформлення). Логіка роботи дошки інкапсульована в окремих кастомних хуках: useKanbanBoard відповідає за керування станом задач, фільтрами, формами та drag-and-drop взаємодією, а useTheme – за вибір і перемикання теми (light/dark). Для кожної колонки створено власний компонент KanbanColumn, а для окремої задачі – компонент картки; форми створення й редагування задач виділені в підкомпоненти, що забезпечує повторне використання та спрощує тестування. Така модульна структура дозволяє чітко розділити візуальне представлення, бізнес-логіку та допоміжні сервіси (збереження в Local Storage), полегшуючи розвиток і підтримку системи.

Базова структура інтерфейсу наведено нижче в лістингу 2.1.

Лістинг 2.1 – Код базової структури

```
<div className={containerClass}>
      <div className="board-header">
      <h1 className="board-title">Kanban Board</h1>
      <div className="board-controls">...</div>
      <div className="board">...</div>
</div>
```

кінець лістингу 2.1

Далі ми створюємо картку задачі. Картка задачі є базовою одиницею інформації на Kanban-дошці та відображає мінімально необхідний набір даних для прийняття рішення користувачем. Структурно картка містить: унікальний

номер задачі (ідентифікатор, наприклад TSK-101), заголовок (короткий опис суті завдання), розширений текстовий опис (деталізація дій, очікуваний результат), блок індикаторів пріоритету та поточного статусу, а також список тегів, які характеризують технологічний або предметний контекст (наприклад, frontend, bug, ui). У верхній частині картки розміщено елементи керування – іконка редагування та видалення задачі, що дає змогу швидко змінювати її вміст або повністю вилучати з дошки. Візуально картка оформлена у вигляді «плитки» з м'якими кутами, тінню та чіткою внутрішньою ієрархією: номер і заголовок мають найбільший візуальний акцент, нижче подано опис, а внизу – капсули з тегами, статусом та пріоритетом. Це відповідає принципам UX-дизайну, зменшує когнітивне навантаження та дозволяє користувачу швидко «сканувати» великий масив задач.

Вигляд цього компоненту наведено нижче на рисунку 2.3.

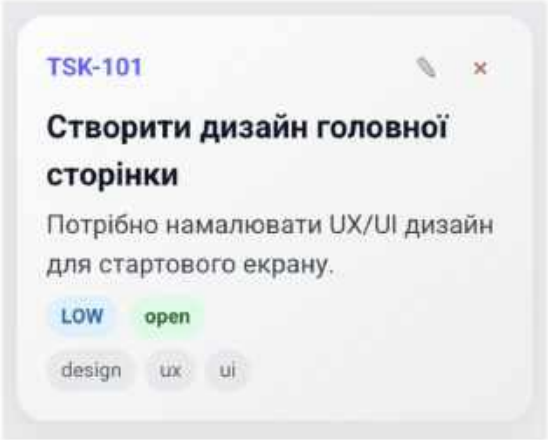

Рисунок 2.3 – Вигляд компоненту картки задачі

Редагування задачі реалізовано за принципом повторного використання тієї ж форми, що застосовується для створення нової задачі, але в режимі заповнення полів існуючими даними. При натисканні на іконку «редагувати» у картці система передає в стан форми поточне значення всіх атрибутів задачі (номер, заголовок, опис, пріоритет, статус, теги), після чого у відповідній колонці відкривається форма з уже заповненими полями. Користувач може змінити будь-який параметр; валідація перевіряє коректність критичних полів

(наприклад, непорожній заголовок, унікальність номера задачі).

Після натискання кнопки «Зберегти» зміни оновлюють відповідний запис у внутрішньому стані застосунку, а також синхронізуються з локальним сховищем браузера. У результаті реалізовано єдиний уніфікований механізм створення й редагування, що спрощує код і підвищує передбачуваність інтерфейсу для користувача.

Вигляд вікна редагування наведено нижче на рисунку 2.4.

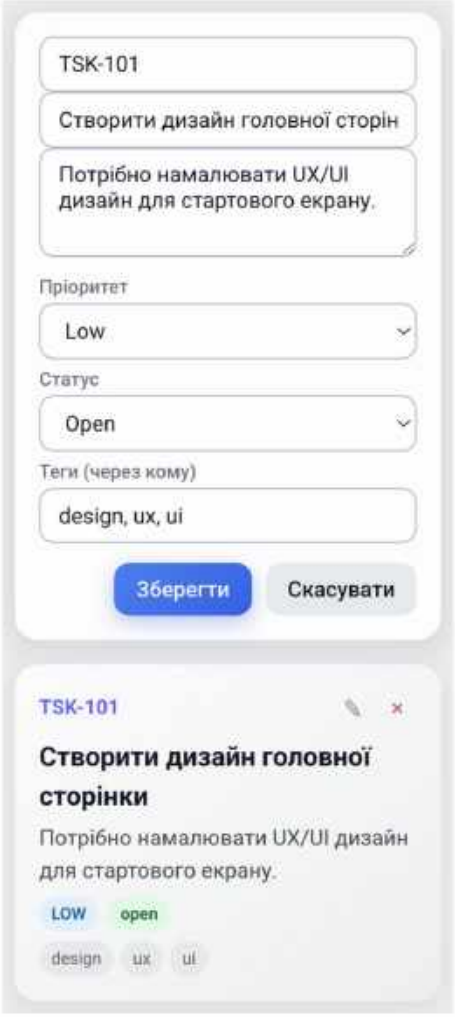

Рисунок 2.4 – Вигляд вікна редагування картки задачі

Функція редагування картки задачі наведено нижче в лістингу 2.2.

Лістинг 2.2 – Код функції редагування

```
const openEditForm = (taskId: string) => {
      const task = tasks.find((t) => t.id === taskId);
      if (!task) return;

      setActiveForm(
            {
                  mode: "edit",
                  columnId: task.columnId,
                  taskId: task.id,
            }
      );

      setFormDraft(
            {
                  columnId: task.columnId,
                  number: task.number,
                  title: task.title,
                  description: task.description,
                  priority: task.priority,
                  status: task.status,
                  tagsText: task.tags.join(", "),
            }
      );

      setFormErrors(null);
};
```

кінець лістингу 2.2

Після створення картки задачі ми створюємо колонки в яких будуть роміщатись наші картки. Кожна колонка Kanban-дошки представляє окремий етап життєвого циклу задачі (Backlog, To Do, In Progress, Review, Done) і реалізована як самостійний компонент, що приймає через пропси метадані колонки та список задач, які до неї належать. У шапці колонки відображається її назва та кнопка з іконкою «+» для відкриття форми додавання нової задачі саме в цю колонку. Всередині колонки послідовно виводиться список карток, які відповідають поточному фільтру та сортуванню.

