

Приватний заклад вищої освіти
Одеський технологічний університет «ШАГ»
Кафедра інформаційних технологій та фундаментальної підготовки

Випускна кваліфікаційна робота бакалавра

**«Інформаційна система гнучкого управління програмними проектами
(на прикладі Trello)»**

на здобуття бакалаврського рівня вищої освіти
зі спеціальності «122 Комп'ютерні науки»

Виконавці проекту

	П.І.Б.	Група
1	Миронов О. М.	КН-П-211
2	Костанда Д. О.	КН-П-211
3	Жданов А. Д.	КН-П-211
4	Бордюжевич О. І.	КН-П-211
5	Голубов В. А.	КН-Д-211

Автор звіту

_____ Жданов Артем Денисович

Науковий керівник	Доцент кафедри ІТ та ФП, к.т.н., доцент Данилець Є. В.
-------------------	--

АНОТАЦІЯ

УДК 004.415.2+004.415.3+004.43:005.8

Ж-42

Жданов А. Д.

Інформаційна система організації управління програмними проектами та завданнями (клієнтська частина): випускна кваліфікаційна робота на здобуття бакалаврського рівня вищої освіти зі спеціальності «122 Комп'ютерні науки». – Одеса: ОТУШ, 2025. – 22 с.

Робота присвячена розробці клієнтської частини інформаційної системи *ItRun*, призначеної для візуального управління проектами у форматі канбан-дошки. Основна мета — створення зручного, адаптивного та продуктивного SPA-застосунку з інтуїтивним інтерфейсом, підтримкою drag-and-drop, керуванням задачами, колонками, дошками та робочими просторами.

Актуальність дослідження обумовлена зростанням попиту на інструменти управління командною взаємодією, які є простими у використанні, гнучкими в налаштуванні та доступними для невеликих команд. Особливу увагу приділено типобезпечній роботі з даними, оптимізації взаємодії з сервером та валідації форм користувача.

Фронтенд частина реалізована з використанням:

- React — як бібліотеки для побудови компонентного UI;
- TypeScript — для забезпечення типобезпечності;
- React Hook Form — для створення багатокрокових форм із валідацією;
- Fetch API — для побудови централізованого API-шару взаємодії з бекендом;
- Drag-and-Drop — для динамічного переміщення карток і колонок у межах дошки;
- React Router — для клієнтської маршрутизації.

Було реалізовано повноцінний інтерфейс із підтримкою всіх основних функцій: створення/редагування задач, управління просторами, дошками, фільтрація, валідація та інтерактивна робота користувача з інтерфейсом. Архітектура застосунку побудована за модульним принципом з поділом на функціональні шари (UI-компоненти, API, форми, роутинг, логіка drag-and-drop), що забезпечує підтримуваність і масштабованість коду.

У результаті реалізації досягнуто:

- типобезпечну інтеграцію з серверною частиною;
- гнучку архітектуру з можливістю розширення;
- високий рівень продуктивності та UX з підтримкою адаптивного дизайну.

Клієнтська частина системи *ItRun* успішно протестована, забезпечує повний цикл роботи користувача та може бути масштабована під інші типи управлінських платформ. У роботі використано 12 джерел, зокрема наукові статті, технічну документацію та практичні приклади реалізації сучасних вебтехнологій.

ЗМІСТ

ВСТУП _____	4
Технічне завдання до проєкту _____	5
РОЗДІЛ 1. Загальна характеристика системи _____	8
РОЗДІЛ 2. Реалізація клієнтської частини _____	10
2.1 Технології та засоби створення інтерфейсу _____	10
2.2 Архітектура клієнтської частини та застосування технологій _____	14
РОЗДІЛ 3. Результати випробувань _____	18
ВИСНОВКИ _____	21
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ _____	22

ВСТУП

Стрімкий розвиток інформаційних технологій суттєво трансформував способи організації праці, взаємодії команд та управління цифровими проєктами. В умовах глобалізації, розподілених команд та зростання обсягу задач виникає потреба в інструментах, що дозволяють гнучко, наочно й ефективно керувати робочими процесами, розподіляти відповідальність та контролювати строки виконання. Саме тому системи управління завданнями та проєктами (так звані таск-трекери) стали невід’ємним компонентом сучасного цифрового середовища.

Незважаючи на наявність великої кількості аналогічних систем — таких як Trello, Jira, Asana — більшість із них мають обмежену гнучкість налаштувань, складний інтерфейс або потребують підключення сторонніх інтеграцій навіть для базових функцій. Крім того, значна частина таких сервісів є платними або обмеженими у функціональності для невеликих команд. Ці недоліки створюють передумови для розробки власного рішення, яке буде адаптовано до специфічних вимог користувачів, матиме відкриту архітектуру, забезпечуватиме контроль над даними та можливість подальшого розширення.

У зв’язку з цим команда розробників реалізувала інформаційну систему управління програмними проєктами та завданнями — ItRun. Основною метою проєкту є створення зручного, адаптивного та масштабованого вебзастосунку для управління завданнями у форматі канбан-дошки, з підтримкою користувачів, робочих просторів, колонок, карток і спільної роботи над проєктами. Платформа реалізована за сучасними технологічними підходами з використанням модульної архітектури, container-based deployment та засобів безперервної інтеграції.

У межах цієї роботи детально розглядається клієнтська частина інформаційної системи ItRun, яка реалізована з урахуванням вимог до сучасних SPA-застосунків. Особливу увагу приділено архітектурі компонентів, ефективній побудові UI та UX, формуванню інтерактивного досвіду взаємодії з інтерфейсом через drag-and-drop механізми, а також впровадженню типобезпечної роботи з формами за допомогою React Hook Form. Також у роботі розглянуто питання інтеграції з серверною частиною через централізований API-шар та організації запитів із використанням Fetch API. Обґрунтовано вибір стеку технологій — React, TypeScript, та інших — як оптимального рішення для розробки масштабованої, продуктивної та зручно підтримуваної клієнтської частини інформаційної системи управління програмними проєктами.

