Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

# КВАЛІФІКАЦІЙНА РОБОТА

## Пояснювальна записка

Рівень вищої освіти перший (бакалаврський)

Онлайн-платформа для організації колективної діяльності

(тема)

Виконав:
здобувач 4 року навчання,
групи КІУКІ-21-2
Дмитро ТОПОРЕЦЬ
(власне ім’я, прізвище)

Спеціальність
123 «Комп’ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма
Комп’ютерна інженерія
(повна назва освітньої програми)

Керівник: ас. Олег ЖУРИЛО
(посада, власне ім’я, прізвище)

Допускається до захисту

Завідувач кафедри ЕОМ _____ Андрій КОВАЛЕНКО
(підпис) (власне ім’я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти перший (бакалаврський)

Спеціальність 123 «Комп'ютерна інженерія»
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Комп'ютерна інженерія
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
"____" _____ 20__ р.

# ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Топорцю Дмитру Олексійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Онлайн-платформа для організації колективної діяльності

затверджена наказом по університету від " 26 " травня 2025 р. № 424 Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 17 червня 2025 р.

3. Вхідні дані до роботи 1) мережева топологія: багаторівнева клієнт-серверна архітектура із сервером-координатором; 2) стек технологій MERN; 3) бета-версія передбачає: редактор проектів та активностей, система сповіщень, інтерактивний розклад, базові інструменти адміністратора

4. Перелік питань, що потрібно опрацювати у роботі
1) аналіз проблеми та огляд існуючих рішень;
2) вибір технології розробки та інструментальних засобів;
3) розробка програмного забезпечення;
4) розробка програмних модулів;
5) відлагодження програмних модулів;
6) висновки.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій

Слайд -презентація – 10 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1 )

| Найменування розділу | Консультант (посада, прізвище, ім'я, по батькові) | Позначка консультанта про виконання розділу | |
|---|---|---|---|
| | | підпис | дата |
| | | | |
| | | | |

## КАЛЕНДАРНИЙ ПЛАН

| № | Назва етапів роботи | Строк / терміни виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | Аналіз проблеми та огляд існуючих рішень | 27.05.25-30.05.25 | |
| 2 | Вибір технології розробки та інструменталь- | 31.05.25-02.06.25 | |
| | них засобів | | |
| 3 | Розробка алгоритмічного забезпечення | 03.06.25-05.06.25 | |
| 4 | Розробка та відлагодження програмного | 06.06.25-08.06.25 | |
| | забезпечення | | |
| 5 | Оформлення матеріалів кваліфікаційної роботи | 9.06.25-12.06.25 | |
| 6 | Подання кваліфікаційної роботи керівникові та | 13.06.25-16.06.25 | |
| | її попередній захист | | |
| 7 | Подання кваліфікаційної роботи на рецензу- | 16.06.25-17.06.25 | |
| | вання | | |
| | | | |

Дата видачі завдання " 26 " травня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ ас. Олег ЖУРИЛО
(підпис) (посада, власне ім'я, прізвище)

# РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 74с., 21рис., 1табл., 2дод., 38джерел.

СИСТЕМА РЕАЛЬНОГО ЧАСУ, РОЗПОДІЛЕНА СИСТЕМА, ІНТЕРНЕТ, MERN, КЛІЄНТ, СЕРВЕР, JAVA SCRIPT, NODE.JS, REACT.JS, EXPRESS.JS.

Метою кваліфікаційної роботи є створення бета-версії веб-платформи для організації колективної діяльності як розподіленої системи реального часу.

У ході виконання кваліфікаційної роботи було розроблено бета-версію стало-орієнтованої веб-платформи для організації колективної діяльності мовою JavaScript із використанням стеку технологій MERN. Платформа має забезпечувати: доступ користувачів до редактору проектів та активностей, з якими користувачі можуть взаємодіяти в реальному часі; інтерактивний розклад; можливості редагування профілю; систему сповіщень та інструменти адміністратора.

Впроваджена топологія, хоч і є обмеженою на даному етапі розробки, покриває весь необхідний функціонал та придатна для розширення через впровадження клієнтських серверів.

# ABSTRACT

Bachelor’s thesis: 74pages, 21figures, 1tables, 2appendices, 38sources.

FIREWALL, GATE, INTERNET, PROTOCOL, ROUTER, SERVER, WI-FI, WIRELESS NETWORK, WLAN.

The major goal of this thesis is to create a beta-version of a web-platform for organizing collective activities as a distributed real-time system.

During the implementation of this thesis, a beta version of a sustainability-oriented web platform for organizing collective activities was developed using JavaScript and the MERN technology stack. The platform is designed to provide users with: access to project and activity editor for real-time interaction, an interactive schedule, profile editing capabilities, a notification system, and administrative tools.

The implemented topology, although limited at this stage of development, covers all the necessary functionality and is designed to scale through the deployment of individual worker server instances assigned to clients.

ЗМІСТ

# СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

БД – база даних

API – спосіб взаємодії між компонентами програмного забезпечення (англ., Application Programming Interface)

DAO — це шаблон проектування, який відповідає за абстрагування доступу до бази даних (англ., Data Access Object)

Dashboard – тип графічного інтерфейсу користувача, що забезпечує наочну презентацію основних показників продуктивності, значимих для конкретної цілі чи процесу

DOM – об’єктна модель документа, яка представляє HTML- або XML-документ у вигляді дерева об’єктів, якими можна керувати з коду (англ., Document Object Model)

HTTP – протокол передачі даних (англ. HyperText Transfer Protocol)

JSON- формат обміну даними (англ., Java Script Object Notation)

JSX – розширення синтаксису JS (англ., JavaScript XML)

LEMP – стек технологій розробки (англ., Linux Nginx MySQL PHP)

MERN – стек технологій розробки (англ., MongoDB Express.js React.js Node.js)

SASS – скриптова метамова (англ., Syntactically Awesome Stylesheet)

SPA – (англ., Single Page Applications) – односторінковий веб-застосунок

UML – уніфікована мова моделювання (англ., Unified Modeling Language)

WebSocket – технологія дву-направленого мережевого з’єднання

# ВСТУП

Разом з тим як у світі, а особливо в ІТ сфері, спостерігається швидке зростання кількості рішень для робочих задач, постає велика проблема. Зворотнім боком цього різноманіття є складність в освоєнні великої кількості інструментів різного формату, які вирішують однакові або дуже подібні задачі, що суперечить сімнадцятій цілі сталого розвитку «Співпраця для досягнення цілей».

Надмірність у використанні цифрових інструментів ускладнює як внутрішню координацію в командах, так і взаємодію між різними організаціями. Наприклад, організація колективної діяльності ІТ підприємств та навчальних закладів потребує одночасного використання інструментів для планування, комунікації, спільної взаємодії та управління ресурсами.

Зазначена проблема не є новою, тому для її вирішення було створено багато інструментів, таких як «“Jira»», «Trello», «Asana», «Notion», «DL.nure», «VSCode Live Share», тощо. Ці інструменти реалізують схожий функціонал, хоч і мають специфічні до окремих галузей елементи. Такими відмінностями можуть бути: реалізація діаграм Ганта, можливість спільної роботи з документами в реальному часі, інтеграція з системами контролю версій або освітніми платформами, підтримка ролей користувачів з різним рівнем доступу, розсилка сповіщень-нагадувань, тощо.

Метою даної дипломної роботи є розробка веб-платформи, яка дозволить поєднати функціонал різних інструментів для організацій колективної діяльності та надасть єдиний динамічно налаштовуваний інтерфейс, що буде залишатись зрозумілим для всіх користувачів платформи, незалежно від відмінностей у налаштуваннях та методах використання.

Розробка та програмна реалізація онлайн платформи виконується з використанням стеку технологій для розробки веб-додатків «MERN» (MongoDB, Express, React, Node.js), який передбачає використання однієї

мови програмування як для клієнту, так і для серверу: JavaScript. Додатково використовуються мова розмітки «Html», мова стилів «Css» з препроцесором «Sass» та мова запитів «MQL» (MongoDB Query Language).

Для вирішення поставлених завдань у роботі використано комплекс методів: аналіз науково-технічної літератури та інтернет-джерел для вивчення проблеми та існуючих рішень; системний аналіз для визначення вимог до рішення та проектування його архітектури; мультипарадигмове програмування для реалізації програмного коду; методи тестування програмного забезпечення для перевірки коректності функціонування розробленого рішення; методи порівняльного аналізу для обґрунтування вибору технологій.

Наукова частина спрямована на розробку розподіленої архітектури та програмної реалізації онлайн платформи на прикладі покриття функціоналу платформи «DL.nure» та частково «Notion», що дозволяє спроектувати мінімальну робочу версію рішення та задає напрямок для подальшого розширення функціоналу.

Для цього розроблено бета-версію платформи, яка реалізує взаємодію між клієнтом, сервером координатором, та робочими серверами; надає основні інструменти для організації діяльності (механізм роботи з проектами та подіями, розклад з можливістю переходу до сторінок подій напряму, панель навігації з історією відвіданих протягом сесії сторінок, оновлення даних проектів та подій в реальному часі, системи сповіщень та чатів).

# 1 АНАЛІЗ ПРОБЛЕМИ ТА ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

## 1.1 Аналіз проблеми

Сучасних розробок для організації колективної діяльності достатньо багато, і часто в одній організації одночасно використовують декілька сервісів схожого призначення. Проте, як зауважують експерти, «навантаження команди занадто великою кількістю інструментів може ускладнювати робоче середовище. Розсіювання даних по різних додатках робить складнішим отримання оперативних відповідей» [1].

У динамічному IT-середовищі та в освіті причини одночасного використання множини систем різні, однак наслідки спільні: фрагментація інформації, дублювання задач, плутанина у комунікації та перевантаження користувача. Наприклад, за даними «Atlassian», 86% організацій використовують понад 6 різних інструментів для управління та співпраці (вдвічі більше, ніж 5 років тому). Кожен департамент обирає «свій» сервіс, що створює розрізнене середовище: інформація розпорошується, а робочі потоки стають фрагментованими. Така розгалуженість призводить до зайвих витрат часу на перемикання між інструментами, дублювання задач і помилок, а також до зниження продуктивності: 93% керівників вважають, що команди могли б працювати вдвічі ефективніше при кращій інтеграції інструментів. Усе це створює «витрати від поганої співпраці» – затримки в проектах, надмірні зустрічі та вигоряння співробітників [2].

Причини появи т.зв. «тіньових» інструментів у компаніях та вишах різні. Іноді офіційні системи застарілі, громіздкі чи не задовольняють потреб користувачів, тому працівники самі знаходять сучасні рішення («Slack», «WhatsApp», тощо). Якщо організація пропонує багато варіантів і не дає чітких рекомендацій, це гарантовано призводить до «фрагментованого середовища» комунікації та співпраці. У підсумку користувачі просто

«розкидають» спілкування та дані по найзручніших для них платформах, що створює інформаційні силоси та загрожує безпеці даних [3]. В результаті виникає плутанина: неясно, де шукати ту чи іншу інформацію, які задачі актуальні, і чи не дублюються пункти роботи, що знижує загальну ефективність організації.

Дослідження показують: стандартизація та інтеграція інструментів значно підвищують ефективність. Зокрема, оптимізація та консолідація сервісів дозволяють заощаджувати кошти (усуваючи дублювання ліцензій) та покращувати користувацький досвід. Наприклад, в освіті перехід на єдину хмарну платформу («Microsoft Office 365» або «Google Workspace») дає «єдиний досвід для всіх учасників, спрощує адміністрування та тех-підтримку і знижує витрати за рахунок скасування дублюючих рішень» [4]. В IT-компаніях комплексна платформа (наприклад, стек «Atlassian» чи «Microsoft») теж зменшує час на перемикання контекстів і допомагає фокусуватися на результатах. Таким чином, хоч повна інтеграція різних систем (наприклад, об’єднання «Slack», «Jira» і «Confluence») потребує зусиль, вона окупається: працівники витрачатимуть менше часу на пошук інформації і більше – на вирішення реальних завдань.

## 1.2 Огляд існуючих рішень

«Jira» («Atlassian») – це потужний інструмент управління проектами та відстеження задач, особливо популярний у розробці ПЗ [5]. «Jira» дозволяє зручно розбивати роботу на під-задачі (issues), налаштовувати кастомні робочі процеси (workflow) і контролювати весь життєвий цикл задачі [6]. Платформа підтримує будь-який «Agile»-підхід («Scrum», Kanban або змішані методології): в «Jira» є «канбан»- та «скрам»-дошки, «беклог» задач, планування спринтів та релізів, а також дорожні карти (roadmaps) і звіти по продуктивності [7]. Користувачі можуть відстежувати баги, історії та задачі, пріоритезувати їх, переглядати аналітику в «dashboards» та будувати звіти.