Компонент колонки також обробляє події drag-and-drop (події знаходження над колонкою, відведення курсора, скидання задачі) та змінює

стилі при наведенні перетягуваної задачі, що дає користувачеві візуальний зворотний зв'язок про те, куди саме може бути переміщена задача. Структура колонок є гнучкою: при необхідності в майбутньому може бути розширена додатковими атрибутами, наприклад лімітами WIP або показниками завантаженості.

Вигляд колонки наведено нижче на рисунку 2.5.

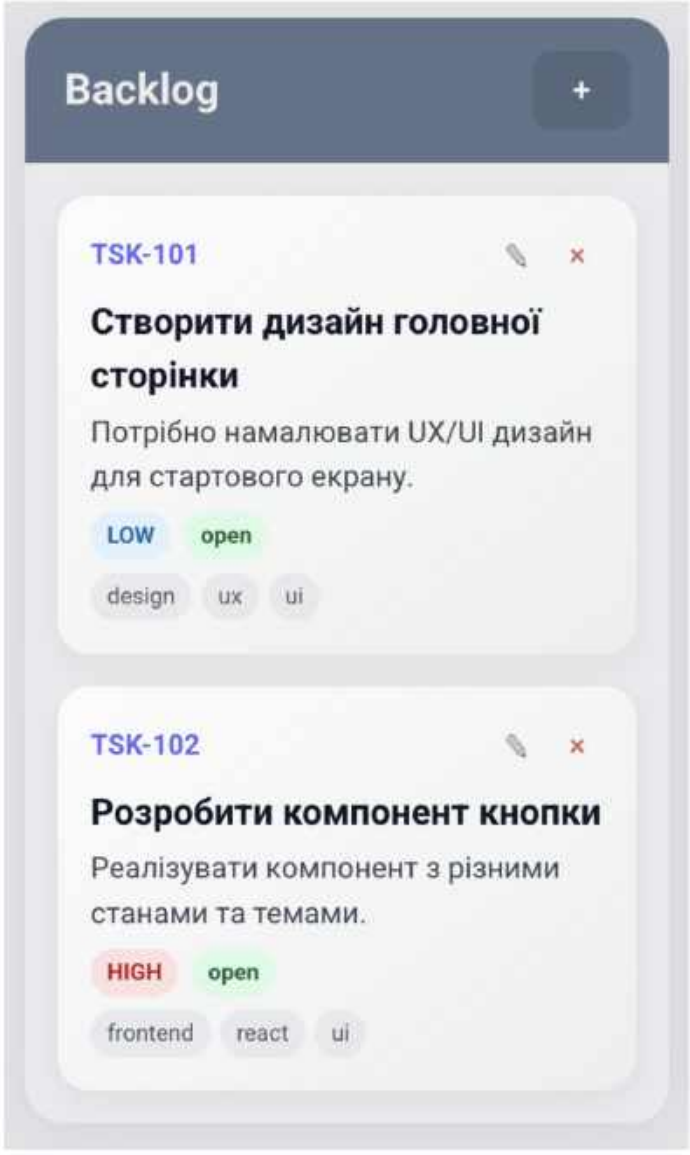

Рисунок 2.5 – Вигляд колонки

Приклад використання motion наведено нижче в лістингу 2.3.

Лістинг 2.3 – Код використання motion

```
<motion.div
      className={columnClass}
      onDragOver={handleDragOver}
      onDrop={onDrop}
      onDragEnter={onDragEnterColumn}
      onDragLeave={onDragLeaveColumn}
      initial={{
            opacity: 0,
            y: 10
      }}
      animate={{
            opacity: 1, y: 0
      }}
      transition={{
            duration: 0.2
      }}>
      ...
</motion.div>
```

кінець лістингу 2.3

Виведення всіх карток, які приймає компонент колонки, і виводить за допомогою map наведено нижче в лістингу 2.4.

Лістинг 2.4 – Виведення карток в колонці

```
{tasks.map((task) => (
      <TaskCard
            key={task.id}
            task={task}
            columnId={column.id}
            onDragStart={onTaskDragStart}
            onDragEnd={onTaskDragEnd}
            onDelete={onTaskDelete}
            onEdit={onTaskEdit}
            isDragging={draggingTaskId === task.id} />
))}
```

кінець лістингу 2.4

Механізм перетягування картки задачі реалізовано за допомогою інтеграції з бібліотекою drag-and-drop (на рівні компонентів карток і колонок) та обробки відповідних подій у хуку useKanbanBoard. Коли користувач починає перетягувати картку, система запам'ятовує ідентифікатор задачі (draggingTaskId) та її вихідну колонку. Під час переміщення над іншими колонками спрацьовують обробники onDragEnterColumn/onDragLeaveColumn, які змінюють візуальний стан колонок (наприклад, підсвічують область можливого скидання). При відпусканні кнопки миші у цільовій колонці викликається метод handleDropOnColumn, який оновлює атрибут columnId у відповідній задачі, за необхідності змінює її статус і перераховує списки задач у вихідній та цільовій колонках. Завдяки такій логіці користувач отримує природний сценарій взаємодії з дошкою, а всі зміни відразу відображаються в інтерфейсі та зберігаються в стані й локальному сховищі.

Вигляд картки, яку переносять нижче на рисунку 2.6.