ТЕХНІЧНЕ ЗАВДАННЯ ДО ПРОЄКТУ

Загальний опис

Програмне забезпечення «ItRun» призначене для організації ефективного управління завданнями та цифровими проєктами в команді. Система створюється у форматі вебзастосунку на базі клієнт-серверної архітектури з можливістю масштабування, контейнеризації та автоматизованого розгортання. Основною ідеєю є створення інструмента для візуального управління проєктами за допомогою канбан-дошок, що дає змогу формувати робочі простори, створювати задачі, призначати відповідальних, відслідковувати статуси виконання, коментувати й координувати дії команди.

Застосунок забезпечує повноцінну взаємодію між учасниками одного або декількох робочих просторів, підтримує контроль версій, зберігає історію дій користувачів, а також включає механізми захисту персональних даних та автентифікації через підтвердження email. Платформа передбачає розподіл ролей та прав доступу, що забезпечує гнучке адміністрування проєктів.

Призначення

Система орієнтована на використання невеликими та середніми командами розробників, стартапами, фриланс-агенціями, а також окремими користувачами, які потребують інструменту для організації власної діяльності. Особлива увага приділена простоті використання, зручному UI/UX, можливості гнучкої інтеграції, та легкому оновленню функціоналу в майбутньому без втрати даних. Застосунок має підтримувати повний цикл взаємодії: створення задач, їх планування, виконання, коментування та завершення.

Функціональні вимоги

1. Реєстрація та автентифікація

Кожен користувач повинен мати можливість створити обліковий запис у системі. Після реєстрації на електронну адресу надсилається підтвердження, яке користувач має ввести для завершення реєстрації. Після цього користувач може увійти до системи, вказавши свої облікові дані. Усі дії, які потребують доступу до персональної інформації або даних проєктів, мають бути доступні лише авторизованим користувачам. Також має бути можливість вийти із системи, в тому числі з усіх пристроїв одночасно. Деякі функції доступні тільки певним категоріям користувачів (наприклад, адміністраторам).

2. Робочі простори та дошки

Користувачі можуть створювати окремі робочі простори — умовні «команди» або «проєкти», всередині яких розміщуються канбан-дошки. У таких дошках відображається весь хід роботи над завданнями. Користувачі мають можливість змінювати зовнішній вигляд дошки (наприклад, завантажувати зображення чи логотипи), а також запрошувати інших людей до участі у роботі над проєктом. При

цьому можна визначати роль кожного учасника — наприклад, дозволити редагування лише окремим особам, а іншим лише переглядати вміст.

3. Колонки та задачі

Всередині кожної дошки можна створювати колонки (етапи роботи), в які додаються окремі завдання. Ці завдання можна редагувати, переміщати між колонками, видаляти або змінювати їх порядок. Кожне завдання може мати дату завершення, пріоритетність, позначки або додані файли. Також має бути візуальне відображення того, на якому етапі виконання перебуває завдання, щоб усі учасники команди могли швидко оцінити прогрес роботи.

Технічні вимоги та рекомендації до реалізації

Клієнтська-частина реалізується у вигляді односторінкового застосунку (SPA) із застосуванням сучасного стеку технологій:

- Реалізація інтерфейсних компонентів з використанням бібліотеки React у поєднанні з мовою TypeScript
- Клієнтська навігація — забезпечується за допомогою бібліотеки React Router
- Інтерактивна функціональність drag-and-drop — реалізована з використанням відповідних інструментів, таких як react-beautiful-dnd
- Інтеграція з серверною частиною через REST API — включає обробку помилок, відображення індикаторів завантаження (loader'ів), валідацію форм введення
- UI/UX-дизайн — відповідає принципам адаптивності, інтуїтивності та доступності для користувача

Серверна-частина спроектована на основі фреймворку NestJS та побудована відповідно до концепції доменно-орієнтованого дизайну (DDD):

- Модульна структура застосунку — з поділом на auth, user, workspace, board, task, comment, history
- Робота з базою даних MySQL — реалізована через ORM-бібліотеку TypeORM
- Побудова REST API — із валідацією запитів за допомогою Data Transfer Object (DTO) та централізованою обробкою винятків
- Контроль доступу до ресурсів — реалізовано згідно з рольовою моделлю користувача (Admin / Member)
- Дотримання архітектурних патернів — що забезпечує масштабованість і підтримуваність системи

DevOps та інфраструктурний рівень охоплює засоби автоматизації розгортання та підтримки життєвого циклу застосунку:

- Контейнеризація проєкту — з використанням Docker для створення окремих контейнерів: frontend, backend та бази даних MySQL
- Опис взаємодії сервісів — через docker-compose конфігурацію
- Розгортання проєкту на VPS — із встановленим CentOS та налаштуванням

зворотного проксингу через веб-сервер Nginx

- Збереження конфігурацій — у вигляді .env-файлів із секретними змінними оточення
- Реалізація CI/CD — із застосуванням GitHub Actions для автоматичного тестування, білду та розгортання системи

Менеджмент та супровід розробки включає організацію роботи команди, планування та документування проєкту:

- Постановка задач, формування user stories та створення дорожньої карти (roadmap) — реалізовано в середовищі Notion
- Використання Kanban-дошки — для відстеження статусів задач у Trello, що забезпечує прозору візуалізацію прогресу
- Проведення регулярних синків команди — з фіксацією результатів у звітах
- Централізована технічна документація — включає опис функціональності, архітектури, бізнес-логіки, API та сценаріїв використання
- Забезпечення відповідності реалізації вимогам технічного завдання — з обов'язковим контролем якості на всіх етапах розробки.

UI/UX-дизайн інтерфейсного рішення платформи ItRun базувався на сучасних підходах до цифрового продуктового дизайну з урахуванням потреб цільової аудиторії та принципів зручності використання:

- UX-дослідження — аналіз Trello, емпатійна карта, Customer Journey Map
- Структура взаємодії — побудова Site Map та User Flow
- Дизайн-система — типографіка (Poppins), кольори (Tiffany, помаранчевий), сітка 16 пікселів
- UI-компоненти — реалізація у станах (default, hover, active, disabled), auto-layout
- Прототипування — високофідельні екрани у Figma: реєстрація, онбординг, workspace, дошка
- UX-тестування — виявлення та виправлення проблем за результатами користувацьких сценаріїв
- Айдентика — створення логотипу, що поєднує IT-естетику та динаміку керування завданнями

Усі частини є незалежними, розробляються окремими учасниками команди, мають інтерфейси взаємодії та підлягають документуванню.

