

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

_____ Миронов Олексій Миколайович

Науковий керівник	доц. Самойленко Д. М.
-------------------	-----------------------

АНОТАЦІЯ

УДК: 004.415.2+004.415.3+004.43:005.8

Авторський знак: М-64

Миронов О.М. Інформаційна система гнучкого управління програмними проектами (на прикладі Trello): випускна кваліфікаційна робота на здобуття бакалаврського рівня вищої освіти зі спеціальності «122 Комп'ютерні науки». – Одеса: ОТУШ, 2025. – 24 с.

Робота присвячена організації життєвого циклу проєкту створення інформаційної системи ItRun — вебзастосунку для управління цифровими завданнями у форматі канбан-дошки. У межах проєкт-менеджмент частини реалізовано повний цикл планування, ведення документації, постановки задач, управління знаннями та синхронізації командної роботи на основі сучасних методологій та цифрових інструментів.

Актуальність дослідження зумовлена потребою в адаптивних, прозорих і технологічно гнучких підходах до управління ІТ-проєктами в умовах обмежених ресурсів та змінних вимог. У роботі проаналізовано сучасні підходи до управління розробкою, зокрема Waterfall, Scrum, XP, SAFe та інші, й обґрунтовано вибір методології Kanban як найбільш придатної до умов невеликої динамічної команди.

У роботі досліджено й впроваджено:

- модель Kanban-управління розробкою із застосуванням Trello;
- централізоване ведення документації в Notion з модульною структурою (архітектура, API, інструкції, зустрічі);
- принципи SMART-підходу до формулювання задач і user stories;
- використання Discord для синхронної взаємодії та Google Meet для регулярних стендапів;
- інтеграцію GitHub як інструменту контролю версій, CI/CD і зберігання технічних матеріалів.

Об'єктом дослідження виступає процес управління командною розробкою інформаційної системи, предметом — методології управління, документообіг, цифрові інструменти супроводу. У ході впровадження було досягнуто високої узгодженості роботи, прозорості процесів та масштабованості: організація проєкту дозволяє легко масштабувати систему та інтегрувати нових учасників без втрати контексту.

Методологічною основою дослідження є принципи системного аналізу, інженерії програмного забезпечення та Agile-підходи до управління проєктами. Робота містить 13 використаних джерел.

ЗМІСТ

ВСТУП _____	4
Технічне завдання до проєкту _____	5
РОЗДІЛ 1. Загальна характеристика системи _____	8
РОЗДІЛ 2. ОРГАНІЗАЦІЯ ЖИТТЄВОГО ЦИКЛУ РОЗРОБКИ ІНФОРМАЦІЙНОЇ СИСТЕМИ _____	10
2.1 Методологічна база управління розробкою _____	10
2.2 Управління знаннями та організація проєктної документації _____	14
РОЗДІЛ 3. РЕЗУЛЬТАТИ ТЕСТУВАННЯ ТА ПЕРЕВІРКИ ФУНКЦІОНУВАННЯ СИСТЕМИ _____	19
ВИСНОВКИ _____	23
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ _____	24

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 дипломної роботи головним напрямом дослідження стала організація управління життєвим циклом проєкту розробки інформаційної системи ItRun, відповідно до сучасних практик управління в IT-середовищі. Основним підходом до організації командної роботи обрано Kanban-модель, що дозволила гнучко реагувати на зміну пріоритетів, зберігати візуалізацію етапів виконання задач та підтримувати сталі темпи розробки. Документація усіх процесів, рішень, інтерфейсів та технічних вимог здійснювалась у середовищі Notion, що забезпечило єдину точку доступу до знань для всієї команди. Розроблено та підтримувано backlog проєкту, визначено user stories, технічні та нефункціональні вимоги, організовано регулярні ретроспективи й синхронізації між розробниками. Всі ці інструменти забезпечили високий рівень прозорості, керованості та адаптивності у процесі розробки складного програмного продукту.

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 проєкту 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 проєктами.

РОЗДІЛ 2 ОРГАНІЗАЦІЯ ЖИТТЄВОГО ЦИКЛУ РОЗРОБКИ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1 Методологічна база управління розробкою

Розробка складних інформаційних систем вимагає не лише технічної експертизи, а й чіткого управлінського підходу, який дозволяє забезпечити узгодженість командної роботи, гнучкість планування та своєчасну адаптацію до змін. Саме вибір методології управління є критичним фактором, що безпосередньо впливає на ефективність реалізації проєкту, його терміни, витрати й кінцеву якість продукту.