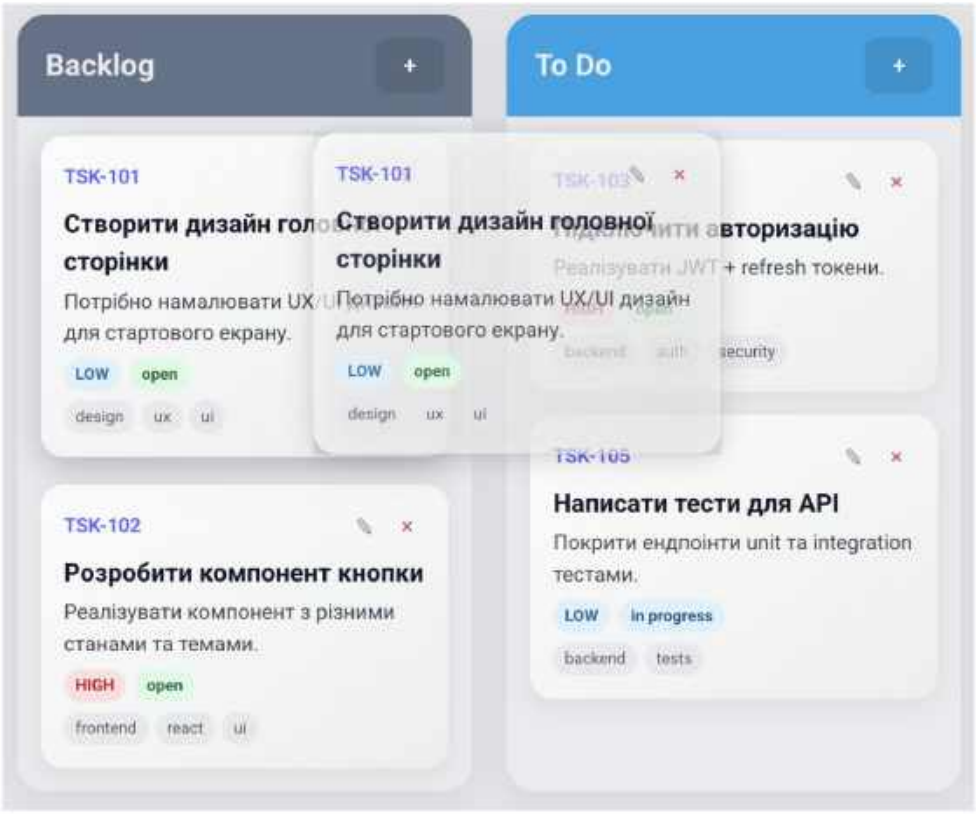

Рисунок 2.6 – Вигляд ефекту drag&drop

Перенесену картку, ми можемо спостерігати вже в іншій колонці, це показано на рисунку 2.7.

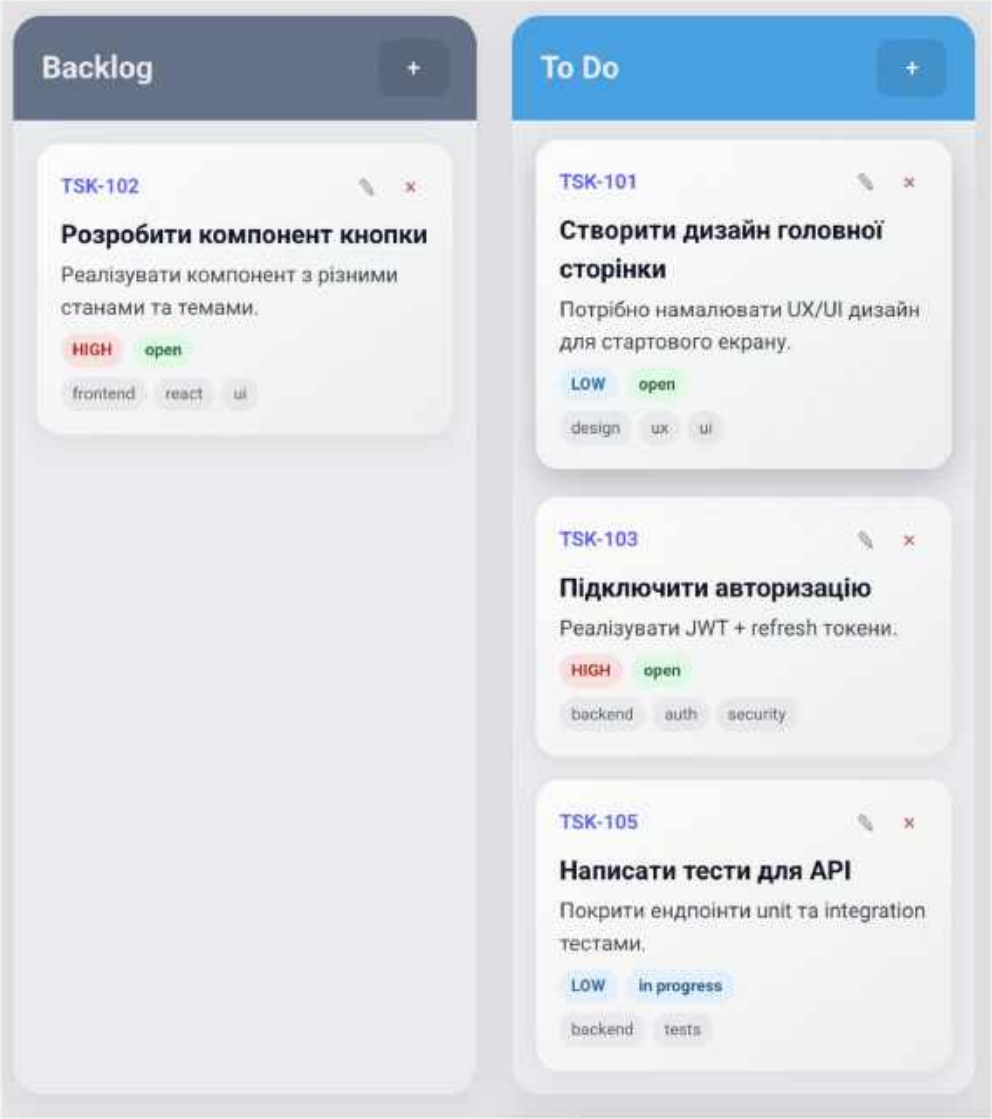

Рисунок 2.7 – Перенесена картка за допомогою drag&drop

Додавання задачі здійснюється через модальну форму, вбудовану безпосередньо в колонку. Після натискання кнопки «+» у відповідній колонці відображається форма з полями для введення номера задачі, заголовка, опису, вибору пріоритету, статусу та переліку тегів (через кому).

За замовчуванням колонка підставляє логічний статус (наприклад, Open для Backlog), а пріоритет може мати середнє значення (Medium), яке користувач за потреби змінює. Після заповнення форми й натискання кнопки «Додати»

відбувається валідація введених даних, створюється новий об’єкт задачі й додається до відповідного масиву в стані дошки.

Одночасно оновлені дані зберігаються в Local Storage, щоб задача не втрачалася при перезавантаженні сторінки. Кнопка «Скасувати» закриває форму без змін, що дозволяє користувачу комфортно керувати процесом додавання нових завдань.

Вікно редагування картки зображено на рисунку 2.8.