Сильні сторони: надзвичайна гнучкість і розширюваність. Адміністратори можуть тонко налаштовувати поля задач, сповіщення, права доступу, автоматичні тригери тощо. «Jira» добре інтегрується з іншими продуктами «Atlassian» («Confluence, Bitbucket») та зовнішніми сервісами («Git», «CI/CD», плагіни). Інструмент підходить для команд будь-якого розміру й дозволяє централізовано керувати багатьма проектами. У нього є потужні засоби звітності і «dashboards», що допомагають контролювати ризики та терміни.

Недоліки: висока складність і крута крива навчання для нових користувачів. Налаштування «Jira» часто вимагає значних зусиль, а інтерфейс може здаватися перевантаженим. Також «Jira» традиційно платна (є безкоштовний тариф для малих команд), а великі установки на власних серверах вимагають часу та ресурсу на підтримку. Крім того, цей інструмент спроектовано передусім під «IT/DEV»: в непов’язаних з IT командах використовується рідко.

Сценарії використання: у IT-командах «Jira» застосовують для повного циклу розробки – від планування функціоналу та резервування до тестування і випуску релізів. В освітньому середовищі «Jira» можуть використовувати в проектних курсах чи студентських командах, де потрібен формалізований механізм відстеження завдань, однак це нетипово. Частіше для навчання використовують спрощені варіанти або самоорганізація в «LMS».

«Trello» – це простий візуальний менеджер задач на основі «Kanban»-методології. Кожен проект у «Trello» – це «дошка» зі списками, які містять картки–завдання. Користувачі перетягують картки між списками, додають описи, чек-листи, терміни, прикріплюють файли і коментують. Система відмінно підходить для оглядового планування і базового контролю статусу (Наприклад, списки «To Do», «Doing», «Done») [8].

Сильні сторони: інтуїтивний інтерфейс і простота освоєння. «Trello» захоплює користувача візуальним представленням проекту: кожен одразу бачить, хто за що відповідає і на якому етапі завдання [9]. Оновлення

відображаються в реальному часі, додавання учасників і нових карток надзвичайно просте. У «Trello» є безкоштовна версія з великою кількістю функцій, а також система плагінів («Power-Ups») – наприклад, календар, відстежування подій за часом, інтеграції з іншими системами.

Недоліки: відсутність розширених функцій проектного менеджменту. «Trello» не має вбудованого відстеження часу, діаграм Ганта, чи потужної системи звітів. Для масштабніших проектів із сотнями завдань простий «Kanban» може стати вузьким місцем: інструмент не найкраще масштабується на великі команди. Багато функціоналу доводиться добирати через зовнішні плагіни («Power-Ups»), що підвищує витрати та може викликати проблеми з сумісністю. Також «Trello» може накопичувати зайві деталі на картках, ускладнюючи перегляд високо-пріоритетних задач. Крім того, деякі користувачі відзначають обмежені можливості налаштування та проблеми з безпекою для конфіденційних даних через хмарне зберігання.

Сценарії використання: В освіті «Trello» використовують наочні дошки для організації групових проектів, планування уроків і домашніх завдань. Наприклад, викладачі чи студенти можуть заводити дошку курсу з картками для лекцій, дедлайнів та ідей. В IT-командах «Trello» часто служить для відстеження простого резервування, організації спринтів або загального огляду задач, особливо в невеликих проектах(startups) або командах маркетингу чи дизайну.

«Asana» – це відстежувач завдань/проектів, орієнтований на простоту і масштабованість. В «Asana» можна створювати проекти у вигляді списку, дошки «Kanban» або таймлайну («Gantt»-подання). Завдання підтримують під-задачі, пріоритети, мітки, коментарі і датування. Система дозволяє встановлювати залежності між задачами і слідкувати за виконанням глобальних цілей (метрик) проекту [10].

Сильні сторони: інтуїтивний інтерфейс і зручність використання. Як зазначено у відгуках, «Asana» «відрізняється простотою та продуманим дизайном, більшість людей освоюють її за кілька хвилин». Вона підтримує

співпрацю: є стрічка активностей, коментарі, сповіщення про зміни, інтеграції з поштою і месенджерами. Багато організацій цінують «Asana» за її фокус на командній роботі та інтеграції з іншими сервісами (Google Диск, Slack, тощо) [11].

Недоліки: основний – відсутність вбудованого відстеження часу, через що за потреби доведеться підключати додаткові сервіси. Також у «Asana» лише один відповідальний за задачу – якщо над задачею працює кілька людей, необхідно створювати дублікати завдань. Кількість опцій іноді ускладнює прості операції, а надлишок підменю може перевантажити новачка.

Сценарії використання: У навчанні «Asana» інколи застосовують для організації групових курсових чи дослідницьких проектів, адже вона зрозуміла студентам і дозволяє відстежувати дедлайни. В IT-командах «Asana» використовується для менеджменту нетехнічних задач, маркетингових планів, а також у невеликих розробницьких командах як альтернатива «Jira» для гнучкого планування. Особливо підходить для команд, які не потребують глибокого налаштування робочих процесів, але цінують наочність та простоту.

«Notion» – це «все-в-одному» онлайн-робочий простір, який поєднує документи, бази знань, бази даних, менеджмент задач та спільну роботу. У «Notion» можна створювати сторінки і в них – таблиці, списки, дошки «Kanban», календарі, файлові сховища тощо. Практично будь-яку сторінку можна зв’язати з базою даних, що робить «Notion» надзвичайно гнучким інструментом для структуризації знань [12].

Сильні сторони: надзвичайна гнучкість і універсальність. «Notion» «комбінує нотатки, задачі, бази даних в одному місці, дозволяючи централізувати весь робочий процес». Його гнучкість – одна з головних переваг: користувачі створюють власні шаблони, налаштовують будь-які бази даних під свої процеси [13]. Платформа підтримує співпрацю в реальному часі: кілька людей може одночасно редагувати одну сторінку,

коментувати і бачити зміни. Інтерфейс перетягування елементів є доволі інтуїтивним. Notion інтегрується зі «Slack», «Google», «Trello» та іншими популярними сервісами, що дозволяє зв’язати його з існуючим стеком робочих інструментів.

Недоліки: багатофункціональність породжує складності. Для новачків «Notion» може здатися перевантаженим – «занадто багато можливостей» інколи дезорієнтує. При великих і складних базах даних спостерігаються гальма в роботі, оскільки виробництво дуже великих документів не завжди оптимізоване. Також обмежені можливості роботи офлайн – без доступу до мережі «Notion» працює урізано. Ціна платних планів достатньо висока для великих команд (після безкоштовного тарифу слідують комерційні підписки).

Сценарії використання: У навчанні «Notion» часто використовують для створення цифрових нотаток, баз знань і часу повсякденних завдань: наприклад, студенти створюють власні блокноти з курсів, викладачі – спільні канцелярії лекцій. В IT-командах «Notion» служить одночасно і дошкою задач, і центром документації.

«DL.nure» – це система дистанційного/електронного навчання, створена на базі «Moodle» – широко відомої LMS (Learning Management System) [14]. Основне призначення – підтримка процесу навчання: керування курсами, лекціями, завданнями, тестами та форумами. Викладачі розміщують у системі матеріали, роздають завдання, контролюють оцінки та взаємодіють зі студентами, студенти отримують контент, здають роботи й проходять онлайн-тести [15].

Сильні сторони: LMS «Moodle» є відкритою та безкоштовною платформою, що дає значну економію та можливість глибокої кастомізації [16]. У «DL.nure» можна додавати модулі та плагіни (відеоконференції, чати, додаткові тести), тому університет самостійно розвиває функціональність під свої потреби. «Moodle» має велику спільноту розробників і користувачів, що гарантує постійне підтримування і численні навчальні ресурси. З технічних

можливостей є все необхідне для навчання: курси та під-курси, форуми, інтерактивні тести, оцінювання, оголошення, відстеження прогресу, календар подій, система повідомлень та ін.

Недоліки: система доволі складна у налаштуванні та підтримці: адміністратору доводиться самостійно забезпечувати хостинг, оновлення і безпеку. Через велику кількість меню і опцій інтерфейс може здаватися застарілим і перевантаженим. Новим викладачам буває нелегко освоїтися з усіма налаштуваннями курсів. Крім того, «Moodle» – це суто навчальна платформа: вона не призначена для гнучкої організації задач нерозробницького характеру або спільної роботи в реальному часі.

Сценарії використання: В освітньому процесі «DL.nure» (Moodle) є основним середовищем дистанційного навчання вишу. Там надають посилання на онлайн-лекції, розміщують матеріали, організовують тести. В IT-компаніях подібні LMS-системи використовуються переважно для корпоративного навчання чи адаптації нових працівників, але в робочому процесі застосовують інші інструменти («Teams», «Slack» та ін.).

«VSCode Live Share» – це розширення від Microsoft для середовища «Visual Studio» та «VS Code», яке дозволяє розробникам спільно працювати над кодом у реальному часі. Учасники обміну сесією (хост і гості) бачать одне й те саме вікно редактора: всі зміни (навіть вставка тексту чи переміщення курсора) одразу відображаються у всіх учасників. «Live Share» також дозволяє спільно запускати відладку (ділитися «breakpoints»), надавати доступ до інтегрованих терміналів, локальних веб-серверів і навіть інших IDE («Visual Studio», «JetBrains», «Eclipce»). Важлива перевага – гостеві не потрібно клонувати репозиторій чи налаштовувати у себе середовище (додавати залежності, перевіряти сумісність доповнень): достатньо приєднатися до сесії і працювати над існуючим відкритим проектом [17].

Сильні сторони: миттєва спільна робота над кодом. «Live Share» дозволяє «co-edit, co-debug, chat with your peers, share terminals, servers, [and] comments» без додаткових налаштувань. Інструмент працює з будь-якою

мовою програмування чи типом додатку – за будь-якої ОС можна поділитися проектом та спільно працювати. Це зручніше за звичайний демонстративний екран чи огляди «Git»: код синхронізується в обох редакторах, а текстовий чат допомагає коментувати хід роботи.

Недоліки: «Live Share» не є самостійним менеджером задач чи платформою для зберігання даних – він фокусується виключно на процесі розробки в редакторі, тому для організації задач потрібні додаткові системи. Щоб ним користуватися, усі учасники повинні мати встановлений «VS Code»/«Visual Studio» та відповідний плагін. Без підключення до Інтернету реального часу немає, бо сесія хоститься в хмарі.

Сценарії використання: «Live Share» широко використовується в освіті та тренінгах: викладачі програмування можуть ділитися кодом з класом, а студенти працюють над ним спільно та задають питання в процесі. Паралельно розробники можуть коментувати та робити поправки – це значно ефективніше за пасивний перегляд екрану. Також «Live Share» застосовується для парного/моб-програмування, спільного дебагу, наставництва та адаптації нових розробників, технічних інтерв'ю та хакатонів. Наприклад, колеги можуть одразу допомогти з багом, зайшовши в сесію без довгого листування кодами та логами помилок.

Для порівняння наведених рішень, їх зведено в таблицю 1.1.

Таблиця 1.1 – Порівняння існуючих рішень

| Інструмент | Тип/галузь | Платформа | Переваги | Недоліки | Сценарії використання |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 |
| “Jira» | Професійний проектний менеджмент для IT | Web, Desktop, Mobile | Глибока кастомізація, інтеграція з Git, Agile-підтримка | Складний для немовних команд, перевантажений інтерфейс | Dev-команди, середні та великі проекти, спринти |

Продовження таблиці 1.1

| 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|
| Trello | Візуальний менеджер задач | Web, Desktop, Mobile | Простий, drag-and-drop, інтеграції з Google Drive, Slack | Лімітована глибина проектів, немає вкладеності | Командні завдання, викладацька робота, навчальні курси |
| Asana | Менеджмент командної роботи | Web, Mobile | Інтуїтивний UX, таймлайни, шаблони | Не підходить для розробки зі складними залежностями | Планування курсів, менеджмент подій, навчальні проекти |
| Notion | Універсальний workspace | Web, Desktop, Mobile | База знань + задачі + документи, підтримка шаблонів | Відсутність повноцінного канбану в базовій формі | Індивідуальне та командне планування, лінгвістика, дослідницькі групи |
| Live Share | Інструмент спільної розробки | Visual Studio Code Extension | Спільне редагування в реальному часі, підтримка терміналу | Обмеженою функціональністю поза VS Code, тільки для коду | Спільне програмування, парне тестування, навчання |
| DL>NURE | LMS (Learning Management System) | Web | Централізоване управління курсами, доступ до матеріалів | Вузькоспеціалізована під конкретний ЗВО, застарілий UX | Навчальні дисципліни, дистанційна освіта, комунікація зі студентами |

# 2 ВИБІР СТЕКУ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ ПРОЕКТУ

## 2.1 Стек технологій MERN

Стек «MERN» є одним з найбільш популярних стеків для розробки клієнт-серверних веб-додатків [18]. Він дозволяє створювати додатки для будь-яких пристроїв на яких можливо встановити браузер. Даний стек є дуже популярним через простоту як у вивченні, так і при розробці через використання єдиної мови як для «front-end», так і для «backend» додатку.