У сучасному програмному інжинірингу існує кілька базових підходів до управління життєвим циклом проєктів. До них належать каскадна модель (Waterfall), гнучка методологія (Agile), спіральна модель, модель V-цикл, інкрементна та ітеративна моделі, а також гібридні підходи, що поєднують їх елементи. Кожен із підходів має свої переваги, недоліки та галузі застосування.

У межах реалізації проєкту ItRun було прийнято рішення використовувати гнучкий підхід до управління (Agile). Гнучка методологія розроблення (Agile Software Development, agile) — це сукупність підходів, орієнтованих на ітеративне створення продукту, динамічне формування вимог і постійну взаємодію між членами самоорганізованої команди. До найвідоміших практик, що входять до складу Agile, належать Scrum, Kanban, Extreme Programming, DSDM тощо [1].

Переваги гнучкої методології полягають у зменшенні ризиків через поділ розробки на короткі ітерації, кожна з яких передбачає повний цикл — від аналізу вимог до тестування та документування. Наприкінці кожної ітерації команда має мінімально життєздатний продукт, готовий до демонстрації або навіть розгортання, що дозволяє швидко виявляти недоліки та враховувати нові вимоги [1].

Засадничі принципи гнучкої розробки закладено в Agile-маніфесті. Він декларує пріоритет взаємодії між людьми над жорсткими процесами, працездатного продукту — над вичерпною документацією, співпраці з замовником — над формальностями контракту, а також готовності до змін — над дотриманням жорсткого плану. Таким чином, Agile підхід ставить в центр процесу цінність, яку отримує користувач, і здатність команди оперативно адаптуватися [2].

Серед внутрішніх методологій Agile підходу було проаналізовано шість найпоширеніших: Scrum, Kanban, Extreme Programming, SAFe, DAD та Waterfall (для контрасту). Результати порівняльної характеристики наведено у таблиці 2.1.

Таблиця 2.1 – Аналіз методологій управління проєктами

Критерії	Waterfall	Scrum	XP	Kanban	AFe	AD
Передбачуваність та планування	+	-	-	-	+	+
Гнучкість та адаптивність	-	+	+	+	+	+
Простота впровадження	+	+	-	+	-	-
Масштабованість	-	+	-	+	+	+
Взаємодія з клієнтами	-	+	+	+	+	+
Управління ризиками	-	+	+	+	+	+
Швидкість виходу на ринок	-	+	+	+	+	+
Дотримання нормативних вимог	+	-	-	+/-	+	+
Ресурсоефективність	+/-	+/-	+/-	+	+/-	+/-
Безперервне вдосконалення	-	+	+	+	+	+

Проаналізувавши переваги та слабкі сторони кожного підходу, як основну методологію для реалізації проєкту було обрано **Kanban** — через його простоту, високу прозорість, адаптивність до змін та придатність до невеликих команд з нерегулярним ритмом надходження завдань.

Метод Kanban — це візуальна система управління завданнями в реальному часі, яка базується на принципах гнучкого виробництва Just-in-Time (JIT) [3]. Основним інструментом Kanban є дошка (Kanban board), що дозволяє відображати весь процес виконання задач у вигляді колонок: наприклад, "Заплановано", "У процесі", "Готово". Це дає змогу всім учасникам бачити навантаження команди та стан виконання задач у будь-який момент [4].

Ключовими перевагами Kanban у межах цього проєкту стали:

- **Прозорість процесів** — наочне відображення задач і прогресу дозволяє ефективно контролювати виконання.
- **Обмеження WIP (Work in Progress)** — обмеження кількості одночасно активних завдань запобігає перевантаженню команди та сприяє концентрації.
- **Гнучкість та адаптивність** — Kanban не вимагає фіксованих спринтів або ролей, що дозволяє швидко реагувати на зміни.
- **Постійне вдосконалення** — ітеративний перегляд процесів сприяє підвищенню ефективності з кожним циклом [5].

Візуальне представлення завдань і стадій їх виконання є ключовим елементом методології Kanban. На рисунку 2.1 зображено типовий приклад Kanban-дошки, яка складається з колонок, що відображають окремі етапи життєвого циклу завдань: від Backlog до Done.