РОЗДІЛ 1. ЗАГАЛЬНА ХАРАКТЕРИСТИКА СИСТЕМИ

Відповідно до технічного завдання (ТЗ), командою розробників було спроектовано та реалізовано інформаційну систему управління програмними проектами за архітектурою клієнт-сервер. Система побудована за принципом SPA (Single Page Application), що забезпечує швидкий відгук користувача, мінімізацію затримок завантаження та зручність інтеграції з API. У процесі розробки було структуровано наступні функціональні підсистеми:

- «Design»
- «Frontend»
- «Backend»
- «DevOps»
- «Project Management»

Частина **Design** містить комплекс макетів та прототипів інтерфейсів користувача, які були розроблені із дотриманням сучасних принципів UI/UX-дизайну. У ході проектування інтерфейсів було визначено кольорову палітру, типографіку, компонентну бібліотеку (UI-kit), а також реалізовано інтерактивні анімації для забезпечення комфортної взаємодії користувача з додатком. Усі графічні макети було створено в середовищі Figma відповідно до узгодженого функціоналу кожного модуля.

Підсистема **Frontend** реалізована за допомогою бібліотеки React із використанням мов програмування TypeScript та JavaScript, а також HTML/CSS для розмітки та стилізації інтерфейсу. Комунікація з сервером здійснюється через HTTP-запити за допомогою вбудованого API клієнта. Особливу увагу було приділено адаптивності та продуктивності інтерфейсу, зокрема реалізації компонентного підходу та валідації форм введення.

Серверна логіка, що складає частину **Backend**, була реалізована на платформі Node.js із використанням фреймворку NestJS. Система підтримує REST API, що забезпечує модульну структуру запитів та відповідей у форматі JSON. Для зберігання даних застосовано реляційну СУБД MySQL, взаємодія з якою відбувається через ORM-бібліотеку TypeORM. У якості механізму автентифікації впроваджено JWT-токени, що дозволяє реалізувати контроль доступу до обмежених ресурсів системи.

Адміністративна частина **DevOps** охоплює процеси автоматизації розгортання та адміністрування застосунку. Для забезпечення ізоляції компонентів було використано Docker-контейнеризацію. Розгортання інфраструктури здійснюється на базі VPS-сервера з використанням Docker Compose, що дозволяє конфігурувати та запускати пов'язані сервіси одночасно. Автоматизацію CI/CD-процесів реалізовано через GitHub Actions, що забезпечує безперервну інтеграцію та доставку оновлень до серверного середовища без необхідності ручного втручання.

У межах підсистеми **Project Management** особливу увагу було приділено управлінню знаннями проекту та організації робочого процесу за методологією Kanban. Документація всіх етапів — від постановки задач до технічних рішень — здійснювалась у Notion, що забезпечило централізований доступ до специфікацій API,

архітектури, записів зустрічей і ретроспектив. Такий підхід дозволив системно керувати беклогом, user stories та приймальними критеріями, забезпечуючи прозорість і гнучкість у процесі розробки.

Таким чином, система реалізована відповідно до сучасних стандартів розробки веб-застосунків і забезпечує високу гнучкість, масштабованість та підтримку безперервної інтеграції, що відповідає запитам сучасних програмних продуктів у сфері управління проектами.

РОЗДІЛ 2 РЕАЛІЗАЦІЯ КЛІЄНТСЬКОЇ ЧАСТИНИ

2.1 Технології та засоби створення інтерфейсу

Клієнтська частина вебзастосунку реалізована з використанням бібліотеки React — однієї з найпопулярніших технологій для створення інтерфейсів користувача. React побудований на компонентному підході, що дозволяє розбивати інтерфейс на окремі ізольовані блоки, які можна багаторазово використовувати. Це сприяє модульності коду, полегшує його супровід та спрощує масштабування проєкту. Завдяки віртуальному DOM React забезпечує високу продуктивність навіть при значних оновленнях стану інтерфейсу. Визначення бібліотеки React, як однієї із технологій для реалізації клієнтської частини можна знайти у науковій літературі і визначається вона як:

React — це сучасна JavaScript-бібліотека, призначена для побудови інтерфейсів користувача в односторінкових веб-застосунках. Вона орієнтована на вирішення задач динамічного оновлення даних без потреби повного перезавантаження сторінки. React розробляється та підтримується компаніями Facebook і Instagram, а також спільнотою незалежних розробників. Свою популярність бібліотека здобула завдяки простоті у використанні, високій продуктивності та активному розвитку екосистеми. [2]

У поєднанні з TypeScript — React стає ще більш надійним інструментом розробки. **TypeScript** — це мова програмування, розроблена корпорацією Microsoft, яка є надбудовою над JavaScript та розширює його функціональність завдяки підтримці статичної типізації, класів, інтерфейсів та модульної архітектури. TypeScript компілюється у чистий JavaScript, що дозволяє використовувати його у будь-якому середовищі, яке підтримує стандартний JavaScript. Його основна мета — забезпечити масштабовану розробку та покращити інструменти розробника на будь-якому етапі проєкту. [3]

На відміну від JavaScript, TypeScript дозволяє явно вказувати типи змінних та аргументів, що дає змогу виявляти помилки вже на етапі компіляції. Завдяки підтримці об'єктно-орієнтованих концепцій, таких як класи, інтерфейси, наслідування, а також можливості створення декларативних описів для сторонніх бібліотек, TypeScript забезпечує підвищену надійність, читабельність і повторне використання коду. Статична типізація дозволяє IDE автоматично підказувати властивості, перевіряти типи та суттєво прискорювати процес розробки. [1] [4]

Використання TypeScript дозволяє виявляти помилки ще на етапі компіляції, зменшуючи кількість критичних помилок у рантаймі. На відміну від фреймворку Angular, React має менше жорстких обмежень у структурі проєкту, що робить його гнучкішим і зручнішим для розробки сучасних SPA (Single Page Applications), особливо в невеликих та середніх командах. Саме тому комбінація React + TypeScript була обрана як оптимальна для реалізації клієнтської частини системи. [4]

На початковому етапі розробки клієнтської частини було проведено аналіз інструментів, що дозволяють ефективно працювати з формами у середовищі React.