Основна перевага стеку «MERN» – це використання однієї мови програмування, а саме JS. Це забезпечує єдність технологій на клієнтській та серверній стороні, що значно спрощує розробку додатків. Вивчення технологій стеку «MERN» є швидким у порівнянні з іншими стеками (наприклад «LEMP»), тому що JS є популярною мовою, яка підтримується багатьма відомими браузеріами та неспецифічність стеку в цілому.

Таким чином розробник може використовувати одну мову для написання коду як для браузера (Front-end), серверної логіки (Back-end), так і для роботи з базою даних (через MongoDB, яка зберігає дані у JSON форматі, сумісному з JavaScript-об'єктами).

Обраний стек є досить зручним для автоматизації тестування, оскільки всі компоненти цього стека взаємодіють через JS. Це спрощує використання тестових фреймворків зі схожими принципами роботи, для спрощення автоматизації тестів для кожної з частин проекту:

- «MongoDB» – із використанням «MongoDB Memory Server», «Mongoose Mock» та «Joi»;
- «Express.js» – із використанням «Mocha» та «Jest»;
- «React.js» – із «Jest», «React Testing Library» та «Enzyme»;
- «Node.js» – із використанням, «Jest», «Mocha», «Chai» та «Jasmine».

## 2.2 Архітектура платформи

Даний проект є комплексним рішенням, яке має частково визначену та частково динамічну структуру одразу на декількох рівнях організації. Таким чином веб-платформа «SciWork» може розглядатись як комп’ютерна мережа, комп’ютерна система реального часу та веб-додаток.

### 2.2.1 Топологія «координаційно-ієрархічна»

Топологія мережі описує, взаємозв’язки між пристроями комп’ютерної мережі. Кожна топологія має свої унікальні особливості, переваги та недоліки. Її вибір залежить від потреб системи, витрат на обладнання, а також вимог до продуктивності та надійності [19].

У випадку онлайн платформи, де клієнтами є не окремі користувачі, а цілі організації (IT компанії та заклади освіти), як основу використано динамічну «багатозв’язну» топологію на основі топології «зірка», в якій центром є сервер координатор.

Сервер координатор відповідає за утримання актуального списку адрес працюючих робочих серверів та надання їх користувачам. Зазначена структура є основою, оскільки передбачає динамічне додавання та видалення кінцевих вузлів та цілих підмереж.

Динамічними елементами мережі є кінцеві користувачі, які приєднуються до сервера координатора для отримання адрес окремих робочих серверів та їх підмереж, і власне робочі сервери та серверні підмережі утворені клієнтами. Такі мережі можуть мати будь-яку структуру, але дотримуються однакового принципу взаємодії з мережею в цілому (наразі наявний тільки загальний сервер, клієнтські сервери буде додано в бета-версії платформи). Ця топологія дозволяє зменшити навантаження на пристрої, обслуговувані платформою, та передати велику частину відповідальності і можливостей налаштування клієнтам.

### 2.2.2 Клієнт серверна взаємодія

Клієнт-серверна архітектура – це модель організації взаємодії між користувачем (клієнтом) і сервером (централізованою чи розподіленою системою), яка забезпечує доступ до ресурсів, даних або послуг. Існує кілька основних типів клієнт-серверних архітектур, кожна з яких має свої особливості, переваги та сценарії використання.

Традиційна одношарова клієнт-серверна архітектура базується на прямій взаємодії клієнта із сервером, в якій сервер виконує всі завдання обробки запитів, зберігання даних і управління користувачами. Це найпростіший тип архітектури. Головними перевагами є простота впровадження і централізоване управління, однак зростання навантаження може швидко перевищити можливості сервера [20].

Двошарова архітектура поділяє обов’язки між клієнтом і сервером, на відміну від однорівневої, клієнт відповідає не тільки за взаємодію з користувачем, а ще частково за обробку логіки. Це знижує навантаження на сервер, але водночас створює необхідність підтримувати оновлення клієнтського програмного забезпечення і ускладнює структуру проекту.

Багатошарова архітектура додає проміжні шари, такі як бізнес-логіка, обробка даних, кешування та маршрутизація. Ці шари можуть бути реалізовані через окремі сервери або мікросервіси. Головні переваги такої архітектури – гнучкість, масштабованість та розподіл навантаження.

Також існує мікросервісна архітектура, яка є різновидом багатошарової, де кожен мікросервіс відповідає за окрему функцію чи бізнес-процес. Вона дуже добре підходить для великих масштабованих систем, дозволяючи незалежно розробляти та розгортати кожен сервіс. Серед недоліків – висока складність моніторингу, налагодження та управління.

Архітектура із сервером-посередником або «шлюзова» модель передбачає наявність центрального вузла, що координує [21]. У такій системі клієнти не взаємодіють безпосередньо із серверами, які виконують основну

роботу. Замість цього запити передаються через центральний сервер, який вирішує, до якого робочого сервера їх направити. Однак центральний вузол стає єдиною точкою відмови, і його продуктивність критично важлива для всієї системи.

Тому для реалізації проекту було обрано модифіковану багаторівневу клієнт-серверну архітектуру із сервером-координатором (рисунок 2.1). Для запобігання відмовам сервер-координатор відповідає тільки за зберігання списків доступних робочих серверів та перенаправлення користувачів до них. Основні операції відбуваються на робочих серверах, які обслуговують конкретних клієнтів (організації). З точки зору топології та типів підключень, використовуються HTTP-запити для комунікації з сервером-координатором та WebSocket для встановлення постійного зв’язку з робочими серверами. Окрім того, реалізовано heartbeat-з’єднання, яке дозволяє координатору відстежувати активність робочих серверів [22].

У порівнянні з іншими моделями, обрана архітектура забезпечує баланс між централізованим управлінням і розподіленою обробкою даних. Завдяки серверу-координатору, можна легко масштабувати систему, додавати нові сервери та організації(клієнтів). Також зазначений розподіл забезпечує додатковий рівень захисту, адже робочі сервери не взаємодіють між собою напряму. У порівнянні з традиційною «шлюзовою» архітектурою, створена модель значно надійніша, та ефективніша в умовах великої кількості користувачів. Водночас вона є простішою у впровадженні, ніж мікросервісна архітектура.

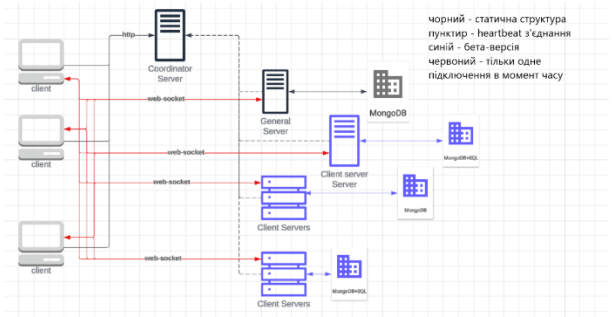



Рисунок 2.1 – UML-діаграма клієнт-серверної взаємодії

### 2.2.3 Дизайн клієнтського та серверних додатків

Дизайн серверу-координатора з використанням Node.js та Express.js зосереджений на ефективності, тому цей сервер виконує лише базові функції. Основні задачі – приймати HTTP-запити від клієнтів і перенаправляти їх до відповідних робочих серверів. В основі лежить структура, що дозволяє підтримувати актуальний список робочих серверів з їх адресами. Функціонал організовано з фокусом на швидкість обробки запитів.

Код починається з підключення необхідних бібліотек, таких як Express.js для роботи з HTTP-запитами та створюється базовий додаток Express. Сервер відповідає на запити клієнтів, надаючи їм адреси робочих серверів [23]. Наприклад, коли користувач звертається до координатора, сервер перевіряє, які з робочих серверів доступні, і повертає клієнту список адрес. Дизайн цього сервера не передбачає роботи з БД, тому всі дані про стан робочих серверів зберігаються в оперативній пам'яті. Оновлення списку серверів реалізується через періодичне надсилання сигналів «heartbeat» від робочих серверів до координатора. Це дозволяє визначити, чи сервер «живий», і видалити його зі списку у разі відсутності сигналу протягом визначеного проміжку часу.

Сервер-координатор не обробляє складну логіку чи великий потік даних. Його структура мінімалістична, а функції орієнтовані на управління мережею замість взаємодії з користувачами та БД.

Дизайн робочого сервера є більш складним і багатофункціональним. Робочий сервер обробляє HTTP-запити клієнтів, а також працює з WebSocket-з'єднаннями для забезпечення двосторонньої комунікації в реальному часі [24, 25]. На відміну від координатора, цей сервер зберігає дані у базі MongoDB, використовуючи бібліотеку Mongoose для управління моделями даних. Основна функція робочого сервера полягає в обробці запитів, які можуть включати авторизацію користувачів, отримання чи оновлення даних, а також підтримку постійного WebSocket з'єднання. При

обробці HTTP-запитів сервер використовує маршрути для організації коду: кожен маршрут відповідає за виконання окремої функції (наприклад, отримання інформації про обліковий запис користувача, оновлення даних у БД).

Для роботи з «WebSocket»-з'єднаннями сервер інтегрує бібліотеку «ws». Для встановлення клієнтами постійного з'єднання, яке використовується для миттєвої передачі даних. Наприклад, клієнт може підключитися до сервера для оновлення розкладу в реальному часі.

Щодо взаємодії з «MongoDB», сервер використовує моделі «Mongoose» для структурування даних [26]. Дані, такі як інформація про користувачів, проекти та події, організовуються у вигляді моделей. Це забезпечує чітку структуру з можливістю виконання складних запитів, використання бібліотек тестування роботи з БД та зручне керування даними.

Робочий сервери одного клієнта забороняють існування більше ніж одного підключення клієнту в момент часу. Це допоможе запобігти перевантаженню серверів та уникати конфліктів даних.

Дизайн додатку в «React.js» базується на ідеї компоненто-орієнтованої архітектури, яка дозволяє будувати масштабовані, динамічні й повторно використовувані інтерфейси [27]. Основою дизайну є поділ інтерфейсу на невеликі, незалежні модулі(компоненти), кожен з яких відповідає за конкретний елемент або функціональність додатку.

Використання JSX: дозволяє описувати структуру інтерфейсу у вигляді HTML-подібного синтаксису напряму в JS [28]. Це спрощує сприйняття коду, адже логіка і відображення знаходяться в одному файлі. Розбиття на рівні: кореневий компонент, сторінки або великі секції й менші під-компоненти. Для прикладу, кореневий компонент (App) може містити маршрутизатор для управління сторінками, а кожна сторінка – набір функціональних модулів, таких як панель навігації. Модульна організація: модулі зазвичай організовуються у вигляді окремих файлів або папок, що дозволяє ізолювати компоненти та їх стилі, зменшити взаємозалежність між ними та

підтримувати розташування елементів проекту інтуїтивно зрозумілим. Функціональні або класові компоненти: у сучасних додатках «React» віддається перевага функціональним компонентам завдяки використанню хуків, таких як «useState» та «useEffect». Це спрощує код, оскільки немає необхідності працювати з класами й методами життєвого циклу.

Використання хуків: хуки забезпечують простий і декларативний підхід до управління станом і побічними ефектами. Наприклад, «useState» дозволяє створити змінні стану, зміна яких автоматично оновлює відображення.

Логіка відображення: JSX дає змогу створювати динамічну логіку відображення. Наприклад, для відображення списку подій можна використовувати методи «.map()» або «.flatMap()» для генерації HTML-коду.

Маршрутизація та динаміка: для реалізації навігації в додатку використовується бібліотека «react-router-dom», яка дозволяє створювати маршрути, що відповідають за різні сторінки додатку.

Інтерактивність: реактивність додатка забезпечується через події та стан. Наприклад, кліки на кнопки, введення даних у форму чи вибір фільтрів змінюють локальний або глобальний стан додатку та автоматично оновлюють інтерфейс, без необхідності перезавантажувати сторінку.