Кожна колонка відповідає певному статусу виконання, а робочі елементи (Work Items) поділяються на типи, що маркуються різними відтінками. Таким чином, дошка дозволяє команді оперативно відстежувати прогрес, виявляти вузькі місця та оптимізувати завантаження учасників команди.

Лейни (lane) проходять горизонтально через всі колонки, дозволяючи групувати завдання за типами або відповідальними. Завдяки такому підходу до візуалізації робочого процесу підвищується прозорість, знижується ймовірність перевантаження та забезпечується гнучке управління потоком завдань у рамках обраної Agile-методології.

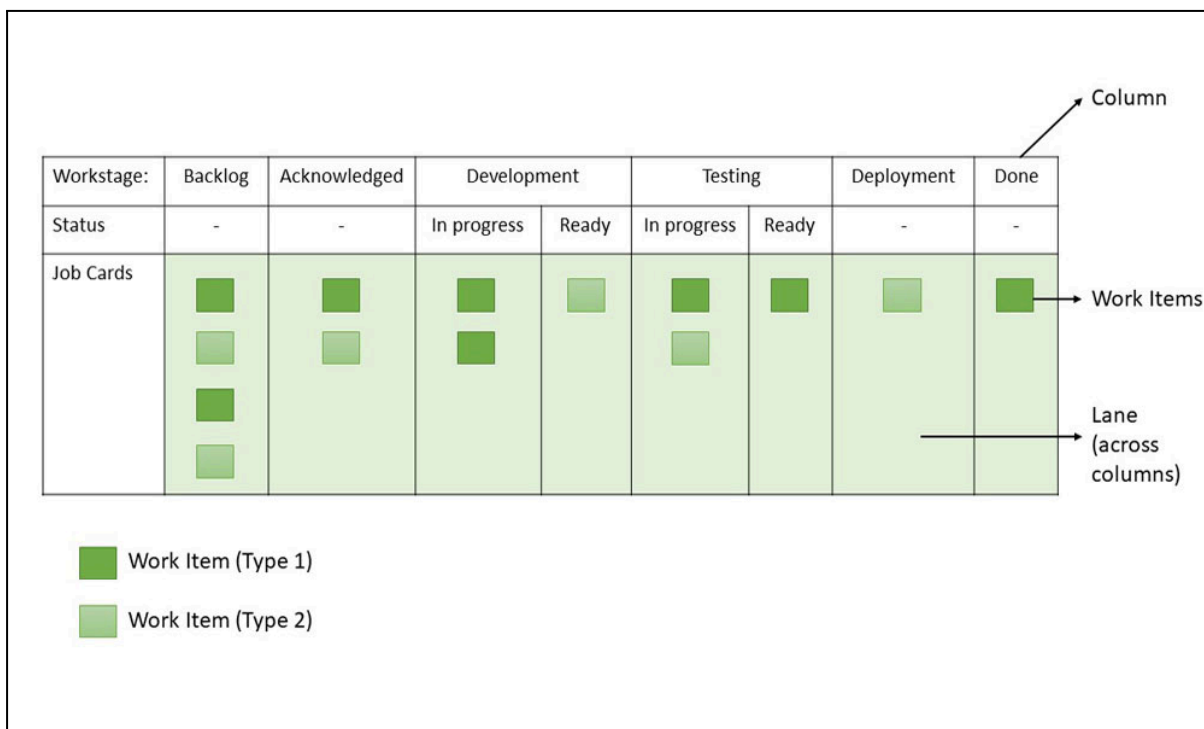


Рисунок 2.1 – Типова структура Канбан-дошки для візуалізації життєвого циклу завдань

Таким чином, Kanban виступив основою для побудови процесу управління задачами в системі ItRun. Обраний підхід дозволив ефективно поєднати організаційну простоту, технічну прозорість і гнучкість планування, що було особливо важливо в умовах обмежених ресурсів та невеликої команди.

У межах організації роботи над інформаційною системою ItRun як основний інструмент керування задачами та візуалізації прогресу було обрано Trello — онлайн-платформу, що реалізує принципи Kanban та забезпечує простий і зрозумілий інтерфейс для взаємодії з командою. Система дозволяє створювати дошки, колонки та картки, наочно відображаючи життєвий цикл кожного завдання. Такий підхід забезпечує прозорість роботи, гнучке планування і просте делегування задач у рамках Agile-проєкту.