Зокрема, було розглянуто дві популярні бібліотеки: **Formik** та **React Hook Form**. Обидві бібліотеки забезпечують зручну обробку форм, валідацію, взаємодію з React-компонентами та мають активне співтовариство, проте суттєво відрізняються у підходах до керування станом і продуктивності [5][6].

Для прийняття обґрунтованого рішення було складено порівняльну таблицю 2.1 основних характеристик:

Таблиця 2.1 – Порівняльна характеристика Formik та React Hook Form

Особливість	Formik	React Hook Form
Обновлення і рендери	Часті рендери при вводі (управлювані компоненти)	Мінімальні рендери (через refs і неконтрольовані компоненти)
Простота використання	Простий синтаксис, багата документація	Вимагає ознайомлення з hook-підходом
Розмір бібліотеки	Середній (≈12–15 КБ)	Малий (≈3–4 КБ)
Продуктивність	Може бути нижчою при великих формах	Висока, підходить для масштабних рішень
Валідація	Добра інтеграція з Yup	Також підтримка Yup та власних рішень
API	Декларативний, імперативний	Функціональний, hook-орієнтований
Документація та спільнота	Велика база прикладів, стабільна підтримка	Динамічно зростаюча спільнота, активна підтримка
Підтримка TypeScript	Є, але іноді виникають труднощі	Відмінна підтримка

Висновки аналізу дозволяють сформулювати рекомендації щодо вибору:

- **Formik** доцільно використовувати у випадках, коли:
 - форма має невелику або середню складність;
 - важлива простота реалізації;
 - питання продуктивності не є критичним .[6]
- **React Hook Form** є кращим вибором для:
 - великих форм із великою кількістю полів;
 - проектів, де критична продуктивність;
 - використання сучасного підходу до управління станом на базі React-хуків [5].

У підсумку, для реалізації клієнтської частини даного проєкту було обрано **React Hook Form** — як більш продуктивне, гнучке та типобезпечне рішення, яке відповідає сучасним практикам фронтенд-розробки.

У реалізації клієнтської частини вебзастосунку для здійснення HTTP-запитів до REST API обрано **Fetch API** — нативний інтерфейс браузера, який не потребує сторонніх бібліотек і дозволяє реалізовувати повний цикл мережових операцій. Метод **fetch()** підтримує передачу заголовків, обробку статусів відповіді, перехоплення помилок, а також читання тіла запиту у форматах JSON, тексту, Blob або FormData. Такий підхід не лише зменшує розмір фінального JavaScript-бандлу, а й є більш універсальним для простих запитів. [7], [8]

Fetch API використовує проміси, що дозволяє зручно інтегрувати його з асинхронною логікою React. Він забезпечує обробку запиту у два етапи — спочатку отримання заголовків відповіді, а потім — зчитування її тіла. [8]

У прикладі нижче описано синтаксис запиту до відкритого API (url) із подальшою обробкою JSON-відповіді:

```
const response = await fetch(url, [options]);
if (response.ok) {
  const users = await response.json();
  console.log(users);
} else {
  console.error('HTTP Error:', response.status);
}
```

*Запит із використанням **Fetch API***

Для організації маршрутизації у межах односторінкового застосунку (SPA) застосовано **React Router** — бібліотеку, що забезпечує клієнтську навігацію без перезавантаження сторінки. Вона дозволяє налаштовувати динамічні маршрути, вкладені компоненти, переадресації та захищені сторінки (через Guard-патерни). Завдяки тісній інтеграції з React, маршрутизація реалізована декларативно, через компоненти **<Route>**, **<Link>** та **<Navigate>**, що забезпечує гнучкість та масштабованість навігаційної логіки. [9]

Для реалізації основного функціоналу інформаційної системи організації управління програмними проєктами та завданнями, зокрема можливості зручного управління задачами (переміщення між колонками, зміна пріоритету тощо), було обрано технологію **Drag-and-Drop**.

Drag-and-Drop — це інтерактивна технологія графічного інтерфейсу, що дозволяє користувачам маніпулювати елементами інтерфейсу шляхом їх "перетягування" з однієї області в іншу за допомогою комп'ютерної миші. Такий підхід значно спрощує взаємодію з великим обсягом даних та підвищує інтуїтивність роботи з інтерфейсом. Він широко використовується в сучасних додатках, зокрема в навчальних,

управлінських та комерційних системах, як один із ключових способів візуального керування контентом [10].

У межах проєкту для реалізації механізму Drag-and-Drop було застосовано бібліотеку **Pragmatik** — інструмент, що полегшує обробку параметрів функцій з гнучким (варіативним) підходом до аргументів. Це дозволяє точно призначати дані відповідним змінним за заздалегідь визначеними правилами, що особливо корисно при побудові динамічної логіки переміщення задач у межах інтерактивного інтерфейсу [11].

У результаті використання сучасних технологій — React, TypeScript, Tailwind CSS, React Hook Form, Fetch API та Drag-and-Drop — вдалося реалізувати функціональну, швидку та масштабовану клієнтську частину, яка відповідає сучасним вимогам до зручності, продуктивності та підтримуваності вебзастосунків.

2.2 Архітектура клієнтської частини та застосування технологій

Застосунок реалізовано за модульною архітектурною моделлю, яка дозволяє досягти високої гнучкості, зрозумілості коду та розділення відповідальності між функціональними складовими. Такий підхід забезпечує структуровану організацію коду, що полегшує масштабування проєкту, його тестування та супровід. Кожен модуль виконує окрему роль у загальній логіці роботи клієнтської частини інформаційної системи. [12]

Основні функціональні модулі клієнтської частини:

- **Інтерфейсні компоненти** — реалізовані з використанням React та TypeScript, що забезпечує декларативність, типобезпечність і повторне використання UI-елементів.
- **API-шар** — відповідає за обробку HTTP-запитів до REST API за допомогою вбудованого методу `fetch`, забезпечуючи передачу та отримання даних від серверної частини.
- **Керування формами** — здійснюється через бібліотеку `react-hook-form`, яка дозволяє створювати продуктивні та контрольовані форми з мінімальною кількістю ререндерів.
- **Функціональність drag-and-drop** — реалізована з використанням Pragmatic DnD, що забезпечує інтерактивну роботу з завданнями у вигляді переміщення карток між колонками.
- **Клієнтська навігація** — забезпечується через React Router, що дає змогу реалізувати маршрутизацію між сторінками в межах SPA без перезавантаження.