Обмін даними між компонентами: для передачі даних між компонентами використовуються «props». Якщо додаток має складну структуру, то для управління станом на більш високому рівні використовуються контексти (React Context) або зовнішні бібліотеки для керування станом, такі як Redux.

Побудова стильових рішень: для стилізації використовуються модулі «SASS» (можливо використання «SCSS» або напряму «CSS»). Кожен компонент має свої стилі у файлі «[Component].sass».

# 3 ОПИС ВИКОРИСТАНИХ ТЕХНОЛОГІЙ

## 3.1 Система управління нереляційною базою даних MongoDB

MongoDB є основою для зберігання даних у стеку «MERN» та належить до класу документно-орієнтованих баз даних «NoSQL». Це означає, що замість традиційних реляційних таблиць, MongoDB оперує документами у форматі «BSON» (Binary JSON), які зберігаються у колекціях. Такий підхід не вимагає попереднього визначення схеми для колекцій. Це дозволяє розробникам зберігати документи з різною структурою в одній колекції, що особливо корисно при роботі з динамічними даними або такими, що розвиваються в процесі використання. Така гнучкість сприяє швидкому прототипуванню та адаптації до змін у вимогах клієнта.

«MongoDB» підтримує горизонтальне масштабування через механізм шардінгу, який дозволяє розподіляти дані по кількох серверах. Крім того, система реплікації забезпечує високу доступність даних, через створення копії БД на різних вузлах, що дозволяє автоматично переключатись на резервний вузол у разі відмови основного. Для обробки та трансформації даних «MongoDB» пропонує потужний агрегаційний фреймворк, який дозволяє виконувати складні операції над даними, такі як групування, обчислення, фільтрація та перетворення. Значним доповненням є підтримка транзакцій. Починаючи з версії 4.0, «MongoDB» підтримує багатодокументні транзакції, для забезпечення атомарності операцій, подібно до реляційних баз даних. Це важливо для додатків зі складно конвеєрною логікою.

Також «MongoDB» має ефективні засоби індексації та пошуку. Вона дозволяє створювати індекси на будь-яких полях документів, включаючи вкладені поля та масиви. Це значно покращує продуктивність запитів, особливо при роботі з великими обсягами даних. Крім того, підтримується повнотекстовий пошук.

Для безпосередньої взаємодії користувача з базою даних використовується «MQL» – це мова запитів, розроблена спеціально для «MongoDB», яка дозволяє взаємодіяти з базою даних через виконання операцій створення, читання, оновлення та видалення документів (CRUD). Синтаксис «MQL» базується на JSON-подібних структурах, що робить його зручним для розробки, особливо при взаємодії з JavaScript. Завдяки чому можливо ефективно працювати з даними, реалізуючи складну бізнес-логіку без необхідності використання сторонніх засобів.

## 3.2 Фреймворк Express.js

«Express.js» спроектований як мінімалістичний фреймворк, що означає наявність лише базових функцій у ядрі. Це дозволяє розробникам додавати необхідні можливості через використання проміжного програмного забезпечення (middleware), забезпечуючи гнучкість та контроль над архітектурою додатку. В Express.js розвинена система маршрутизації, яка дозволяє обробляти HTTP-запити з використанням методів, таких як GET, POST, PUT, DELETE та інших, у поєднанні з визначеними шляхами.

В Express.js є підтримка «middleware» – функцій, які мають доступ до об'єктів запиту (req), відповіді (res) та наступної функції в циклі обробки. Middleware дозволяє виконувати широкий спектр завдань, включаючи попередню обробку запитів, автентифікацію користувачів, ведення журналів подій, обробку помилок, тощо. «Express.js» також підтримує інтеграцію з різними шаблонізаторами, такими як «Pug», «EJS», «Handlebars» та іншими для генерування динамічних HTML-сторінок на стороні сервера. Це особливо корисно при створенні веб-додатків з рендерингом інтерфейсу на сервері.

«Express.js» також спрощує обслуговування статичних файлів – таких як зображення, CSS та JavaScript. Це дозволяє швидко і просто організувати доступ до ресурсів інтерфейсу користувача без додаткового налаштування серверної логіки.

## 3.3 Бібліотека React

«React.js» – бібліотека JavaScript для створення користувацьких інтерфейсів. Розроблена компанією «Meta» (раніше «Facebook»), React.js дозволяє будувати інтерфейси з окремих компонентів, що сприяє модульності та повторному використанню коду. «React.js» базується на концепції компонентів, які є незалежними та повторно використовуваними частинами інтерфейсу. Ці компоненти можуть бути функціональними або класовими, хоча сучасна практика віддає перевагу функціональним компонентам з використанням хуків («Hooks»). Хуки, такі як «useState», «useEffect», «useMemo» та «useCallback», дозволяють керувати станом та побічними ефектами в компонентах без необхідності використання класів.

Однією з ключових особливостей «React.js» є використання віртуального DOM, який оптимізує оновлення інтерфейсу, мінімізуючи кількість змін у реальному DOM. React.js також підтримує декларативний підхід до програмування, де розробник описує, як інтерфейс повинен виглядати в певному стані, а React.js самостійно його оновлює за зміни стану.

«React» є «базовою» бібліотекою для побудови користувальницького додатку та не має складного спеціалізованого функціоналу. Тому було створено бібліотеки для розширення його функціоналу. Наприклад для реалізації перетягування елементів у «React.js» часто використовується бібліотека «dnd-kit», а для маршрутизації односторінкових сайтів – «React Router».

### 3.3.1 Бібліотека-доповнення dnd-kit

Бібліотека «dnd-kit» – це сучасний, модульний і потужний набір інструментів для реалізації функціональності перетягування у додатках, побудованих на «React.js». Вона створена з урахуванням сучасних стандартів веб-розробки та оптимізована для гнучкості, розширюваності й доступності.

Однією з основних переваг «dnd-kit» є його архітектурна побудова, що базується на реактивних хуках, що робить інтеграцію максимально природною для розробників, які працюють з «React». Бібліотека пропонує набір API, які можна комбінувати, налаштовувати і розширювати відповідно до специфічних потреб проекту, для реалізації як простих, так і складних сценаріїв перетягування з підтримкою таких функцій, як сортування, перенос між різними контейнерами, вкладені «draggable» елементи та багато іншого.

«dnd-kit» дотримується принципів доступності (accessibility) та підтримує стандартні методи керування клавіатурою, що особливо важливо для користувачів із обмеженими можливостями [29].

Відзначається також ретельна оптимізація продуктивності: бібліотека використовує оптимізовані механізми рендерингу та мінімізує повторні обчислення, що важливо для додатків із великими наборами елементів. Окрім того, «dnd-kit» підтримує мобільні пристрої. Також інструментарій «dnd-kit» добре документований і має активну спільноту, що постійно розвиває бібліотеку та доповнює її новими можливостями.

### 3.3.2 Бібліотека-доповнення React Router

«React Router» – це одна з ключових бібліотек екосистеми «React», яка надає засоби для організації маршрутизації в SPA. Вона дозволяє створювати навігацію між різними представленнями або сторінками інтерфейсу користувача без необхідності перезавантаження сторінки, зберігаючи при цьому всі переваги клієнтського рендерингу та гнучкість реактивного підходу до побудови інтерфейсів.

«React» сам по собі не включає вбудованих засобів для маршрутизації, тому «React Router» виступає як стандартна де-факто бібліотека, яка реалізує цей функціонал. Вона забезпечує декларативний підхід до визначення маршрутів, тобто дозволяє описувати, які компоненти повинні відображатися при певному URL-шляху, як частину структури JSX.

«React Router» підтримує різні типи навігації: як історичну (history API), так і хеш-навігацію (hash-based), що дозволяє адаптуватися до різних серверних конфігурацій. У більшості сучасних застосунків використовується «BrowserRouter», який працює з історією браузера, імітуючи повноцінну навігацію між сторінками, хоча насправді переходи відбуваються без оновлення документа (DOM).

Також можливо реалізувати вкладену маршрутизацію, яка дозволяє створювати складні ієрархії сторінок або візуальних шарів всередині головного шаблону. Наприклад, в рамках одного проекту можна реалізувати маршрути для загальних сторінок (типу /projects), а також вкладені маршрути (наприклад, /project/projectId/activityGroup/ActivityId), кожен з яких буде відображати свою частину інтерфейсу без конфлікту з іншими компонентами. Також «React Router» дозволяє використовувати «динамічні параметри» у шляхах, завдяки чому можна створювати маршрути, що адаптуються до даних, які отримує користувач.

Ще однією важливою перевагою є підтримка програмної навігації. За допомогою хуків, таких як «useNavigate», можна змінювати шлях у відповідь на певну подію – наприклад, перенаправити користувача після входу або реєстрації. Крім того, «React Router» інтегрується з механізмами контексту «React» (Context API), що дозволяє поширювати інформацію про активний маршрут між компонентами та забезпечує централізовану логіку навігації. Також підтримується «lazy loading», яке дозволяє завантажувати компоненти маршрутів динамічно лише тоді, коли вони справді потрібні.

«React Router» активно підтримується і оновлюється спільнотою та командою розробників, постійно адаптуючись до змін у самому «React». Починаючи з версії 6, бібліотека зазнала значного оновлення, яке включало спрощення синтаксису, нові концепції, як-от «route objects», покращену підтримку вкладених маршрутів та використання хуків як стандартного способу взаємодії з маршрутизатором.

## 3.4 Платформа виконання коду Node.js

«Node.js» – це платформа для виконання JavaScript-коду поза браузером, створена на основі движка V8 від Google, який використовується у «Chrome». Вона дозволяє запускати серверні додатки з високою продуктивністю та масштабованістю мовою програмування JavaScript.

Однією з ключових особливостей «Node.js» є його подієва (event-driven), неблокуюча (non-blocking) модель вводу-виводу, заснована на циклі подій (event loop). Така архітектура дозволяє ефективно обробляти велику кількість одночасних підключень без створення нових потоків, що суттєво знижує витрати ресурсів сервера. Завдяки цьому «Node.js» особливо підходить для розробки веб-додатків реального часу, API-серверів, чатів, ігор та інших систем із високими вимогами до паралельної обробки запитів.

«Node.js» надає потужний набір вбудованих модулів, які полегшують роботу з файловою системою, мережею, потоками даних, процесами, а також підтримує створення HTTP-серверів без додаткових бібліотек. Крім того, Node.js має розвинену екосистему пакетів через менеджер пакетів «npm» (Node Package Manager), який дозволяє легко встановлювати та використовувати тисячі бібліотек для різних завдань – від роботи з базами даних до створення складних мікросервісних архітектур.

У контексті стеку «MERN», Node.js виконує роль серверної платформи, на якій працює серверна частина додатку, забезпечуючи логіку обробки запитів, взаємодію з базою даних «MongoDB» через «Express.js», а також зв’язок із клієнтською частиною, реалізованою на «React». Використання Node.js дозволяє застосовувати єдину мову програмування – JavaScript – як на клієнті, так і сервері, що спрощує розробку, підтримку та масштабування.

Додатково до «Node.js» часто інтегрують різні фреймворки та бібліотеки для валідації, обробки даних та управління бізнес-логікою. Наприклад бібліотека «Joi», яка дозволяє описувати схеми даних і перевіряти їх коректність перед збереженням або обробкою.

# 4 ОГЛЯД АЛГОРИТМІВ, АРХІТЕКТУРНИХ РІШЕНЬ

## 4.1 Front-end

### 4.1.1 Шаблони проектування

У клієнтському додатку застосовано різні шаблони для організації компонентів і логіки. По-перше, помітний шаблон «підняття стану вгору» (lifting state up) [30]. Коли декілька дочірніх компонентів потребують доступу до одного й того ж стану або мають узгоджувати його, стан виноситься до найближчого спільного предка, який передає його вниз через параметри (props). Такий підхід робить компонент батьківської ланки «джерелом правди» (source of truth) для спільного стану, чим забезпечує узгодженість значень у дочірніх компонентах. Крім того, застосовано композиційну модель з передачею дітей через «props.children», що є класичним шаблоном для «React». Це підтверджено офіційною документацією «React»: використання «props.children» дозволяє компонування без застосування складної ієрархії наслідування [31]. «React» наполягає на композиції замість спадкування, оскільки композиція (через props, об’єкти React або функції) дає необхідну гнучкість без ризику зміни ієрархії компонентів [32].