Основними аргументами на користь вибору Trello стали інтуїтивна логіка побудови дошок, висока швидкість впровадження та підтримка спільної роботи. Як зазначає Joey Rosenberg, директор з глобального лідерства організації *Women Who Code*, Trello чудово підходить для спрощення складних процесів: «Я можу розбити [процеси] на невеликі фрагменти і делегувати їх учасникам команди, зберігаючи при цьому загальну картину всієї роботи». Крім того, за результатами дослідження TechValidate, 75 % організацій повідомляють, що Trello приносить перші результати їх бізнесу вже протягом 30 днів [6].

У межах організації комунікації між учасниками проєкту було обрано **Discord** — сучасну платформу для голосового, текстового та відеозв'язку, що забезпечує високу якість обміну інформацією в режимі реального часу. Основною причиною такого вибору є зручність організації тематичних каналів, що відповідають окремим напрямкам розробки (наприклад, backend, frontend, дизайн, DevOps), а також підтримка інтеграцій з іншими сервісами.

Discord дозволяє створювати спільноти (сервери) з ієрархічною структурою доступу, що забезпечує гнучке керування правами користувачів і приватністю обговорень. Завдяки можливості використання сповіщень, закріплення повідомлень, демонстрації екрана та запису зустрічей, платформа стала основним каналом оперативної комунікації як для регулярних стендапів, так і для екстрених обговорень критичних задач.

Таким чином, Discord виявився ефективним інструментом для підтримки синхронного обміну знаннями, що доповнював асинхронні засоби документування у Notion і керування завданнями в Trello [7].

Вибір методології Kanban у поєднанні з такими інструментами, як Trello та Discord, дозволив сформувати прозору, адаптивну й ефективну модель управління командною розробкою. Простота впровадження, гнучкість планування, візуалізація завдань і якісна синхронізація між учасниками забезпечили стійкий прогрес проєкту навіть за умов обмежених ресурсів і змінних вимог. Інтеграція сучасних платформ комунікації та управління стала основою злагодженого циклу створення програмного забезпечення.

2.2 Управління знаннями та організація проєктної документації

У сучасному IT-середовищі управління знаннями є критичним фактором ефективної командної розробки, особливо при створенні складних програмних систем. Успішна реалізація інформаційної системи потребує не лише технічної досконалості, а й систематичного підходу до документування архітектури, технічних рішень, розподілу обов'язків і процесів. Грамотна організація документації дозволяє забезпечити прозорість проєкту, повторюваність процесів та ефективну передачу знань між членами команди, зокрема під час масштабування або зміни складу розробників.

У межах дипломного проєкту особлива увага була приділена структурованому та всебічному документуванню кожного аспекту системи. Для кожного модуля було створено окрему сторінку в системі управління знаннями Notion, що містила опис функціональності, API-ендпоінтів, схем взаємодії з іншими компонентами, технічні специфікації та інструкції з використання. Такий підхід дозволив забезпечити незалежність модулів, спростити процес супроводу та забезпечити можливість швидкої інтеграції нових розробників до проєкту без втрати продуктивності. Завдяки високому рівню деталізації, проєкт може бути відтворений або масштабований у майбутньому з мінімальними витратами на аналіз.

Архітектура програмного забезпечення (англ. software architecture) — це структура програми або обчислювальної системи, яка включає програмні компоненти, видимі зовні властивості цих компонентів, а також відносини між ними. Цей термін також охоплює документування архітектури, що є важливим засобом комунікації між зацікавленими сторонами, фіксації рішень щодо високорівневого дизайну системи та повторного використання її шаблонів у майбутньому [8].

Процес архітектурного проєктування є початковим етапом створення системи, в якому визначається загальна структура, взаємозв'язки між підсистемами та логіка керування. Він виконує роль з'єднувальної ланки між вимогами до системи та її реалізацією, забезпечуючи базу для детальнішого проєктування окремих модулів. На цьому етапі формуються ключові підсистеми, визначаються їх функції, канали взаємодії та ролі в межах цілісної системи [9].

Архітектурне проєктування зазвичай передбачає декілька основних фаз: структурування (визначення незалежних підсистем), моделювання керування (визначення потоків взаємодії), модульну декомпозицію (виділення окремих модулів із чіткими межами відповідальності). Кожна з підсистем описується окремо, із зазначенням доступних сервісів та інтерфейсів. Це дозволяє створити надійну основу для реалізації, уникаючи дублювання коду та логічних помилок.