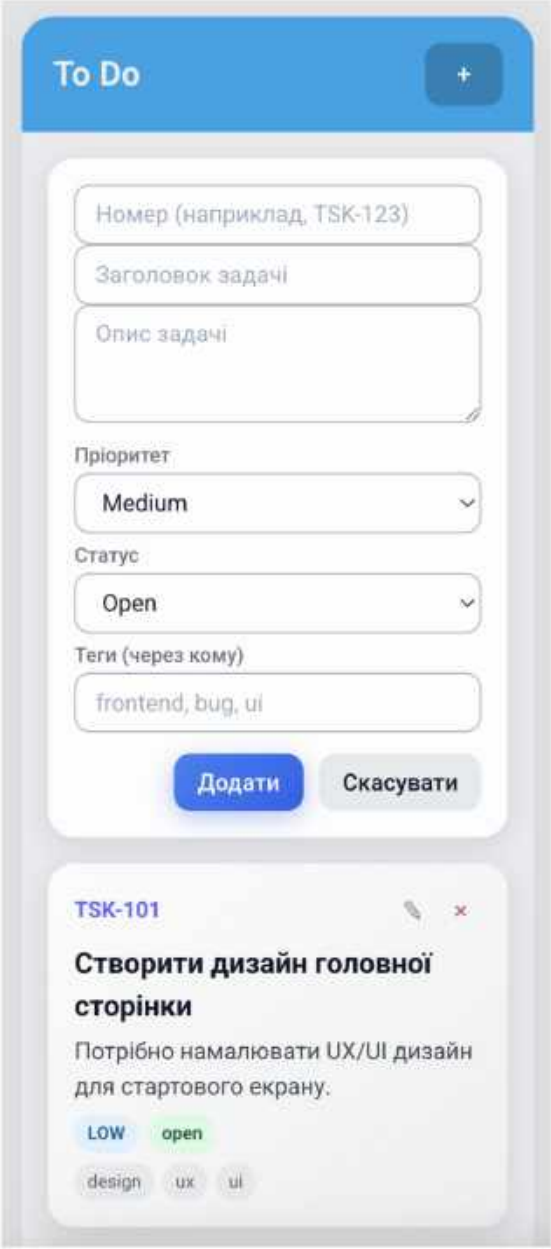

Рисунок 2.8 – Вікно редагування картки

Теги, пріоритети та статуси реалізовано як окремі атрибути задачі, які впливають як на візуальне відображення, так і на логіку фільтрації. Теги – це список ключових слів, що зберігається у вигляді масиву рядків; вони виводяться на картці у вигляді невеликих капсул (pill-компонентів) та використовуються для фільтрації за технологією, типом задачі чи контекстом (наприклад frontend, bug, analytics). Пріоритети (Low, Medium, High) мають фіксований перелік можливих значень і відображаються як кольорові бейджі, які дозволяють швидко оцінити важливість задачі. Статуси (Open, In progress, Blocked, Done) відображають поточний стан виконання й логічно пов'язані з розташуванням задачі в певній колонці. Наприклад, переміщення задачі в колонку Done супроводжується встановленням статусу Done.

Таке розділення атрибутів дає можливість одночасно використовувати їх як семантичні маркери для користувача та як параметри для програмної обробки (фільтри, статистика, майбутня аналітика).

Теги, пріоритети та статуси зображено на рисунку 2.9.

Рисунок 2.9 – Теги пріоритети та статуси

Виведення тегів, пріоритетів та статусів наведено нижче в лістингу 2.5.

Лістинг 2.5 – Виведення карток в колонці

```
<div className="task-meta-row">
     <span className={`badge badge-priority-${priority}`}>
          {priority.toUpperCase()}
     </span>
     <span className={`badge badge-status-${status}`}>
          {status.replace("-", " ")}
     </span>
</div>
```

```
{tags.length > 0 && (
      <div className="task-tags-row">
            {tags.map((tag) => (
                  <span key={tag} className="tag-pill">
                        {tag}
                  </span>
            ))}
      </div>
)}
```

кінець лістингу 2.5

Фільтри реалізовані як окрема частина стану дошки (board.filters) і включають кілька параметрів: текстовий пошук, фільтрацію за тегом, пріоритетом та статусом. При зміні будь-якого з цих значень через UI (input або select) викликається метод handleFilterChange, який оновлює стан фільтрів.

Під час обчислення списку задач для кожної колонки застосовується послідовна фільтрація: задачі відбираються за відповідністю текстового поля (номер, заголовок, опис), наявністю вказаного тегу, а також співпадінням обраного пріоритету й статусу (якщо для них встановлено значення, відмінне від all).

Таким чином, користувач може гнучко комбінувати фільтри, звужуючи множину задач до тих, що відповідають поточному контексту роботи, що особливо важливо при великій кількості елементів на дошці.

Виведення тегів, пріоритетів та статусів наведено нижче в лістингу 2.6.

Лістинг 2.6 – Метод handleFilterChange

```
const handleFilterChange = (
      field: keyof FiltersState, value: string
) => {
      setFilters((prev) => ({ ...prev, [field]: value }));
};
```

кінець лістингу 2.6

Пошук, фільтри по тегу, селект пріоритетів, селект статусів та кнопка вибору теми зображено на рисунку 2.10.

Рисунок 2.10 – Перемикач тем, фільтри та пошук

Пошук реалізовано як часткове текстове співпадіння за кількома полями задачі. Значення з поля «Пошук по номеру, тайтлу, опису…» зберігається у фільтрі search, після чого при відборі задач усі рядкові поля (ідентифікатор, заголовок, опис) приводяться до єдиного регістру й перевіряються на наявність підрядка. Завдяки такому підходу користувач може знайти задачу як за точним номером (наприклад, TSK-103), так і за ключовим словом з опису («авторизація», «аналітика» тощо). Пошук працює в реальному часі: список задач оновлюється одразу при введенні нового символу, що підвищує швидкість взаємодії та відповідає сучасним вимогам до UX.

Приклад пошуку по слову зображено на рисунку 2.11.