Також у коді є відокремлення на контейнерні та презентаційні компоненти, що відповідає шаблону «Container-Presentation». Контейнерні компоненти відповідають за отримання даних або обробку бізнес-логіки, тоді як презентаційні компоненти відповідають лише за відображення інтерфейсу (render). Це слідує принципу розділу відповідальності (SRP), що спрощує тестування та повторне використання UI-компонентів. Даний підхід запобігає утворенню «монолітних» компонентів, в яких змішуються логіка та відображення. Шаблон «Container-Presentation» забезпечує модульність і полегшує підтримку коду.

Для повторного використання поведінки застосовано шаблон HOC (Higher-Order Component) або аналогічні підходи, які «вводити» загальну логіку в компоненти. HOC – це функція, яка приймає компонент і повертає розширений компонент із додатковою функціональністю. Це також логічно випливає з того, що React віддає перевагу композиції над спадкуванням, використовуючи HOC як механізм розширення без створення спеціальних ієрархій компонентів. Альтернативою HOC можуть бути шаблон «Render Props» чи кастомний хук, проте HOC дозволяє «вводити» властивості та функції в компонент без змін його вихідного коду.

У частині керування станом застосовується модель контрольованих компонентів. Дані форм та полів вводу зберігаються в стані компонента, а зміни обробляються через «onChange» обробники. Це підтверджує використання описаного патерну «Controlled Components»: такий підхід дозволяє «React» контролювати стан і робить UI передбачуваним і простішим для налагодження. Альтернативно, неконтрольовані компоненти, що покладаються на пряму взаємодію з DOM, ускладнили б підтримку та могли б викликати неявні побічні ефекти. Тому вибір контрольованого патерну робить код читабельним, а його виконання передбачуваним.

Також використано шаблон «введення залежностей» (DI, dependency injection), який вже згадувався раніше. Хоча «React» не має явного механізму «DI», залежності (наприклад, сервіси або утиліти) передаються через props або через «Context API» як «глобальні» провайдери. Наприклад, якщо компоненти мають використовувати один і той же сервіс, його можна створити в батьківському компоненті та передавати всім потрібним нащадкам через props (або через контекст), що аналогічно «DI» залежностей. Офіційна рекомендація полягає в тому, що «Context API» дозволяє надавати залежності всім компонентам у дереві, не виконуючи їх «prop drilling». Таким чином, замість множинного передачі одних і тих же «props» вниз по ієрархії (prop drilling), більш доцільно створити «Context.Provider», що забезпечить доступ до потрібних даних у будь-якому глибинному компоненті.

### 4.1.2 Алгоритми

У коді клієнтського та адміністраторського додатків використовуються звичайні алгоритмічні підходи при обробці даних. Наприклад, для сортування або реорганізації списків може застосовуватися стандартний метод «Array.sort()», який працює за середньою складністю O(n log n). При цьому важливо не змінювати оригінальний масив стану: у «React» для сортування спочатку створюють копію масиву (наприклад, через оператор розпакування [...array], slice(), або structuredCopy()) і вже до неї застосовують sort(), щоб не змінювати стан безпосередньо [33]. Подібно, вставку або видалення елементів зі списку виконують через непомутнюючі методи (pure methods): замість splice() (мутуючий метод) часто використовують slice(), filter() або concat().

Щодо навігації, клієнтський та адміністраторський додатку використовують декларативну маршрутизацію (react-router-dom). У «React Router» маршрути відображають компоненти залежно від поточного URL без повного перезавантаження сторінки [34]. Основний алгоритм роботи маршрутизатора полягає у порівнянні поточного шляху з набором оголошених маршрутів і рендерингу відповідного компоненту. Цей процес здійснюється на клієнті та синхронізує UI з URL, що дозволяє реалізувати SPA зі збереженням історії браузера.

Збереження стану програми реалізовано за допомогою хуків «useState»: кожна зміна стану зберігається у внутрішньому об’єкті компонента і при її оновленні «React» викликає повторний рендер відповідних компонентів. Якщо використовується «useEffect», то логіка збереження може розміщуватися в таких ефектах (наприклад, зберігати дані в localStorage або викликати API). Зазначений шаблон роботи та оновлення стану відповідає стандартному підходу «React» – оновлювати стан через «pure methods» та викликати «функцію-setter» або диспетчер дій, що призводить до оновлення інтерфейсу.

### 4.1.3 Загальна логіка

Додатки побудовані за принципом одностороннього потоку даних (unidirectional data flow) [35]. Батьківські компоненти передають дані та обробники подій у дочірні через «props», а дочірні викликають «callbacks» при події (наприклад, натисканні чи завершенні перетягування), повідомляючи батьківські компоненти про необхідність зміни стану. Це підтверджує, що всі зміни стану централізовано обробляються «вгору» по дереву, а дочірні компоненти не змінюють стан напряму. Події обробляються через «React Synthetic Events»: функції-обробники (onClick, onDragEnd і т.ін.) пов’язані з елементами JSX і при спрацьовуванні викликають зміни стану у відповідних методах. Після зміни стану за допомогою «setState» та інших змінних стану, «React» повторно рендерить ті компоненти, чиї «props»/стан змінились, оновлюючи DOM. Важливо, що програмна логіка побудована із урахуванням реактивності – зміни стану негайно відображаються в UI, а дані передаються з компонентів в компоненти явно через props або контекст.

У коді визначено низку абстрактних компонентів та утиліт, призначених для повторного використання. Наприклад, є загальні UI-компоненти (кнопки, поля введення, списки), які використовуються у різних місцях додатків. Ці компоненти розроблені таким чином, щоб приймати параметри через «props» і бути перевикористовуваними в різних контекстах. Окремі компоненти винесені у вигляді компонентів вищого порядку (HOC) або хук-компонентів, що інкапсулюють спільну логіку. Наприклад, якщо є складний інтерфейс «drag-and-drop», то у проекті реалізовано обгортки (компоненти «ItemList», «Item») навколо елементів (через HOC або кастомний хук useDraggable), щоб у кожному компоненті не дублювати одну й ту ж конфігурацію «drag-and-drop». При появі схожої логіки у трьох і більше місцях її виділено у спільний компонент чи HOC [36]. Крім того, шаблон «Container-Presentation» також створює рівень абстракції: загальний контейнер може використовувати різні презентаційні компоненти для

виведення. Наприклад, може існувати єдиний компонент-список, який рендериться з різними набором даних (через «props»), а логіка завантаження даних чи фільтрації винесена в контейнерний компонент.

## 4.2 Back-end

### 4.2.1 Шаблони програмування

У коді серверів використано традиційні шаблони:

- Singleton;
- Factory;
- MVC (Model-View-Controller);
- Observer;
- Middleware (Chain of Responsibility);
- Module (Encapsulation);
- Service Layer;
- DAO (Data Access Object).

Шаблон «Singleton» реалізовано через кешування модулів Node.js: при імпорті одного й того ж модуля (наприклад, підключення до бази даних) завжди повертається один екземпляр.

Шаблон «Factory» використовується для створення об’єктів (наприклад, екземплярів сервісів чи моделей) без прив’язки до конкретного класу.

У структурі коду проглядається шаблон «MVC»: моделі (Mongoose) описують дані (Model), контролери/маршрути Express обробляють HTTP-запити (Controller), а у випадку REST API «View» – це JSON-відповіді. Сам шаблон MVC сприяє чіткому розділенню обов’язків і повторному використанню коду.

Шаблон «Observer» реалізовано через подійну модель Node.js (EventEmitter) та «WebSocket.

Шаблон Middleware широко використовується в Express для обробки запитів ланцюжком функцій (для логування, автентифікації тощо).

Модульний шаблон прослідковується у виділених файлах- модулях, які експортують публічний API та інкапсулюють дані і логіку всередині.

Нарешті, виділено «сервісний шар» і шар доступу до даних (DAO): маршрути викликають сервіси з бізнес-логікою, а вони – DAO для звернень до MongoDB (або навпаки). Це відповідає рекомендаціям шарової архітектури (Router–Service–DAL) для розділення відповідальностей.

### 4.2.2 Алгоритми

Обхід дерева: якщо дані мають ієрархічну структуру (наприклад, вкладені категорії), логіка може використовувати рекурсію (DFS/BFS) для обходу всіх вузлів. Типово це реалізується функцією, яка рекурсивно обходитиме вкладені об’єкти або здійснюватиме запити з «$graphLookup$» у «MongoDB».

Диференціація JSON-об’єктів: для порівняння «старої» та «нової» версій даних при оновленні застосовувються JSON-дописи. Прості підходи використовують «JSON.stringify(obj1) === JSON.stringify(obj2)» для невеликих об’єктів. Для глибшої перевірки часто використовують бібліотеки (наприклад, deep-diff) чи власні алгоритми рекурсивного порівняння ключів. Це дозволяє знаходити різницю між об’єктами і застосовувати мінімальні змінні у БД, що також спрощує обробку паралельних запитів через зменшення кількості накладок (overlaps).

Контроль версій (__v): Mongoose додає поле «__v» до документа для оптимістичної конкуренції. При паралельних оновленнях API під час запису враховується поточна версія: запит оновлення включає умову «__v": поточне_значення», а після успіху версія інкрементується (наприклад, $inc: {__v:1}). Таким чином, якщо версія не збігається (інший запит вже змінив документ), оновлення не відбудеться і викликається помилка.

Контроль доступу: реалізовано ролевий доступ (RBAC) через проміжні методи (middleware). Кожен запит перед обробкою в контролері проходить перевірку аутентифікації (наприклад, перевірка JWT або сесії) і авторизації за роллю. Ідея «RBAC» у «Node.js» – розмежувати права доступу: наприклад, роль «admin» має всі права, «editor» – обмежені до редагування, а «viewer» – лише читання. Це забезпечує безпечний доступ до різних ресурсів через механізми middleware, які аналізують токен та параметри ролі користувача і відхиляють недозволені запити на ранніх етапах обробки.

### 4.2.3 Загальна логіка

Загальна архітектура – мультишарова. Шар даних представлено моделями Mongoose (MongoDB ODM) із визначеними схемами. Шар логіки («сервіси») виконує перетворення і перевірки даних, які потім передаються в «DAO» для збереження чи отримання. Шар контролерів/маршрутів на базі Express встановлює кінцеві точки API і делегує запити сервісам. Відповідно до «Layered Architecture», маршрутизатор викликає сервіс, а той – шар доступу до даних.

У коді для кожної сутності, є «обробник додавання/редагування» (наприклад, «handleAddEdit»), який вирішує, чи створювати новий документ або оновлювати існуючий, використовуючи методи DAO і контролюючи версію (__v). Функція «getData» повинна вибирати і повертати дані з БД з урахуванням фільтрів і пагінації. «updateItemFields» виконує перевірку придатності та покрокове оновлення полів документа: тут застосовуються алгоритми диференціації JSON та рекурсивний пошук оновлених полів (наприклад, для вкладених структур).

Усі «HTTP» методи супроводжуються проміжною обробкою: middleware для автентифікації/авторизації (перевірки токена чи сесії), валідації даних, логування збережених змін та сповіщень про помилки, і централізованої обробки помилок.

Важливим елементом є «WebSocket» – реалізований із використанням бібліотеки «ws», «WebSocket» забезпечує двосторонній зв’язок у реальному часі між клієнтом та сервером. Логіка обробки WebSocket-повідомлень зазвичай винесена в окремий модуль або middleware, що повторно використовує DAO/сервіси.

## 4.3 Правила іменування та конвенції

У проекті спостерігається типовий стиль іменування для «React» [37]:

- класи та «REST API»: бібліотечні CSS-класи та шляхи «REST API» часто дотримуються певного стилю, в даному випадку «kebab-case»;
- компоненти: назви компонентів використовуються в «PascalCase» (кожне слово з великої літери), що дозволяє чітко відрізняти компоненти від звичайних HTML-елементів;
- змінні та методи: пишуться в «camelCase» (наприклад, handleSubmit, userName), оскільки це звичний стиль для JS. Хоча узгодженість стилю є важливою, константи, відомі до виконання (hard-coded значення), позначаються у «SCREAMING_SNAKE_CASE» [38].

Також, для підтримки читабельності коду, у проекті використано наступні правила оформлення коду:

- використання тільки «arrow-functions»: підтримка послідовності методів в коді відповідно до їх залежностей;
- визначений порядок стилів класу (розмір, відступи, відображення та положення, межі, кольори, текст, решта): спрощення тестування використовуваних стилів;
- іменування та вміст файлів: кожен файл може містити тільки один елемент верхнього рівня (компонент, хук, чи метод) та назва файлу повністю повторює назву зазначеного елементу.
- розбиття довгих строк: методи у ланцюгах та опції операторів розгалуження з нової строки.