Залежно від потреб проєкту, архітектура може моделюватися за допомогою статичних, динамічних, інтерфейсних та інтерактивних моделей. Кожна з них виконує окрему функцію: демонстрацію структури, опис часу виконання, взаємозв'язки між елементами або сервіси, доступні для інших частин системи. В дипломному проєкті було використано комбінований підхід до архітектурного опису, який передбачав створення як UML-схем, так і структурованих текстових описів у Notion. Це

забезпечило гнучкість, масштабованість і високий рівень розуміння між усіма учасниками розробки.

Архітектура значною мірою впливає на нефункціональні характеристики системи: продуктивність, захищеність, безпеку, надійність і зручність супроводу. Наприклад, застосування модульного підходу дозволяє локалізувати збої, а багаторівнева структура – підвищити безпеку системи. У нашому проєкті ці принципи було реалізовано шляхом розділення логіки на окремі компоненти: бекенд, фронтенд, інтерфейс керування, базу даних, систему DevOps та управління проєктом. Це дозволило спростити інтеграцію та забезпечити високу якість під час проходження тестувань [9].

У процесі управління проєктом важливо не лише визначити задачі, але й сформулювати їх таким чином, щоб вони були зрозумілими, вимірюваними та орієнтованими на результат. У своїй роботі я керувався принципом SMART, який є загальноприйнятим стандартом у сучасному менеджменті. Цей підхід дозволяє формувати задачі таким чином, щоб вони максимально відповідали цілям проєкту та були контрольованими в процесі виконання.

В основі підходу SMART-цілі повинні відповідати таким критеріям:

1. Specific (конкретний) – чітко описана дія, яку необхідно виконати;
2. Measurable (вимірюваний) – наявність кількісних або якісних індикаторів досягнення;
3. Achievable (досяжний) – обґрунтована реалізація у межах наявних ресурсів;
4. Relevant (актуальний) – відповідність меті та контексту проєкту;
5. Time-bound (обмежений часом) – визначений термін виконання завдання [10].

Notion обрано як головний інструмент для зберігання та організації проєктної документації завдяки його універсальності, гнучкості й простоті використання. Notion дозволяє поєднати формат баз даних, вікі, таблиць і візуальних структур в єдиному просторі, що особливо зручно в умовах колективної розробки.

Notion — це потужний універсальний робочий простір, призначений для нотаток, завдань, контент-планів, стратегій і ще багатьох різних функцій. Він є ефективним інструментом управління проєктами, а також покращує командну співпрацю, обмін знаннями, інсайтами та продуктивністю [11].

Серед ключових переваг використання даної платформи варто виокремити такі аспекти:

- Наявність інтуїтивно зрозумілого інтерфейсу, що дозволяє адаптувати структуру документів відповідно до специфіки проєкту; гнучкість у створенні й кастомізації шаблонів, які охоплюють як технічні, так і організаційні аспекти розробки;
- Підтримка мультимедійного контенту (зображення, відео, гіперпосилання, код), що розширює функціонал традиційних текстових систем; наявність інструментів фільтрації, сортування та пошуку, що спрощує доступ до потрібної інформації;

- Можливість представлення інформації у різних візуальних форматах: таблиці, календаря, галереї, списку та канбан-дошки;
- Режим спільного редагування в реальному часі, що сприяє ефективній командній взаємодії [12].

Документація до інформаційної системи ItRun в середовищі Notion була структурована відповідно до модульної архітектури самого програмного забезпечення. Кожен розділ документації відповідає окремому аспекту проєкту, що забезпечує логічну ієрархію та зручність навігації. Схематично структура документації може бути представлена наступним чином, що зазначений на рисунку 2.2:

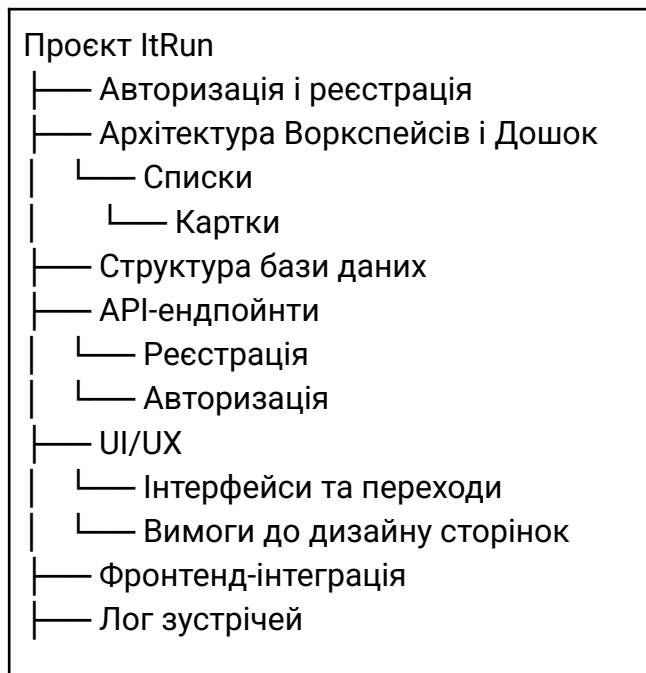


Рисунок 2.2 – Ієрархічна модель документаційної структури проєкту ItRun.

Зазначена структура цифрового середовища управління документацією, побудована в Notion, дала змогу забезпечити логічну модульну декомпозицію знань із чітким розподілом за відповідними напрямками розробки. Такий підхід дозволяє не лише структуровано зберігати інформацію, але й ефективно її оновлювати, масштабувати, а також здійснювати швидку навігацію для всіх учасників команди — розробників, дизайнерів, DevOps-спеціалістів, аналітиків тощо. Формування централізованої системи знань в електронному середовищі також значно полегшує процес залучення нових учасників у проєкт, адже надає їм повний контекст та необхідні матеріали для ознайомлення з архітектурою системи, бізнес-логікою та інтерфейсами взаємодії.

Ключовим елементом документаційної підтримки розробки стало уніфіковане описання API-ендпойнтів, яке містить чіткі специфікації кожного маршруту, його параметри, відповідні типи запитів, очікувані відповіді та бізнес-логіку, що реалізується на сервері. Такий підхід сприяє узгодженості між фронтенд- і бекенд-розробниками, а також забезпечує відтворюваність програмного рішення в

довгостроковій перспективі, включаючи супровід, масштабування й інтеграцію нових функціональних блоків.

У таблиці нижче наведено приклад опису API, що обслуговує процес поетапної реєстрації користувача:

Таблиця 2.2. Фрагмент API-специфікації модуля реєстрації користувача.

Етап	HTTP-запит	Опис функціоналу
Step 1	POST <code>/api/register-step1</code>	Приймає електронну пошту, перевіряє її унікальність
Step 2	POST <code>/api/register-step2</code>	Отримує ім'я, пароль та email, хешує пароль, створює користувача, надсилає verification token
Step 3	POST <code>/api/register-step3</code>	Завантажує аватар користувача, зберігає посилання на зображення
Step 4	POST <code>/api/register-step4</code>	Фіксує вибір мети використання системи
Verify	GET <code>/api/verify-email</code>	Отримує токен із посилання, верифікує обліковий запис

GitHub виступив як ключовий інструмент для управління версіями, координації роботи команди та інтеграції коду в рамках розробки інформаційної системи ItRun. Через функціонал pull requests було реалізовано внутрішній контроль якості, що дало змогу рецензувати код до його включення в основну гілку. Інструменти гілкування (branching) дозволили ефективно розділяти завдання на ізольовані фрагменти, мінімізуючи конфлікти та забезпечуючи паралельну роботу над окремими модулями системи.

Використання GitHub у межах розробки проєктів забезпечує низку істотних переваг, які роблять його одним із найпотужніших інструментів для організації командної роботи та контролю версій. По-перше, платформа дає змогу зручно зберігати вихідний код, супроводжувати його документацією у форматі Markdown і підтримувати контроль змін у багатокористувацькому середовищі. Завдяки системі pull requests та історії комітів, GitHub дозволяє легко відслідковувати, обговорювати й впроваджувати зміни в коді, що критично важливо при роботі над динамічними проєктами. По-друге, GitHub інтегрується з низкою сторонніх сервісів (зокрема Google Cloud та Amazon), що розширює можливості автоматизації CI/CD-процесів. Крім того, платформа підтримує інструменти аналітики коду, забезпечує наочну демонстрацію портфоліо розробника та підвищує шанси працевлаштування через відкритість до спільноти [13].

Також використовувалась GitHub Actions для автоматизації CI/CD-процесів. Це дозволяло автоматично запускати тести та збірку після кожного оновлення, забезпечуючи стабільність основної гілки проєкту. Крім того, GitHub зручний для ведення базової документації у форматі README, яка містила вступ до проєкту,

інструкції для локального розгортання, специфікацію технологій та залежностей, що значно полегшило онбординг для нових учасників.