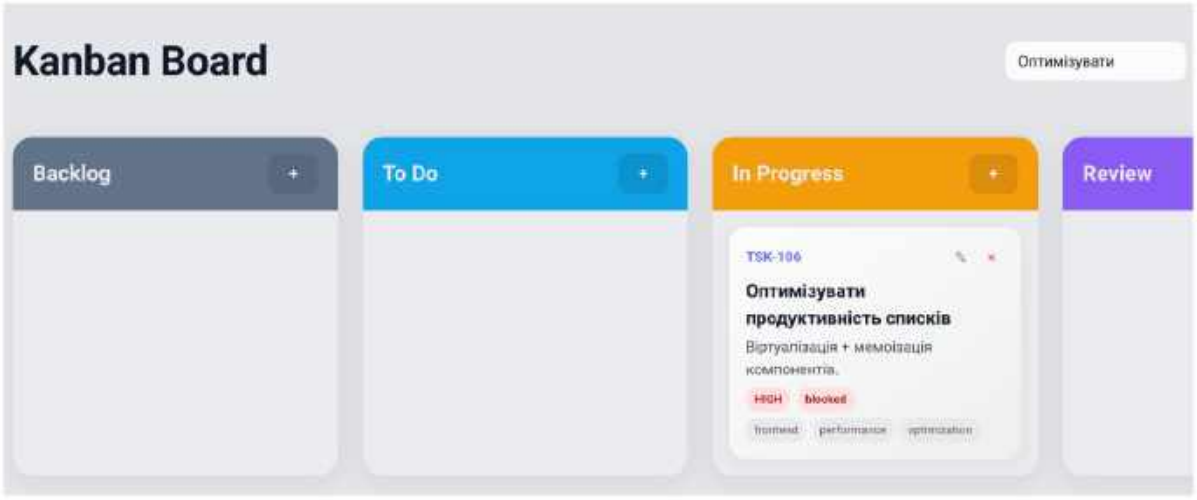

Рисунок 2.11 – Результат пошуку по слову

Приклад пошуку по номеру задачі зображено на рисунку 2.12.

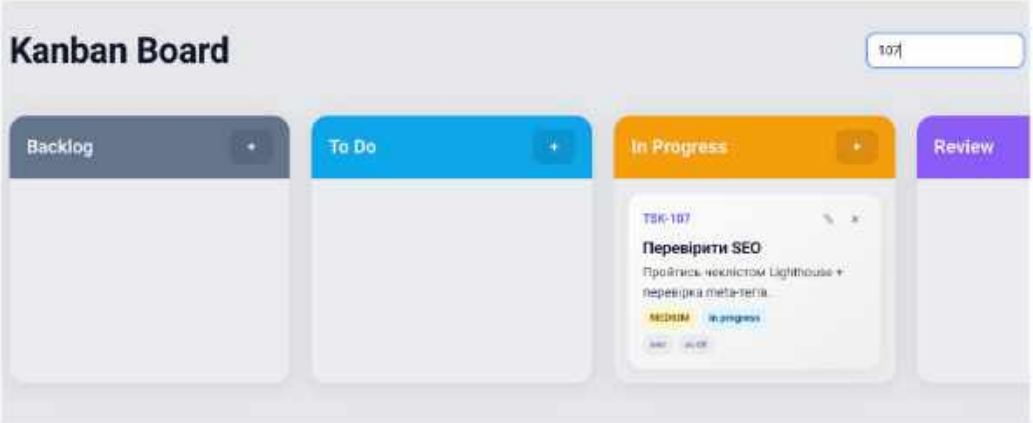

Рисунок 2.12 – Результат пошуку по номеру задачі

Візуальне оформлення дошки реалізовано за допомогою SCSS-стилів (kanban-board.scss) з використанням змінних для кольорових палітр light/dark-теми. Макет побудовано на основі flex-розмітки: колонки розташовуються в один ряд з рівномірними відступами, картки мають тінь, заокруглені кути та легкий ефект «підняття» при наведенні, що створює відчуття фізичності елементів. Для анімацій використано CSS-транзиції малого інтервалу (200–300 мс) – плавна зміна тіні, фону, масштабу при hover, а також плавне підсвічування колонок при drag-and-drop. Перемикання теми оформлення реалізується шляхом зміни класу theme-light / theme-dark на кореневому контейнері інтерфейсу. Після встановлення відповідного класу автоматично активується інший набір кольорових змінних, що визначають фон, текстові елементи, акцентні відтінки та допоміжні ефекти. Такий механізм дозволяє централізовано керувати всією палітрою застосунку без потреби у складних стилістичних переозначеннях.

Завдяки цьому підходу перемикання теми відбувається швидко й непомітно для користувача, не створюючи зайвих візуальних навантажень. Інтерфейс зберігає відчуття легкості та плавності, що позитивно впливає на загальне користувацьке сприйняття. Візуальний приклад зміни теми подано на рисунках 2.13 та 2.14.