# 5 ІНСТРУКЦІЯ ДО ВИКОРИСТАННЯ

## 5.1 Інструкція для користувача

### 5.1.1 Вхід та реєстрація

При вході і додаток користувач потрапляє на стартову сторінку «Home Page» (рисунок 5.1). Далі в заголовку сторінки потрібно обрати «Log In/Register» що відкриє відповідний діалог (рисунок 5.2). На початку діалогу знаходиться перемикач між режимом реєстрації та входу (за замовчуванням обрано «Log in»). Для входу/реєстрації потрібно ввести дані користувача та обрати сервер, до якого потрібно підключитись.


Рисунок 5.1 – Placeholder стартової сторінки клієнтського додатку

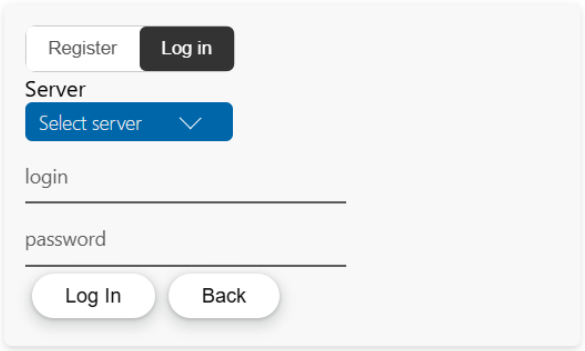



Рисунок 5.2 – Діалог входу до облікового запису

### 5.1.2 Панель навігації

Панель навігації відображає список доступних користувачу проектів та їх активностей (рисунок 5.3). При натисканні на назву проекту буде відкрито/приховано його активності. При натисканні на активність, відкривається відповідний проект та прокручується до обраної активності. Протягом сесії відвідані активності зберігаються у списку «нещодавно відвіданих» (recent). Також є можливість очистити список «нещодавно відвіданих» активностей. При відвідування хоча б однієї активності, одразу після списку «нещодавно відвіданих» з’являється кнопка для його скидання

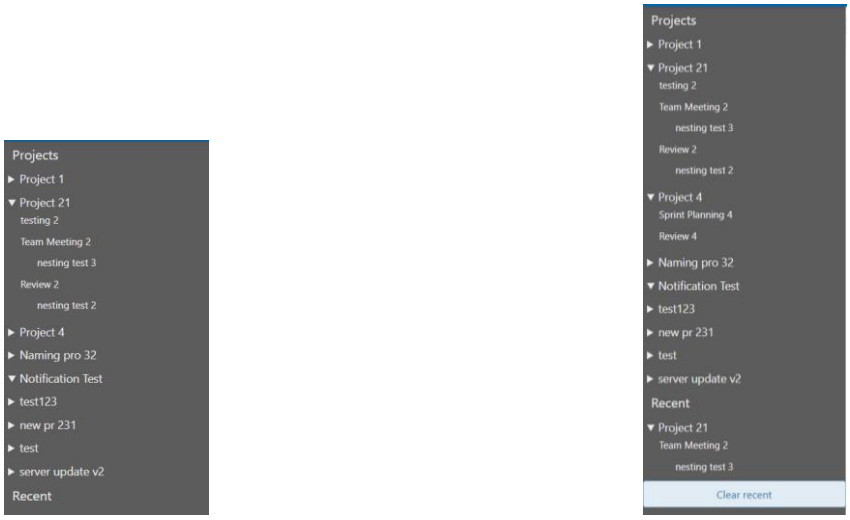
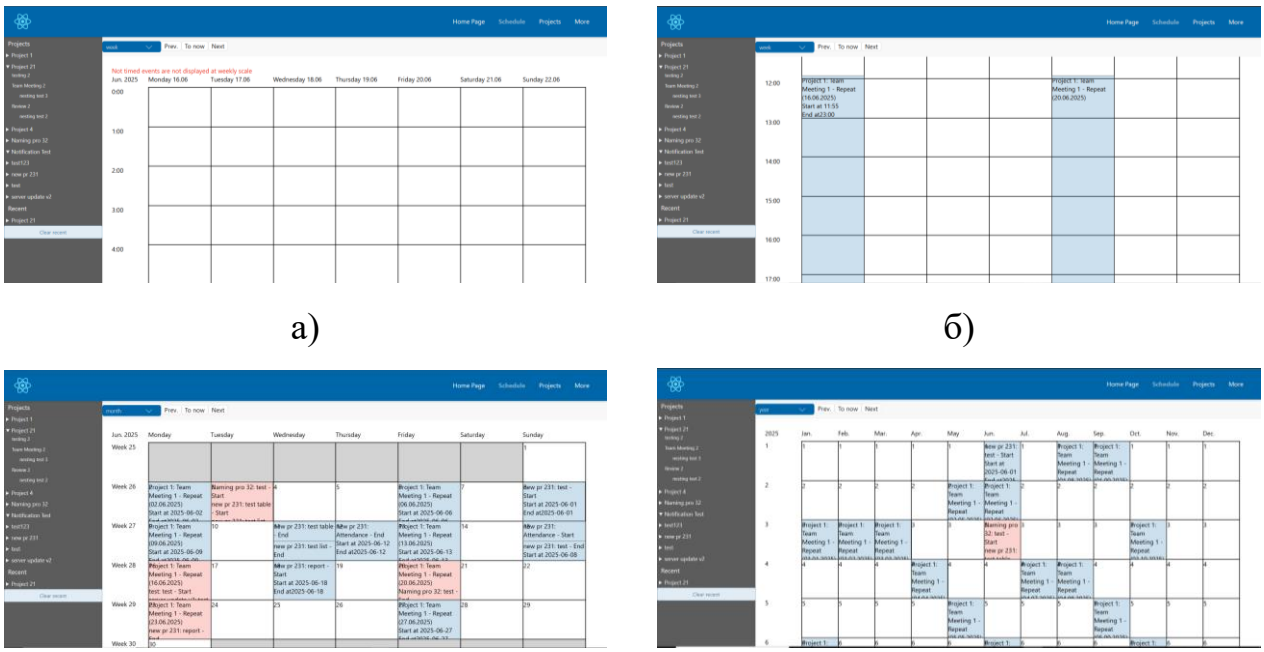



Рисунок 5.3 – Панель навігації: а) без списку «нещодавно відвіданих» активностей; б) зі списком «нещодавно відвіданих» активностей

### 5.1.3 Розклад

Розклад у додатку знаходиться на однойменній сторінці. Розклад являє собою список посилань на проекти та активності та має три масштабу відображення: тиждень, місяць, рік (рисунок 5.4).



а) б)

в) г)

Рисунок 5.4 – Сторінка розкладу: а-б) масштаб «тиждень»;
в) масштаб «місяць»; г) масштаб «рік»

За кожного масштабу користувач може пересуватись розкладом аналогічно до використання календаря. Панель інструментів завжди залишається на горі сторінки одразу після заголовку(якщо є доступні інструменти). Масштаби відображення мають обмеження: події, що прив’язані тільки до дати, а не часу, не відображаються при масштабі «тиждень».

Кожна подія що відображається у розкладі містить:

- назва проекту;
- назва активності (за наявності);
- тип події (початок/повтор/кінець, за наявності);
- час початоку (за наявності);
- час завершення (за наявності).

В розкладі також передбачено «накладки» подій. В таких випадках накладка двох подій відображається через розподіл місця між подіями, а якщо більше двох – як діалог зі списком події, що накладаються (рисунок 5.5).

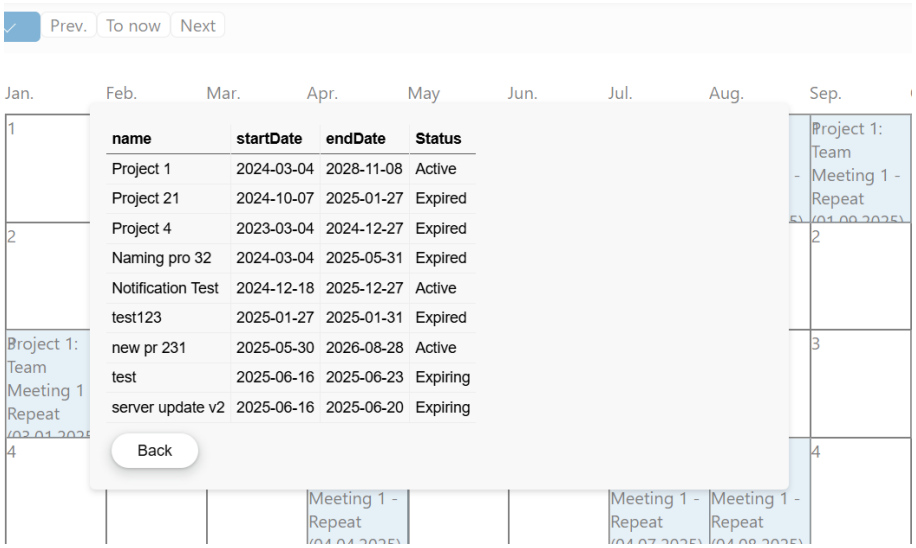
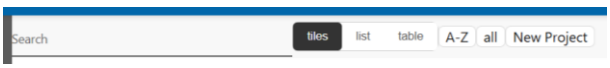



Рисунок 5.5 – Список подій, що накладаються

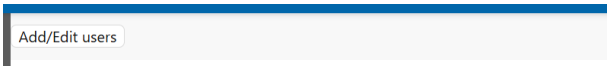
### 5.1.4 Редактор проектів та активностей

Редактор проектів та активностей є найбільшою частиною додатку та доступний на сторінці «Projects» (або «Subjects»). Панель інструментів містить пошук, перемикач типів відображення, фільтри та кнопку для виклику діалогу створення проекту/активності (рисунок 5.6). Пошук працює як звичайне поле вводу та виконує пошук за будь-яких змін його вмісту. Фільтри та сортування доступні при відображенні НЕ як таблиця, оскільки таблиці мають вбудоване сортування, про це пізніше. Також фільтри та сортування доступні тільки для проектів. При роботі з активностями панель інструментів надає доступ до редагування доступу користувачів до проекту.




а)

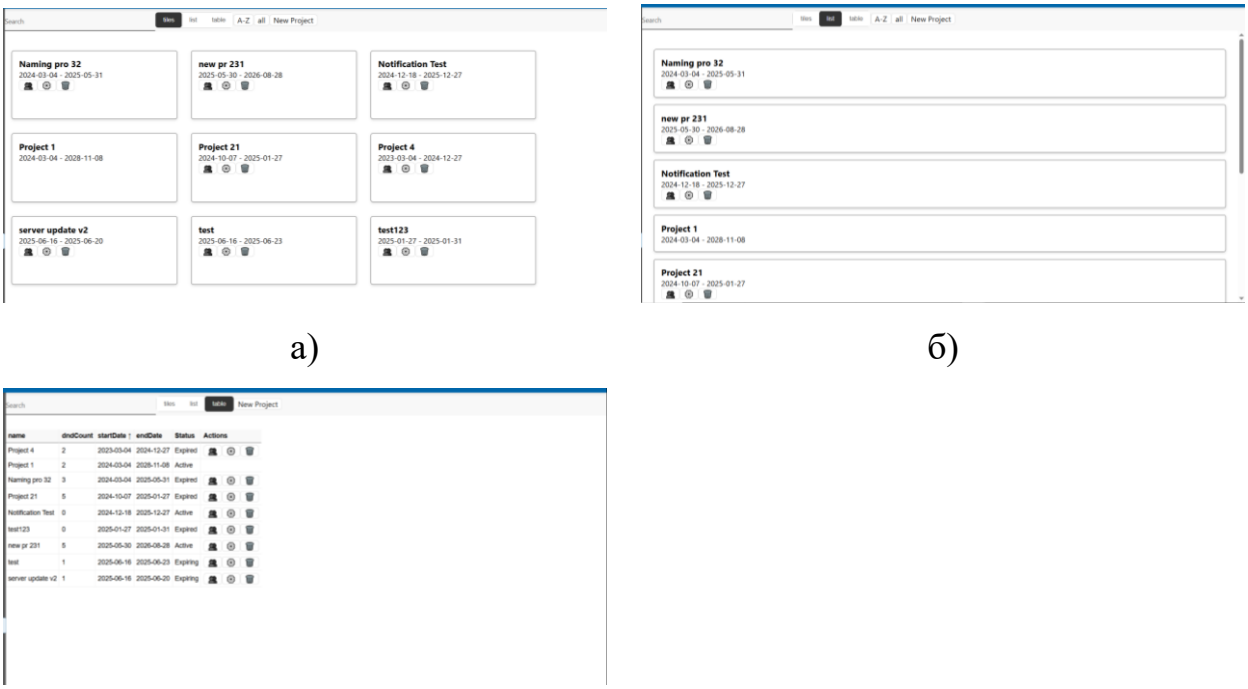

б)




в)

Рисунок 5.6 – панель інструментів головного редактора: а) для проектів; б) для табличного відображення проектів; в) для активностей

Редактор проектів надає три представлення списку проектів (рисунок 5.7), можливості їх сортування, фільтрування та пошуку за назвою. Також всі проекти, як і активності, за наявності у користувача прав на редагування містять інструменти для: редагування прав користувачів, зміни налаштувань та «видалення» елементу (рисунок 5.8). Діалоги додавання та редагування проектів аналогічні (рисунок 5.9).