У клієнтській частині застосунку `react-hook-form` використовується для обробки подій введення, валідації та управління станом багатокрокових форм. Завдяки типізації через `TypeScript` та динамічному `resolver` залежно від кроку, забезпечується гнучке налаштування логіки перевірки полів. Такий підхід підвищує продуктивність інтерфейсу та зменшує кількість зайвих ререндерів, що особливо важливо при великій кількості полів.

Нижче наведено приклад ініціалізації форми з підтримкою типів:

```
const {
  register,
  handleSubmit,
  formState: { errors },
  getValues,
  setValue,
} = useForm<FormData>({
  resolver: getResolver(step) as any,
  mode: "onChange",
});
```

Приклад ініціалізації форми з використанням react-hook-form

У межах проєкту **react-hook-form** забезпечує динамічну валідацію даних на кожному з п'яти етапів реєстрації, адаптуючи правила залежно від кроку. Реалізована інтеграція з Yup для перевірки коректності email та пароля, а також підключена перевірка email через API на унікальність. Здійснюється валідація файлів за розміром (до 5MB) та збереження введених даних між кроками без втрати стану. Завдяки оптимізованій роботі форми зменшується кількість перерендерів, а обробка помилок відбувається автоматично.

У реалізації інтерфейсу типу канбан у межах інформаційної системи управління проєктами ключову роль відіграє підтримка динамічного переміщення елементів користувачем. Для забезпечення такої взаємодії застосовано функціонал drag-and-drop, що реалізований за допомогою бібліотеки Pragmatik. Такий підхід дозволяє створити гнучке середовище з інтуїтивним управлінням завданнями: переміщення карток між колонками, зміна порядку самих колонок, а також інтерактивна візуалізація дій користувача в режимі реального часу.

Нижче наведено приклад реалізації drag & drop для карток та колонок:

```
// Налаштування drag & drop для карток
const cleanCardDrop = dropTargetForElements({
  element: ref.current,
  getData: () => ({ targetColumnId: column.id }),
  canDrop: ({ source }) => source.data.cardId !== undefined,
  onDrop: ({ source }) => {
    const { cardId, sourceColumn } = source.data;
    moveCard(cardId, sourceColumn, column.id);
  },
});

// Налаштування drag & drop для колонок
const cleanColumnDrop = dropTargetForElements({
  element: ref.current,
  getData: () => ({ columnId: column.id, columnIndex }),
  canDrop: ({ source }) => source.data.columnId !== undefined,
  onDrop: ({ source }) => {
    moveColumn(sourceColumnId, columnIndex);
  },
});
```

Приклад реалізації налаштування drag & drop для карток та колонок

Імплементація забезпечує зручне перетягування карток між колонками, зміну порядку розташування колонок та надає візуальні підказки під час виконання дій. Усі зміни автоматично синхронізуються через API, що гарантує збереження поточного стану інтерфейсу та забезпечує злагоджену роботу між клієнтською та серверною частинами системи.

Щоб забезпечити ефективну взаємодію клієнтської частини із сервером, необхідно впровадити чітко структурований підхід до реалізації HTTP-запитів. У цьому

проекті для цього передбачено централізований API-шар, який забезпечує типізовану взаємодію з сервером за допомогою `fetch()`-запитів. Така архітектура дозволяє чітко організувати логіку комунікацій, спрощує підтримку та масштабування коду. Усі запити згруповані за функціональними категоріями: управління колонками, операції з користувачами та робота з картками.

Основні категорії API-запитів включають наступне:

1. **Управління колонками** — що включає створення, переміщення, оновлення структурних елементів дошки. Нижче наведено приклад реалізації коду створення колонки:

```
export async function createColumn(workspaceId: string, boardId: string, data: {name: string}) {
    const response = await
    fetch(`${API_URL}/workspace/${workspaceId}/board/${boardId}/column`, {
        method: 'POST',
        headers: {
            'Authorization': `Bearer ${token}`,
            'Content-Type': 'application/json',
        },
        body: JSON.stringify(data),
    });
    return response.json();
}
```

Створення колонки

2. **Операції з користувачами** — що включає функціональності, пов'язані із реєстрацією, перевіркою email, встановленням мети та завантаженням аватара. Нижче наведено приклад реалізації вказаних функціональностей:

```
const newUser = await registerUser(userData);
const emailCheck = await checkEmail(data.email);
await uploadAvatar(photoFile, newUser.id);
await setUserPurpose({ userId: newUser.id, purpose: purpose
});
```

Операції з користувачами

3. **Операції з картками** — створення та отримання списку карток у межах заданих колонок:

```
const response = await createCard(workspaceId, boardId, columnId, { title: cardData.name });
const response = await getColumns(workspaceId, boardId);
```

Операції з картками

Такий підхід дозволяє підтримувати розділення логіки, зберігати гнучкість і масштабованість під час роботи з API, а також значно спрощує тестування, налагодження та розширення функціоналу.

Для забезпечення надійної обробки даних і підтримки масштабованості клієнтської частини інформаційної системи, у проєкті активно використовується статична типізація на основі **TypeScript**. Це дозволяє заздалегідь визначити структуру об'єктів, які очікуються з API, та зменшити кількість потенційних помилок у процесі виконання. Типи визначаються через інтерфейси та типи, що описують обов'язкові та необов'язкові поля, а також взаємозв'язки між сутностями.

<pre>type Card = { id: string; content: string; title?: string; description?: string; position?: number; };</pre>	<pre>type Column = { id: string; title: string; cards: Card[]; position?: number; createdAt?: string; updatedAt?: string; };</pre>
---	--

Оголошення типів Card і Column у форматі TypeScript для моделювання сутностей системи

Таким чином, архітектура клієнтської частини застосунку реалізована відповідно до принципів модульності, повторного використання коду та типобезпечності. Завдяки чіткому поділу на компоненти, централізованому API-шару, використанню сучасних технологій управління формами та інтерактивним функціональним блокам (наприклад, drag-and-drop), забезпечено не лише зручність для користувача, а й ефективність у розробці, тестуванні та масштабуванні системи.