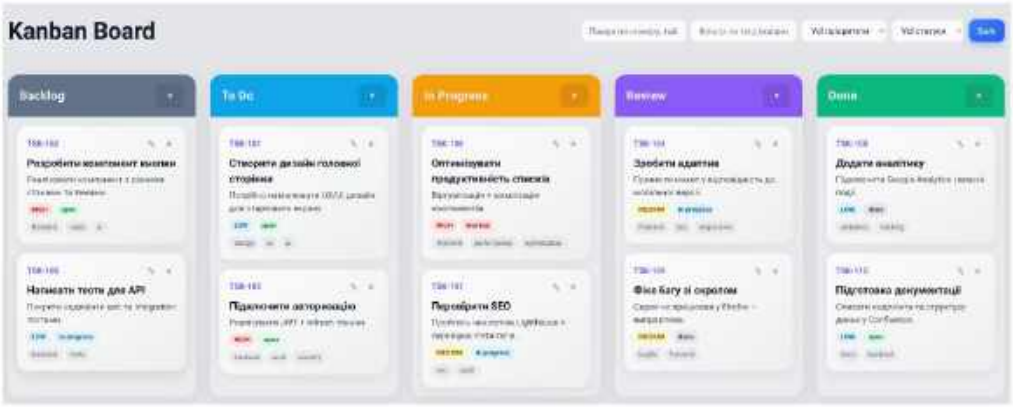

Рисунок 2.13 – Візуальне оформлення дошки

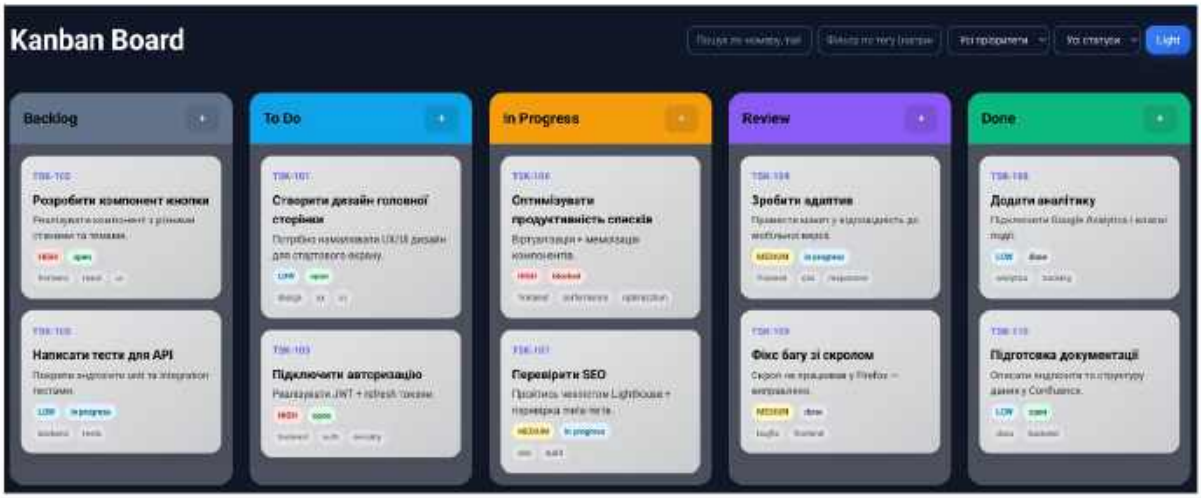

Рисунок 2.14 – Темна тема для інтерфейсу

Для забезпечення збереження стану між сесіями у застосунку використовується локальне сховище браузера (Local Storage), яке є одним з найпоширеніших механізмів клієнтського зберігання даних. На етапі первинного завантаження інтерфейсу спеціальний хук useKanbanBoard виконує перевірку наявності збережених даних у Local Storage за визначеним ключем. Якщо такі записи існують, вони десеріалізуються з формату JSON та стають актуальним набором задач, що дозволяє повністю відновити робочий контекст користувача..

Кожна зміна, що стосується задач – додавання нової картки, редагування атрибутів (назви, опису, пріоритету, тегів), видалення або переміщення між колонками з використанням drag-and-drop – моментально фіксується у

застосунку та супроводжується оновленням локального сховища. Оновлений масив задач проходить процес серіалізації в JSON-формат і повторно записується в Local Storage, забезпечуючи повну синхронізацію відображених на екрані даних і інформації, збереженої у браузері. Формат в якому зберігаються дані зображено на рисунку 2.15.

Такий підхід гарантує безперервність роботи користувача навіть у разі перезавантаження сторінки, вимкнення браузера або завершення сесії. Застосунок не потребує серверної частини для зберігання стану, що спрощує архітектуру, підвищує продуктивність і зменшує затримки при взаємодії. Таким чином, використання Local Storage суттєво підсилює надійність, автономність і практичну цінність Kanban-системи, роблячи її ефективною навіть у режимі повного офлайн-доступу.

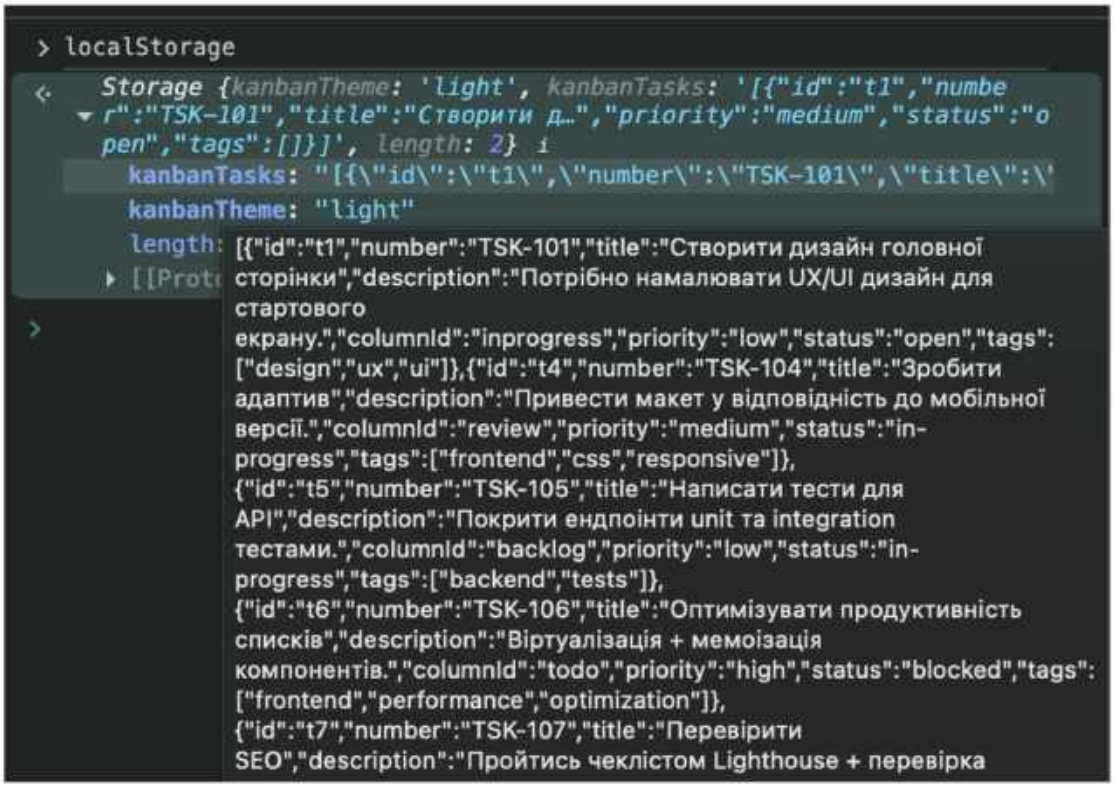

Рисунок 2.15 – Збереження даних в Local Storage

РОЗДІЛ 3
ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ РЕЗУЛЬТАТИВНОСТІ ІНТЕРФЕЙСУ ДЛЯ ІНТЕРАКТИВНОГО УПРАВЛІННЯ ПРОЕКТАМИ З KANBAN-ДОШКОЮ

3.1 Методика проведення дослідження

Методика експериментального дослідження була сформована з метою комплексної оцінки функціональних та продуктивних властивостей розробленого інтерфейсу Kanban-дошки шляхом вимірювання технічних параметрів, що характеризують швидкодію, стабільність роботи, масштабованість та ефективність взаємодії компонентів. Дослідження передбачало використання об'єктивних кількісних метрик, що дозволяють оцінити реальну продуктивність інтерфейсу без залучення користувачів та без залежності від суб'єктивних оцінок. Такий підхід забезпечує високу відтворюваність результатів та чіткість порівняння характеристик у різних умовах.