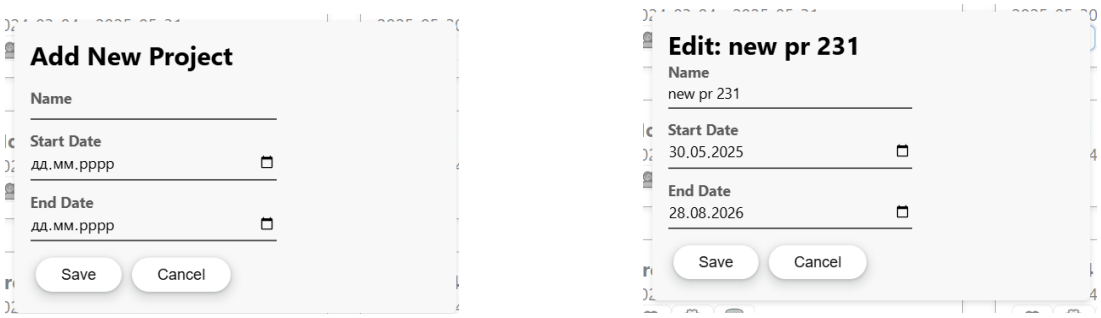

а) б)

в)

Рисунок 5.7 – Типи відображення списку проектів:
а) картки; б) список; в) таблиця

а) б) в)

Рисунок 5.8 – Інструменти взаємодії з елементами: а) для проектів;
б) для активностей; в) для елементів списку




а) б)

Рисунок 5.9 – Діалог: а) додавання проекту; б) редагування проекту

Редагування прав доступу користувачів відбувається через діалог, який фактично складається з двох таблиць: користувачі з доступом та користувачі без доступу (рисунок 5.10). При наданні/видаленні та зміні рівня прав доступу користувача зміни зберігаються одразу.

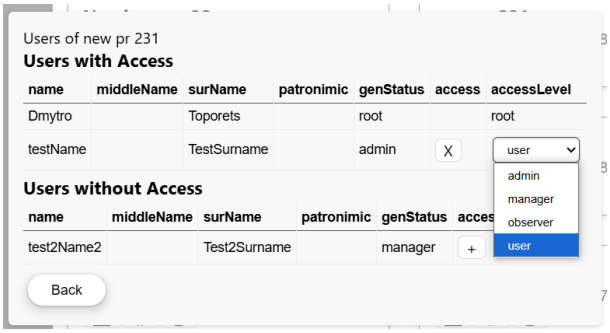



Рисунок 5.10 – Діалог редагування прав доступу

Редактор підтримує сім типів активностей: «група», «текст», «список», «таблиця», «відвідування», «звіт», «чат». З яких «таблиця», «відвідування», «звіт», «чат» засновані на списку. Для наглядності в прикладах активностей можуть зустрічатись активності типу «Dev». Вони є виключно прикладом та використовуються для тестування, тому звичайні користувачі не мають можливості їх створювати (рисунок 5.11).


Рисунок 5.11 – Тестові активності

«Група»: елемент-контейнер (рисунок 5.12). Дані активності є контейнерами для інших активностей. Вони містять назву, та власне список активностей. Можна використовувати для візуального розмежування та розділу за доступом. Кожна група на початку містить опцію «згорнути/розгорнути» яка приховує/показує вкладені активності. Групи, як і елементи всередині можна перетягувати (DnD) між собою і перенос елементів між групами можливий, за наявності дозволу на редагування обох груп.

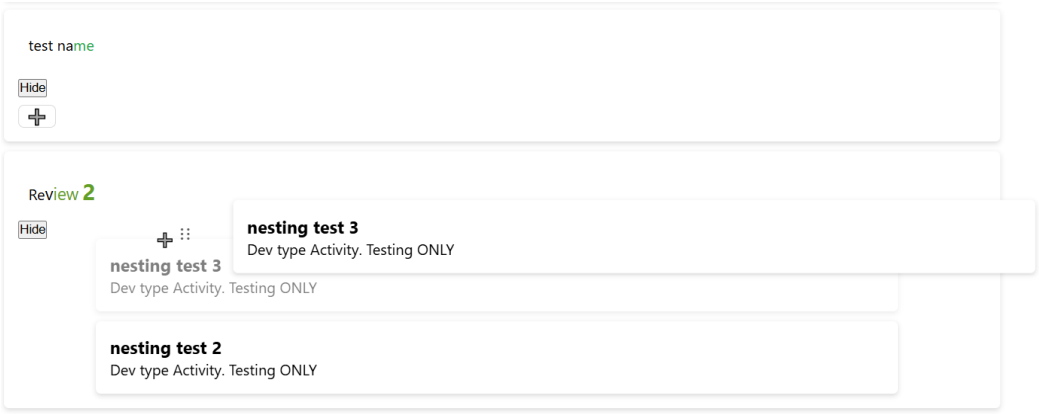
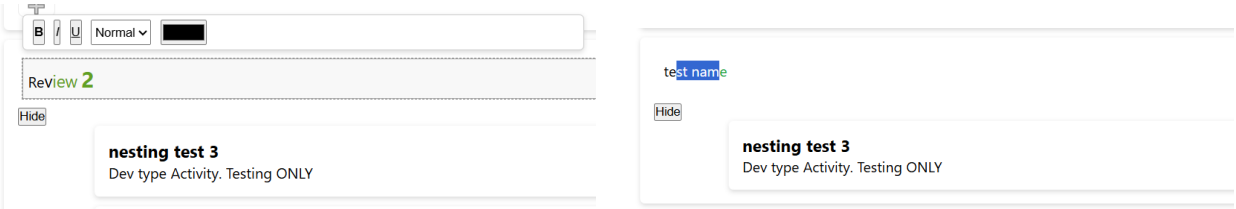



Рисунок 5.12 – «Групи»

«Текст» : елемент текстового поля, який часто є частиною інших активностей (рисунок 5.13). Даний елемент, залежно від прав користувача, є або звичайним текстом, або простим текстовим редактором, який підтримує редагування наступних стилів: жирність, курсив, підкреслювання, розмір та колір. Зміни внесені до тексту зберігаються при виході з режиму редагування (клік поза редактором).



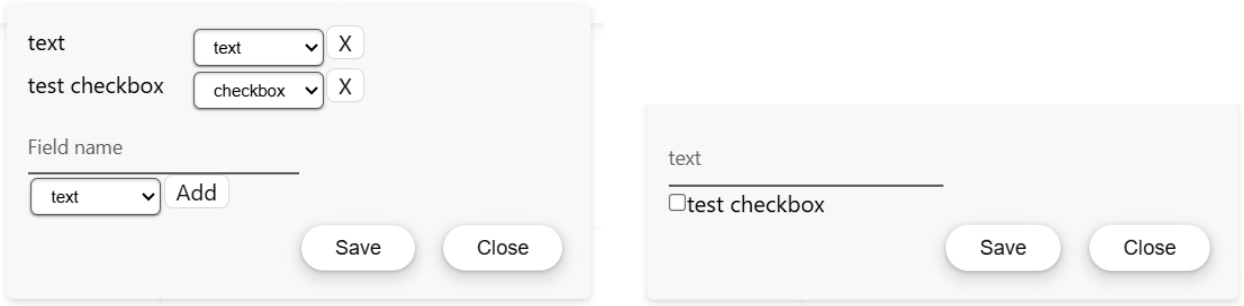

а) б)

Рисунок 5.13 – «Текст»: а) як редактор; б) як звичайний текст

«Список» : елемент списку. За наявності прав на редагування, користувач може додавати елементи до списку, редагувати їх структуру та змінювати порядок перетягуванням (DnD)(рисунок 5.14-15). Також на початку списку наявний перемикач для включення/відключення нумерації елементів списку.


а) б)

Рисунок 5.14 – «список»: а) розгорнутий; б) згорнутий




а) б)

Рисунок 5.15 – діалоги «списку»: а) редагування структури елементів; б) додавання елементу

«Таблиця» : табличне представлення списку з аналогічним функціоналом та можливістю сортування (рисунок 5.16). Для сортування достатньо натиснути на один із заголовків таблиці, якщо він придатний для сортування – воно відбудеться одразу (рисунок 5.17). Сортування не зберігається при перезавантаженні таблиці.

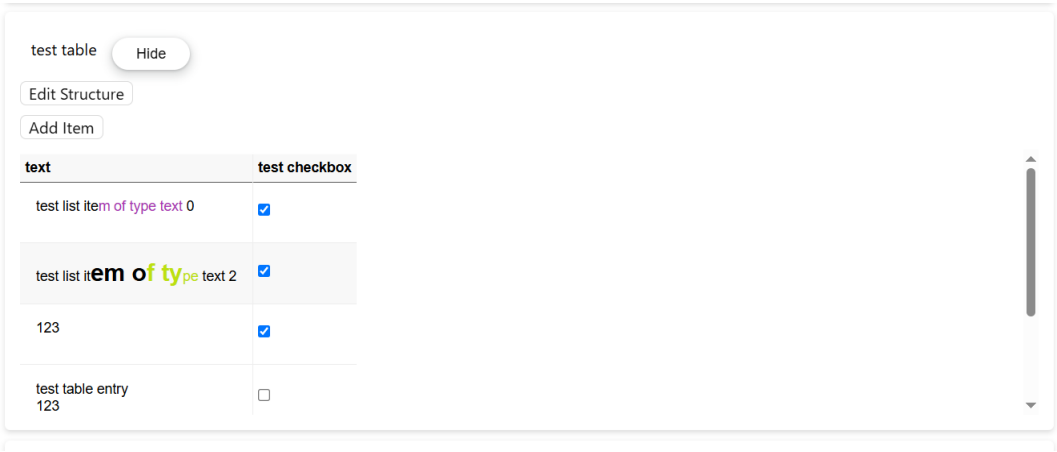
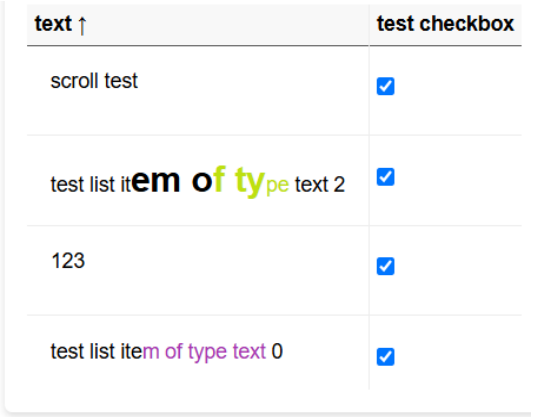
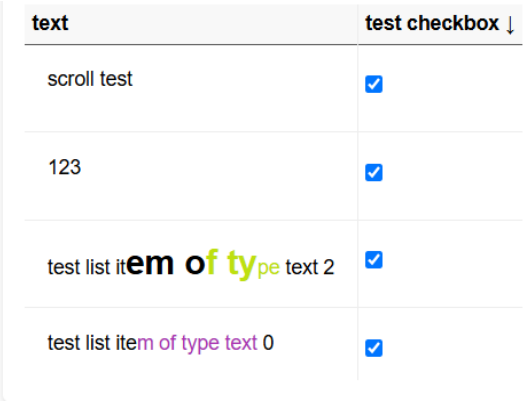



Рисунок 5.16 – «Таблиця»




а) б)

Рисунок 5.17 – Сортування таблиці: а) в прямому порядку;
б) в зворотному порядку

«Відвідування» : активність на основі списку та таблиці (рисунок 5.18). Даний тип активності відображає елементи списку як:

- для користувача який додав елемент: таблиця відміток користувачів яким надано до нього доступ.

- для решти користувачів, які мають доступ: звичайний елемент списку з активним «checkbox», після натискання на який, замість нього відображається час відмітки.