РОЗДІЛ 3 РЕЗУЛЬТАТИ ВИПРОБУВАНЬ

Результатом компіляції клієнтської частини інформаційної системи управління програмними проєктами є повнофункціональний вебзастосунок, що працює у браузері користувача. Випробування проведено у браузерах Google Chrome та Mozilla Firefox на операційній системі Windows 11. Система успішно виконує всі передбачені сценарії, починаючи з реєстрації нового користувача, завершуючи створенням робочих просторів, дошок та управлінням завданнями в режимі drag-and-drop.

На початковому етапі взаємодії користувач має можливість авторизуватись через електронну пошту та пароль або за допомогою сторонніх сервісів (наприклад, Google), про що свідчить демонстрація на Рисунку 3.1. У разі створення нового облікового запису реєстрація реалізується через багатокрокову форму. Перший крок передбачає введення електронної пошти (Рисунок 3.2), другий — імені та пароля (Рисунок 3.3), третій — вибір аватару (Рисунок 3.4). На кожному етапі валідація виконується в режимі реального часу, що забезпечує коректність даних до моменту їх надсилання.

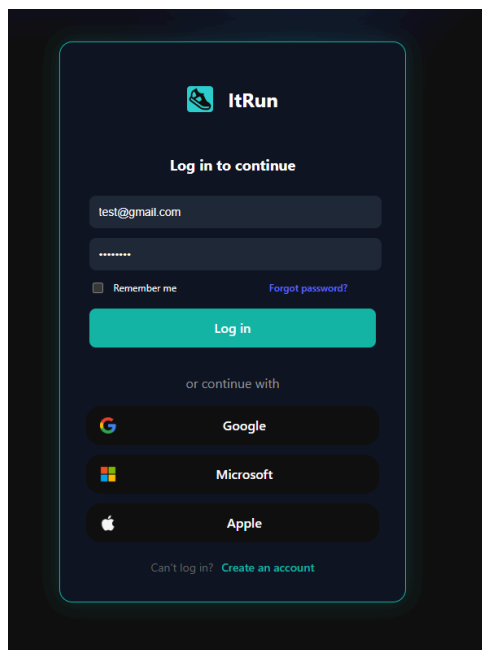


Рисунок 3.1

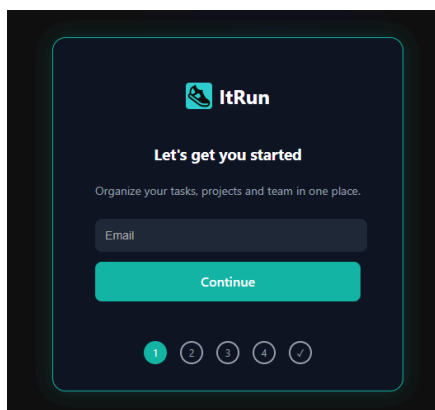


Рисунок 3.2

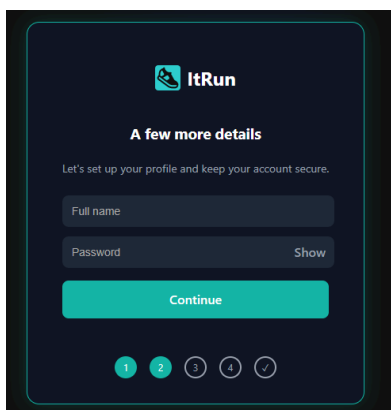


Рисунок 3.3

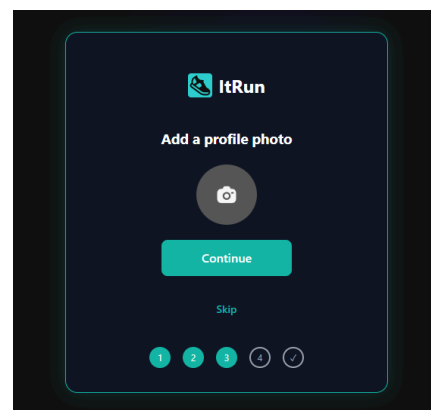


Рисунок 3.4

Після успішного проходження процедури авторизації/реєстрації користувач потрапляє на головну сторінку вебзастосунку, що виконує функцію центрального навігаційного вузла. Інтерфейс головної сторінки (Рисунок 3.5) орієнтований на швидкий доступ до ключових функцій системи, зокрема до створення нового робочого простору, що виступає базовою організаційною одиницею для подальшого управління бордами та завданнями. Ліва панель навігації надає можливість здійснювати перехід до існуючих просторів або ініціювати створення нових, а центральна частина сторінки демонструє шаблони проєктів, що можуть бути використані як основа.