На першому етапі було визначено набір ключових показників, що підлягали вимірюванню. До них увійшли: час первинного рендерингу інтерфейсу, час рендерингу окремих компонентів, затримка обробки drag-and-drop подій, частота оновлення кадрів (FPS), швидкість виконання фільтрації та пошуку задач, час оновлення даних у Local Storage, а також обсяг використання оперативної пам'яті залежно від кількості задач. Ці параметри є показовими для аналізу якості реалізації SPA-застосунку, побудованого на React, та дозволяють оцінити його технологічну ефективність.

Другий етап передбачав моделювання реальних сценаріїв експлуатації інформаційної системи. Було сформовано чотири набори тестових даних: 50, 100, 500 та 1000 задач. Кожен набір імітував відповідний рівень навантаження – від роботи з невеликими групами задач до обробки великих потоків інформації. Для кожного обсягу виконувалися однакові процедури тестування: рендеринг інтерфейсу, активне переміщення задач, масова фільтрація та пошук, оновлення

атрибутів задачі, очищення та повторний запис даних у локальне сховище. Це дало змогу дослідити поведінку системи під різним рівнем навантаження та оцінити її масштабованість.

На третьому етапі здійснювалися інструментальні вимірювання, які проводилися за допомогою інструментів Chrome DevTools: Performance, Memory та Lighthouse. Під час вимірювань фіксувалися тимчасові характеристики, кількість повторних рендерів, стабільність роботи механізмів drag-and-drop та обсяг споживаних ресурсів. Для дослідження продуктивності алгоритмів використовувалися теоретичні методи аналізу складності відповідних операцій, що дозволило оцінити оптимальність структур даних, застосованих у системі.

На четвертому етапі проведено порівняння отриманих результатів з аналогічними показниками популярних інструментів Kanban-класу, зокрема Trello та Jira. Для цього використовувалися відкриті дані, а також власні вимірювання продуктивності інтерфейсів зазначених систем в аналогічних умовах. Порівняння дозволило визначити відмінності у швидкості рендерингу, плавності роботи та стабільності інтерфейсів при виконанні базових операцій.

На завершальному етапі результати були систематизовані у вигляді таблиць і графічних залежностей, що надало можливість провести кількісний аналіз характеристик системи, простежити тенденції зміни параметрів під навантаженням та сформулювати висновки щодо ефективності реалізованого інтерфейсу.

3.2 Обробка та аналіз отриманих результатів

Проведене експериментальне дослідження дозволило отримати низку кількісних характеристик, що відображають продуктивність та технічну ефективність розробленого інтерфейсу Kanban-дошки. Зібрані дані були впорядковані та подані у вигляді таблиць і графіків, що забезпечує наочність порівняння та дозволяє глибше оцінити поведінку системи при різних умовах

роботи. Аналіз отриманих результатів виконувався за кількома напрямами: швидкодія рендерингу, ефективність drag-and-drop, продуктивність обробки даних, масштабованість та ресурсомісткість.

Час первинного рендерингу Kanban-дошки становив у середньому 70 мс, що є високим показником для SPA-застосунків. Подальший аналіз показав, що відображення колонки займає близько 1,5 мс, а картки задачі – лише 0,3 мс, що свідчить про оптимізовану структуру компонентів та ефективне використання механізмів віртуального DOM (табл. 3.1).

Таблиця 3.1 – Час рендерингу компонентів

Компонент	Час, ms
Первинний рендер дошки	70
Рендер колонки	1,5
Рендер картки	0,3

Механізм перетягування задач продемонстрував стабільні показники плавності. Середній FPS при 100 задачах становив 58 кадрів/сек, а при 1000 задачах – 45 кадрів/сек, що забезпечує комфортний рівень взаємодії (табл. 3.2).

Таблиця 3.2 – Характеристики drag-and-drop (демонструє рівномірне зниження FPS при збільшенні навантаження)

Кількість задач	FPS	Drag latency, ms
50	60	6–7
100	58	7–8
500	52	9–10
1000	45	14–16

Аналіз використання пам’яті показав її лінійне збільшення без ознак витоків або різких стрибків. Це свідчить про оптимальну структуру збереження задач (табл. 3.3).

Таблиця 3.3 – Споживання пам’яті

Кількість задач	Пам’ять, MB
50	1.2
100	2.1
500	9.8
1000	18.3

Фільтрація навіть 1000 задач виконувалася за 18 мс, що забезпечує миттєву реакцію інтерфейсу (табл. 3.4).

Таблиця 3.4 – Час фільтрації

Кількість задач	Час, ms
100	4
300	9
500	12
1000	18

Порівняння показників з Trello та Jira засвідчило переваги реалізованої системи за основними технічними характеристиками (табл. 3.5).

Таблиця 3.5 – Порівняння продуктивності

Метрика	Trello	Jira	Наш інтерфейс
Первинний рендер, ms	120	180	70
Drag latency, ms	20-30	40	8-10
FPS при 500 задачах	40-50	35-45	52
Фільтрація 1000 задач, ms	25-40	40-60	18

Висновки до розділу 3

Отримані результати експериментального дослідження свідчать про стабільну, надійну та прогнозовану роботу розробленого інтерфейсу, що підтверджує високу якість його технічної реалізації. Проведені заміри часу рендерингу та реакції на користувацькі дії засвідчили низькі затримки навіть за умов значного збільшення обсягу даних, що є критично важливим для Kanban-систем, орієнтованих на щоденне використання в умовах інтенсивної зміни задач. Висока плавність механізму drag-and-drop, стабільне значення частоти оновлення кадрів (FPS) та мінімальні затримки при виконанні операцій переміщення задач свідчать про те, що інтерфейс забезпечує комфортну взаємодію без ривків, артефактів та відчутної для користувача затримки відгуку.

Аналіз продуктивності обробки даних показав, що фільтрація, пошук і оновлення статусів задач відбувається з практично миттєвою швидкістю, без необхідності перезавантаження сторінки або витратних запитів до серверної частини, що особливо важливо для локальних чи офлайн-сценаріїв роботи користувачів. Масштабованість системи, яка проявляється у лінійному зростанні використання пам'яті та стабільній продуктивності при збільшенні кількості задач до 1000 елементів, підтверджує ефективність обраної внутрішньої структури даних та оптимальність архітектурних рішень.

Порівняння продуктивних характеристик із поширеними Kanban-інструментами, такими як Trello та Jira, показало конкурентні переваги розробленого рішення: менший час первинного рендерингу, нижче drag latency та стабільніший FPS у процесі перетягування задач. Це означає, що в умовах однакових функціональних вимог інтерфейс забезпечує вищий рівень чуйності, що безпосередньо впливає на швидкість прийняття рішень користувачами та підвищує загальну ефективність робочого процесу.