Елементи даного списку мають обов’язковий набір полів: дата, час початку та час кінця періоду часу, протягом якого дозволено відмічатись.


а) б)

Рисунок 5.18 – Елемент активності «Відвідування» для:
а) автора; б) решти користувачів

«Звіт» : активність аналогічна «відвідуванню», але замість «checkbox» використовується поле вводу тексту (рисунок 5.19).


Рисунок 5.19 – Елемент активності «Звіт» для «не автора»

«Чат»: базовий чат (рисунок 5.20). Повідомлення в чаті містять: власне повідомлення, повне ім’я користувача, який надіслав повідомлення та час і дату відправки.

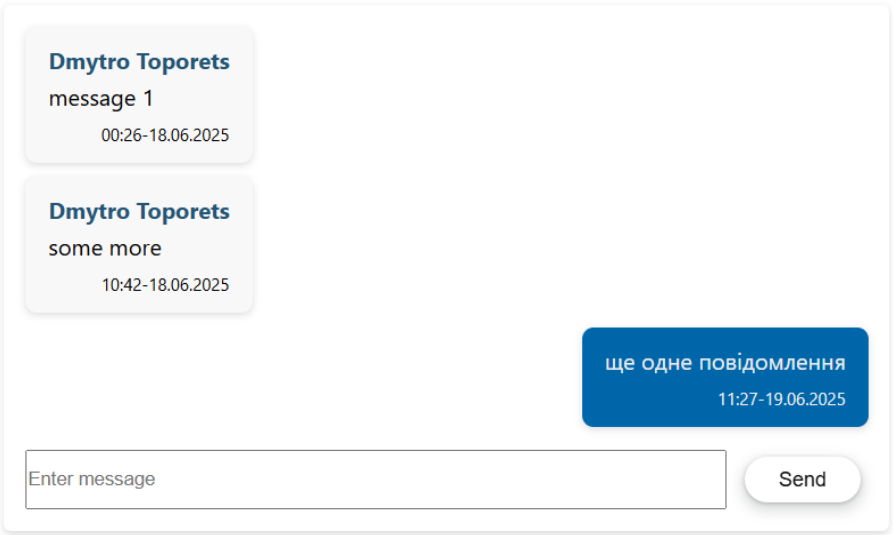



Рисунок 5.20 – «чат»

## 5.2 Інструкція для адміністратора

### 5.2.1 Вхід та реєстрація

При входу і додаток адміністратор потрапляє на стартову сторінку «Home Page». Далі в заголовку сторінки потрібно обрати «Log In», що відкриє відповідний діалог. Для входу потрібно ввести дані користувача та обрати сервер, до якого потрібно підключитись.

Для реєстрації в якості адміністратора, необхідно зареєструватись як звичайний користувач (у користувальницькому додатку) після чого вже існуючий адміністратор має надати права адміністратора (при створенні серверу, за замовчуванням існує один акаунт з правами адміністратора, його можна видалити після реєстрації хоча б одного додаткового адміністратора).

### 5.2.2 Інструменти адміністратора

Інструментарій адміністратора включає: перегляд логів серверу, редагування прав та видалення/бан користувачів і редактор шаблонів і типів активностей (в розробці).

Логи серверу зберігаються групами на сесію (від старту до відключення) серверу. Групи відображаються аналогічно до проектів у користувальницькому додатку, а самі логи – як таблиця.

Редагування прав користувачів, їх бан та видалення за оформленням повністю аналогічні редагуванню прав користувачів для елементу(проект чи активність). В даному випадку таблиця користувачів без доступу (незареєстрованих) відсутня.

# ВИСНОВКИ

У результаті проведеного аналізу сучасних засобів організації колективної діяльності в освітньому та IT-середовищі було встановлено, що надмірне використання цифрових рішень зі схожим функціоналом призводить до фрагментації інформаційного простору, дублювання даних, ускладнення взаємодії між користувачами та зниження загальної ефективності командної роботи. З метою усунення зазначених проблем розроблено бета-версію веб-платформи, яка поєднує функціональні можливості систем управління навчанням (LMS), відстеження задач та засобів колективного редагування.

Архітектура платформи базується на координаційно-ієрархічній мережевій моделі з виділеним сервером-координатором, що забезпечує гнучке масштабування, підтримку ізольованих підмереж та підвищення відмовостійкості. Комунікація між клієнтом і сервером реалізована на основі комбінованого підходу – через протоколи «WebSocket» (події в реальному часі) та «REST API» (запити до основних сервісів системи).

У якості технологічного стеку використано «MERN» (MongoDB, Express.js, React, Node.js), що забезпечує уніфікацію логіки клієнтської та серверної частин за рахунок використання єдиної мови програмування – JavaScript. Front-end реалізовано з використанням компонентного підходу (React). Серверна частина створена на базі «Node.js» з використанням Express.js, що дозволило організувати гнучкий «REST API» для обробки запитів і забезпечити масштабовану структуру взаємодії з клієнтом. Для передачі даних у реальному часі впроваджено «WebSocket»-підключення з механізмом heartbeat-контролю. Система аутентифікації та збереження даних користувачів реалізована із застосуванням сесій та токенів. У якості «СУБД» використано «MongoDB», для гнучкої роботи з вкладеними структурами документів та ефективного зберігання ієрархічних зв’язків між даними.

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. The hidden problems of too many collaboration tools / GoTo. – URL: https://www.goto.com/blog/the-hidden-problems-of-too-many-collaboration-tools (дата звернення: 14.05.2025).

2. Are your project management tools causing friction? / Atlassian. – URL: https://www.atlassian.com/blog/jira/are-your-project-management-tools-causing-friction (дата звернення: 14.05.2025).

3. 3 symptoms of a fragmented ecosystem for collaboration and communication / CMSWire. – URL: https://www.cmswire.com/digital-workplace/3-symptoms-of-a-fragmented-ecosystem-for-collaboration-and-communication/ (дата звернення: 14.05.2025).

4. Benefits of standardizing collaboration tools in higher ed / EdTech Magazine. – URL: https://edtechmagazine.com/higher/article/2022/06/benefits-standardizing-collaboration-tools-higher-ed (дата звернення: 15.05.2025).

5. Jira Software – Project Management Software / Atlassian. – URL: https://www.atlassian.com/software/jira (дата звернення: 12.05.2025).

6. Agile project management with Jira Software / Atlassian. – URL: https://www.atlassian.com/software/jira/agile (дата звернення: 14.05.2025).

7. What is Jira software? / GeeksforGeeks. – URL: https://www.geeksforgeeks.org/what-is-jira-software/ (дата звернення: 15.05.2025).

8. What is Trello? / Trello. – URL: https://trello.com/guide.

9. The pros and cons of using Trello software / Bridge24. – URL: https://bridge24.com/blog/the-pros-and-cons-of-using-trello-software-9387815/?afmc=pmnet (дата звернення: 14.05.2025).

10. Asana – Work Management Software for Teams / Asana. – URL: https://asana.com/ (дата звернення: 12.05.2025).

11. Asana pros and cons / FreshBooks. – URL: https://www.freshbooks.com/hub/projects-management/asana-pros-and-cons (дата звернення: 15.05.2025).

12. What is Notion? / Notion Labs. – URL: https://www.notion.so/help (дата звернення: 13.05.2025).

13. Notion review 2024: Features, pros & cons / KDnuggets. – URL: https://www.kdnuggets.com/review/notion-review-2024-features-pros-cons (дата звернення: 16.05.2025).

14. What is an LMS? / G2. – URL: https://www.g2.com/articles/what-is-an-lms (дата звернення: 13.05.2025).

15. DL.NURE – Дистанційне навчання НУ «ХАІ» / DL.NURE. – URL: https://dl.nure.ua/ (дата звернення: 13.05.2025).

16. Pros and cons of Moodle as an LMS platform / CourseOrbit. – URL: https://courseorbit.com/blog/pros-and-cons-of-moodle-as-an-lms-platform/ (дата звернення: 16.05.2025).

17. Live Share – Real-time Collaborative Development / Visual Studio Code. – URL: https://visualstudio.microsoft.com/services/live-share/ (дата звернення: 13.05.2025).

18. How to create an awesome Kanban board using dnd-kit [Відео] / Chetan Verma. – 8 серпня 2023. – Доступно за посиланням: https://www.youtube.com/watch?v=IZ3w2PNh-hE.

19. Курос Дж. Ф., Росс К. В. Комп’ютерні мережі : підхід зверху вниз / Дж. Ф. Курос, К. В. Росс. – 8-ме вид. – Бостон : Pearson, 2020. – 864 с.

20. Курос Дж. Топології в мережах : стаття. – Доступно за посиланням: https://www.sciencedirect.com/.

21. Форузан Б. А. Передача даних та мережі / Б. А. Форузан. – 5-те вид. – Нью-Йорк : McGraw-Hill, 2013. – 1 136 с.

22. Таненбаум А. С., Везералл Д. Дж. Комп’ютерні мережі / А. С. Таненбаум, Д. Дж. Везералл. – 5-те вид. – Аппер-Седдл-Рівер : Pearson, 2011. – 960 с.

23. Голмс А. Початок роботи з Node.js : вивчіть основи Node.js за вихідні / А. Голмс. – Сан-Франциско : Apress, 2018. – 230 с.

24. Браун В. Express у дії : написання, побудова та тестування застосунків на Node.js / В. Браун. – Нью-Йорк : Manning Publications, 2016. – 265 с.

25. How to use WebSockets with React and Node [Відео] / Ably Realtime. – 14 листопада 2023. – Доступно за посиланням: https://www.youtube.com/watch?v=4Uwq0xB30JE&t=2368s.

26. Чодоров К. MongoDB : докладний посібник / К. Чодоров. – 3-тє вид. – Себастополь : O'Reilly Media, 2019. – 482 с.

27. Грайдер С. Сучасний React з Redux : онлайн-курс / С. Грайдер. – Udemy,2023. – Доступно за посиланням: https://udemy.com/course/react-redux/.

28. Фленаган Д. JavaScript : докладний посібник / Д. Фленаган. – 7-ме вид. – Себастополь : O'Reilly Media, 2020. – 706 с.

29. dnd kit – a modern drag and drop toolkit for React [Електронний ресурс]. – Офіційний сайт dnd-kit, 2025. – Режим доступу: https://dndkit.com/ (дата звернення: 26.06.2025).

30. Meta Platforms Inc. React: Lifting State Up [Електронний ресурс]. – React.js Documentation. – Режим доступу: https://legacy.reactjs.org/docs/lifting-state-up.html (дата звернення: 18.05.2025).

31. Meta Platforms Inc. React: Composition vs Inheritance [Електронний ресурс]. – React.js Documentation. – Режим доступу: https://legacy.reactjs.org/docs/composition-vs-inheritance.html (дата звернення: 18.05.2025).

32. Refine.dev. React Design Patterns [Електронний ресурс]. – Refine Blog, 05.09.2024. – Режим доступу: https://refine.dev/blog/react-design-patterns/ (дата звернення: 30.05.2025).

33. React.dev. Updating Arrays in State [Електронний ресурс]. – Режим доступу: https://react.dev/learn/updating-arrays-in-state (дата звернення: 18.05.2025).

34. React Router: Navigation and Routing Essentials [Електронний ресурс]. – 4Geeks Academy. – Режим доступу: https://4geeks.com/lesson/react-router-navigation (дата звернення: 15.06.2025).

35. Denhup L. Understanding unidirectional data flow in React [Електронний ресурс]. – Medium.com, 25.09.2017. – Режим доступу: https://medium.com/@lizdenhup/understanding-unidirectional-data-flow-in-react-3e3524c09d8e (дата звернення: 30.05.2025).

36. Kondov A. Lessons Learned: Common React Code-Smells and How to Avoid Them [Електронний ресурс]. – HackerNoon, 27.05.2020. – Режим доступу: https://medium.com/hackernoon/lessons-learned-common-react-code-smells-and-how-to-avoid-them-f253eb9696a4 (дата звернення: 23.05.2025).

37. Sanjayamal R. Naming Conventions Best Practices in React [Електронний ресурс]. – Medium.com, 14.04.2023. – Режим доступу: https://medium.com/@rajithasanjayamal/naming-conventions-best-practices-in-react-37624d020288 (дата звернення: 23.05.2025).

38. Carlotta M. Is there an official style guide or naming convention for React-based projects? [Електронний ресурс]. – StackOverflow, 10.05.2019. – Режим доступу: https://stackoverflow.com/questions/55221433/is-there-an-official-style-guide-or-naming-convention-for-react-based-projects (дата звернення: 25.05.2025).