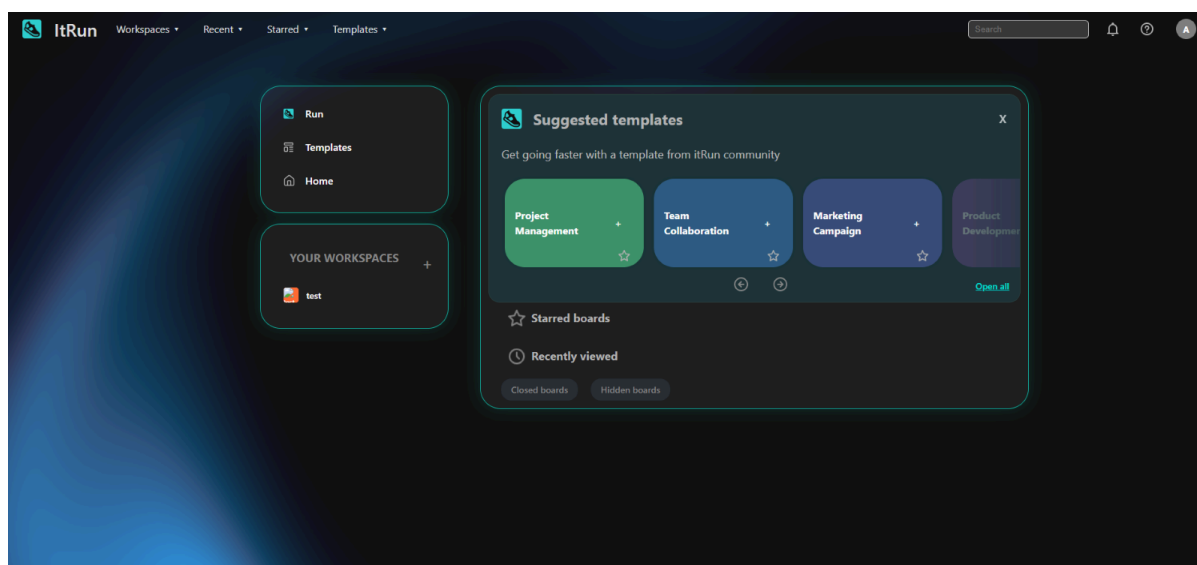


Рисунок 3.5

Після створення робочого простору користувач отримує доступ до функціоналу додавання нових бордів (дошок), а також можливості переходу до розширених налаштувань простору. Ці налаштування включають зміну його назви, керування учасниками та доступ до вкладених структур. На Рисунку 3.6 зображено процес створення дошки та елементи інтерфейсу, які забезпечують конфігурацію простору і доступ до відповідних функціональних розділів.

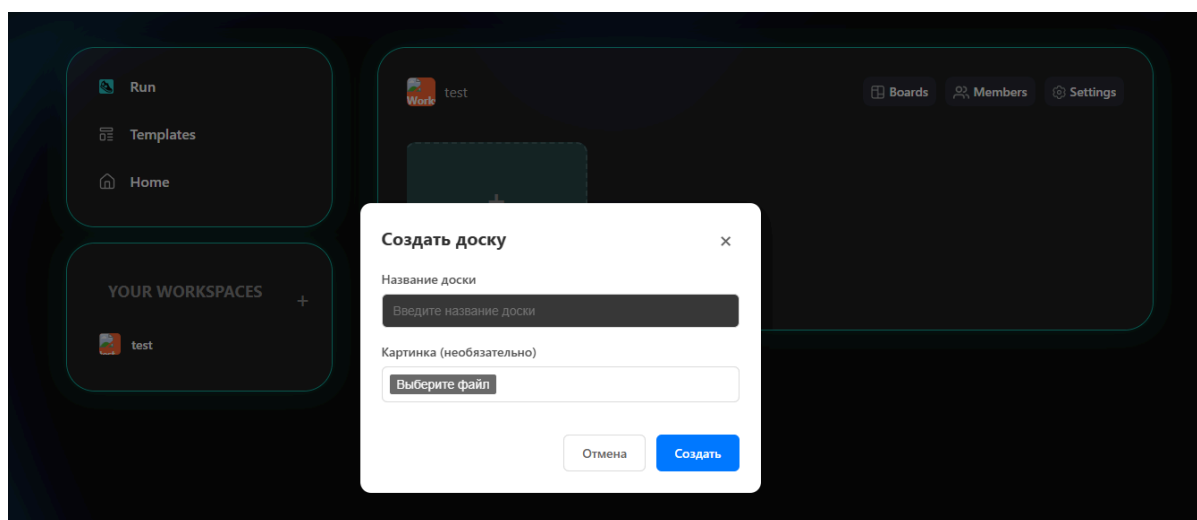


Рисунок 3.6

Інтерфейс самого робочого простору (Рисунок 3.7) реалізований таким чином, щоб забезпечити логічну ієрархію переходів між бордами та іншими вкладками. Навігація здійснюється через інтуїтивну бічну панель, де кожен борд представлено окремим елементом, перехід до якого активує відповідний візуальний простір для управління завданнями.

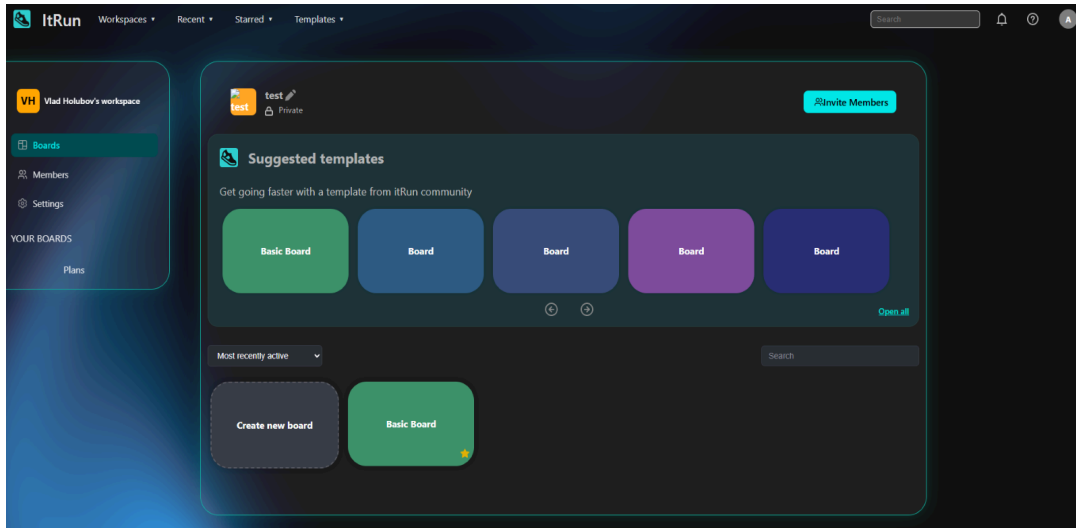


Рисунок 3.7

Функціональність борду (Рисунок 3.8) включає створення та редагування колонок, додавання карток, а також реалізацію механізму drag-and-drop для зміни їхнього порядку. Такий підхід дозволяє користувачеві швидко адаптувати структуру борду до поточних потреб проєкту, забезпечуючи візуальну гнучкість та ефективність у роботі з завданнями. Усі дії, пов'язані з переміщенням елементів, супроводжуються відповідною візуальною індикацією, а зміни синхронізуються з сервером через API.