Таким чином, виконане дослідження доводить, що розроблена Kanban-система не лише відповідає сучасним вимогам до швидкодії та якості UI/UX, але й демонструє значний потенціал для подальшого вдосконалення.

Зокрема, збережений запас продуктивності дозволяє впроваджувати нові модулі, такі як інструменти командної взаємодії, аналітика потоку роботи, синхронізація в режимі реального часу та інтеграції з іншими корпоративними сервісами без потреби радикального перегляду архітектури. Це забезпечує довготривалу життєздатність і масштабованість системи, а також підтверджує успішність прийнятих технічних рішень у ході її розробки.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи магістра було здійснено комплексне дослідження, спрямоване на розробку інтерактивної Kanban-системи, яка забезпечує оптимізацію робочих процесів, підвищення ефективності управління задачами та покращення користувацького досвіду. Усі етапи роботи були спрямовані на досягнення поставлених цілей і виконання визначених завдань.

У результаті виконання системного аналізу предметної області було визначено ключові особливості інтерактивних Kanban-систем, зокрема принципи візуалізації робочих процесів, основні ролі користувачів та вимоги до гнучкості керування задачами. Проаналізовано існуючі інструменти та їхні обмеження, що дозволило сформувати чіткий набір функціональних, технічних та UX-вимог до розроблюваного веб-застосунку, орієнтованого на зручність взаємодії та оптимізацію робочих процесів.

Проведене експериментальне тестування продуктивності користувацького інтерфейсу показало стабільність роботи системи при виконанні основних сценаріїв: переміщення задач між колонками, додавання та редагування карток, фільтрація та пошук. Отримані результати підтвердили відповідність застосунку вимогам швидкодії та інтерфейсної чуйності, що є критично важливим для інтерактивних Kanban-дошок.

Проектування інтерфейсу було здійснено з урахуванням принципів мінімізації когнітивного навантаження користувача та забезпечення оптимальної візуалізації інформації. Представлені рішення дозволяють швидко орієнтуватися у статусах задач, пріоритетах та виконавцях, а також сприяють більш ефективному прийняттю рішень у процесі планування та виконання робіт.

Створена архітектурна модель веб-застосунку на базі сучасних фреймворків забезпечує масштабованість, повторне використання компонентів та підтримку динамічної взаємодії. Реалізоване drag-and-drop керування задачами та реактивне оновлення інтерфейсу сприяють високому рівню

інтерактивності та адаптивності системи до потреб користувачів.

Розроблено механізми локального збереження даних у браузері, що дозволяє працювати з системою автономно без втрати задач. Реалізовані фільтрація та пошук забезпечують зручність управління великими обсягами інформації, підвищуючи ефективність користувацького досвіду та швидкість доступу до потрібних елементів.

У межах роботи сформовано рекомендації щодо подальшого вдосконалення Kanban-системи, включно з можливістю розширення функціоналу, інтеграцією з зовнішніми сервісами та впровадженням системи користувацьких ролей. Проведено аналіз перспектив масштабування рішення, що підкреслює його потенціал для використання як у малих командах, так і в корпоративних середовищах.

Загалом, результати кваліфікаційної роботи підтверджують актуальність обраної тематики та ефективність розробленого рішення. Створена система відповідає сучасним вимогам до веб-технологій, забезпечує інтерактивність, надійність та зручність експлуатації. Подальший розвиток проєкту дозволить розширити його функціональні можливості та сприятиме впровадженню у реальні процеси керування задачами та командною діяльністю.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Шворак П. Development And Optimization Of An Interactive Interface For Project Management Based On The Kanban Methodology. V Міжнародна науково-практична конференція: Research in Science, Technology and Economics. 10-12 грудня, 2025. Люксембург. С. 246-248.

2. Anderson D. J. Kanban Guide for Agile Teams. Kanban University. 2020. URL: https://resources.kanban.university/kanban-guide (дата звернення: 10.05.2025).

3. Leopold K., Wintergerst S. Flight Levels Academy – Official Documentation. Flight Levels. 2022. URL: https://flightlevels.io/resources (дата звернення: 09.12.2025). URL: https://kanban.university/kmm (дата звернення: 11.06.2025).

4. Олійник М. Г. Оптимізація продуктивності прогресивного web-застосунку бібліотеки на основі моделі RAIL. Чорноморський національний університет імені Петра Могили. Миколаїв, 2023. 82 с.

5. Моісеєнко Н. В. Веб-застосунок Kanban для підвищення ефективності управління IT-проєктами. Матеріали міжнародної конференції «Інформатика та системні науки». Кропивницький, 2024. С. 92-99. URL: https://elibrary.kdpu.edu.ua/bitstream/123456789/12013/1/paper32.pdf (дата звернення: 10.05.2025).

6. Шевченко І. О. Використання Agile-та Kanban-підходів у процесах розробки програмного забезпечення. Науковий журнал «Комп'ютерні науки та інформаційні технології». Одеський національний політехнічний університет, 2022. № 4. С. 33-40.

7. Agile Ukraine. Офіційні матеріали та переклади Agile-маніфесту, Kanban-гайдів, статей. 2023. URL: https://agile.org.ua (дата звернення: 12.07.2025).

8. Trello. The Official Trello Guide: Kanban, Productivity Patterns, and Workflows. 2024. URL: https://trello.com/guide (дата звернення: 10.09.2025).

9. MDN Web Docs. Drag-and-Drop API: Patterns & Performance Tips. 2025. URL: https://developer.mozilla.org/en-US/docs/Web/API/HTML_Drag_and_Drop_API (дата звернення: 11.09.2025).

10. Notion Labs. Notion API – Official Documentation. 2024. URL: https://developers.notion.com (дата звернення: 20.10.2025).

11. React Team. React Docs – Rendering, State, Hooks. Meta (Facebook), 2023. URL: https://react.dev (дата звернення: 20.10.2025).

12. Vercel. Next.js Documentation – App Router, Rendering, Optimizations. 2024. URL: https://nextjs.org/docs (дата звернення: 24.10.2025).

13. Microsoft. TypeScript 5.x Release Notes: Performance & Productivity. 2023. URL: https://devblogs.microsoft.com/typescript (дата звернення: 01.11.2025).

14. Evan You, Vite Team. Vite – Next Generation Frontend Tooling. 2024. URL: https://vitejs.dev/guide (дата звернення: 01.11.2025).

15. Sass Team. Sass Official Documentation – Modules, Mixins, Functions. 2023. URL: https://sass-lang.com/documentation (дата звернення: 01.11.2025).