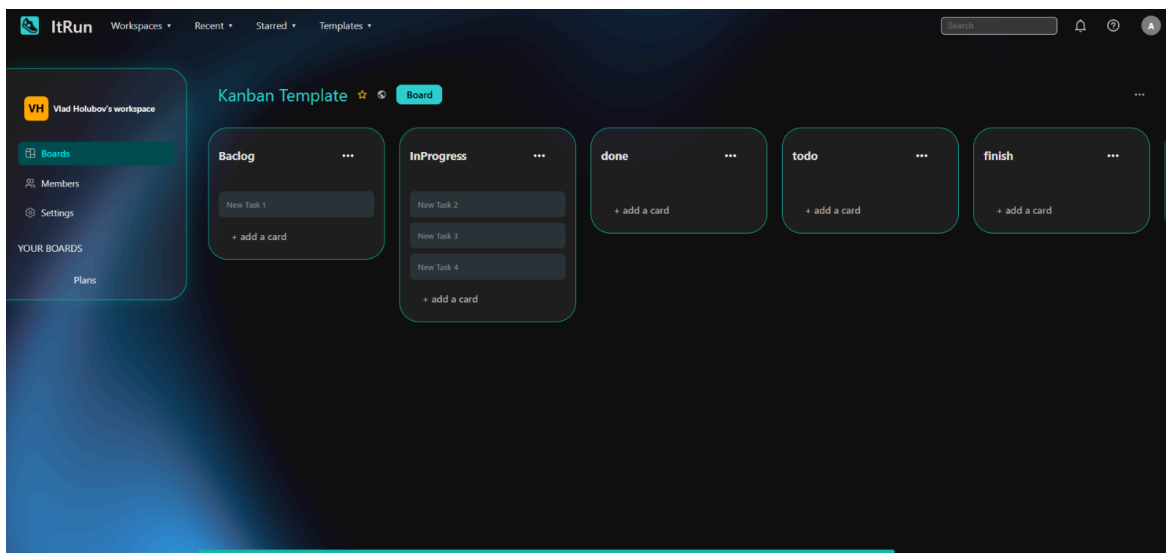


Рисунок 3.8

У результаті реалізації клієнтської частини інформаційної системи забезпечено повноцінний функціонал взаємодії з основними сутностями — робочими просторами, бордами, колонками та картками — що відповідає встановленим технічним вимогам та підтримує гнучкість у керуванні проєктами.

ВИСНОВКИ

У рамках проведеного дослідження та для виконання дипломного проєкту з реалізації клієнтської частини інформаційної системи організації управління програмними проєктами та завданнями було розглянуто сучасні технології та підходи до фронтенд-розробки, проаналізовано архітектурні рішення та реалізовано функціональний інтерфейс користувача відповідно до встановленого технічного завдання.

У результаті аналізу предметної області та вимог до системи було обґрунтовано вибір архітектурного стилю SPA (Single Page Application) та технологічного стеку, до якого увійшли React, TypeScript, React Hook Form, Drag-and-Drop та Fetch API. Ці інструменти забезпечили ефективну реалізацію інтерфейсу, високий рівень продуктивності, гнучкість структури застосунку та масштабованість у майбутньому розвитку.

У процесі розробки було реалізовано типобезпечну взаємодію з серверною частиною через централізований API-шар, що організовує усі HTTP-запити за функціональними модулями. Такий підхід значно спростив супровід, тестування та розширення функціоналу системи.

Застосування React Hook Form дозволило ефективно реалізовувати динамічні форми введення, з урахуванням валідації, збереження стану та інтеграції з типізацією.

В рамках дослідження було визначено, що реалізована в рамках проєкту функціональність drag-and-drop забезпечує інтуїтивну взаємодію з ключовими сутностями системи — картками, колонками та дошками.

Архітектура клієнтської частини побудована відповідно до принципів компонентного підходу, повторного використання коду, інкапсуляції логіки та розділення відповідальностей. Це дозволяє забезпечити підтримуваність коду, зручність масштабування та ефективну колективну розробку.

Таким чином, клієнтська частина інформаційної системи успішно спроектована, реалізована та випробувана відповідно до вимог технічного завдання. Забезпечено повноцінну інтерактивну взаємодію користувача з основними функціональностями системи, що створює передумови для подальшого розвитку, адаптації під інші задачі та ефективного використання у сфері управління програмними проєктами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ:

1. Ткаченко К. О., Мороз О. О. Типізація даних в JavaScript за допомогою TypeScript // Цифрова економіка та інформаційні технології : матеріали Міжнародної науково-практичної конференції, м. Київ, 2021 р. / М-во освіти і науки України, Державний університет інфраструктури та технологій. — Київ : Видавничий центр ДУІТ, 2021. — С. 100 - 102.
2. Джерело: Безверхий, О., Куценко, О. (2022). Ефективність застосування бібліотеки React. *Інформаційні технології та суспільство*, 2(4), 13–19. DOI: <https://doi.org/10.32689/maup.it.2022.2.2>
3. TypeScript Official Documentation. – [Електронний ресурс]. – Режим доступу: <https://www.typescriptlang.org/>
4. Advantages And Disadvantages Of ReactJS [Електронний ресурс]. – Режим доступу: <https://www.w3schools.blog/advantagesdisadvantages-reactjs>
5. React Hook Form Documentation. – [Електронний ресурс]. – Режим доступу: <https://www.react-hook-form.com/>
6. Formik Official Documentation. – [Електронний ресурс]. – Режим доступу: <https://formik.org/docs/overview>
7. Fetch API [Електронний ресурс]. – Mozilla Developer Network. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API
8. Сучасний підручник JavaScript. Розділ: Мережеві запити. Тема: Fetch [Електронний ресурс]. – Режим доступу: <https://uk.javascript.info/fetch>
9. React Router [Електронний ресурс]. – Офіційна документація. – Режим доступу: <https://reactrouter.com/>
10. Малкіна В. М., Зінов'єва О. Г., Мирошніченко М. Ю. Програмний модуль для створення навчальних засобів з використанням технології Drag-and-drop. *Ukrainian Journal of Educational Studies and Information Technology*, 2018, Vol. 6, No. 2, с. 8–15. Режим доступу: <https://core.ac.uk/download/pdf/541102255.pdf>
11. Pragmatik [Електронний ресурс]. – npmjs.org – Режим доступу: <https://www.npmjs.com/package/pragmatik>
12. Deutsch, S. (2023). *5 React Architecture Best Practices for 2024*. SitePoint. Режим доступу: <https://www.sitepoint.com/react-architecture-best-practices> (дата звернення: 20.06.2025).