У межах проєкту ItRun було реалізовано комплексну модель управління знаннями, що охоплює документаційний супровід, архітектурне моделювання, API-специфікацію, організацію цифрового простору (Notion), а також синхронізацію коду (GitHub). Така інтегрована система дозволила забезпечити прозорість процесів, уніфікований підхід до ведення документації та зручну навігацію в інформаційному середовищі проєкту. Завдяки цьому ItRun став не лише технічно реалізованим, а й повністю документованим продуктом, готовим до масштабування, тестування та довгострокової підтримки.

РОЗДІЛ 3 РЕЗУЛЬТАТИ ТЕСТУВАННЯ ТА ПЕРЕВІРКИ ФУНКЦІОНУВАННЯ СИСТЕМИ

У процесі дослідження та впровадження інформаційної системи ItRun було проведено серію випробувань на предмет працездатності інструментів управління проектом, синхронізації учасників і забезпечення цілісності технічної документації. Метою даного етапу було перевірити ефективність обраного інструментарію (Trello, GitHub, Notion, Discord) у підтримці життєвого циклу програмного проекту в умовах реального командного середовища.

На рис. 3.1 зображено канбан-дошку Trello, яка відображає етапи виконання завдань у межах розробки. Колонки Backlog, Design, To-Do, Doing, Testing та Done дали змогу ефективно організувати роботу над задачами відповідно до обраної методології Kanban. Такий підхід забезпечив прозорість процесу, видимість пріоритетів і контроль за поточним станом кожного модуля або компонента.

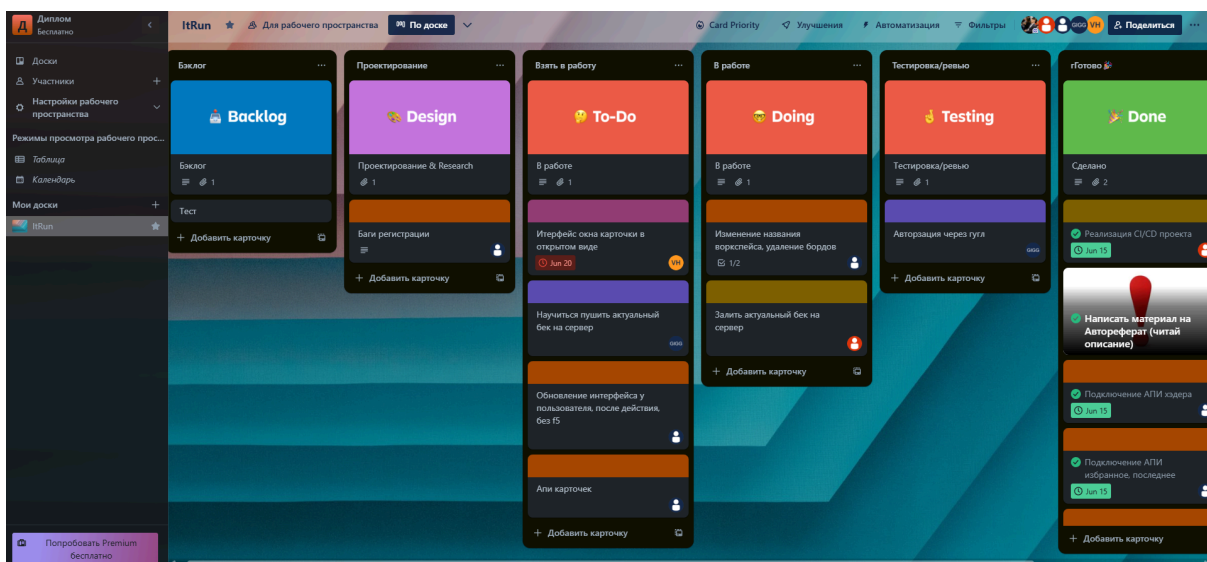


Рисунок 3.1 – Канбан-дошка Trello для управління задачами

Інструмент GitHub було використано як централізоване середовище для зберігання, контролю версій і супроводу вихідного коду. Репозиторії було розділено за функціональним принципом: один обслуговував фронтенд-частину, а інший — бекенд. Такий підхід дозволив забезпечити незалежний розвиток модулів системи, а також спростити тестування та деплой окремих компонентів. В обох репозиторіях велося версіювання із застосуванням гілок `main` і періодичних комітів, що відображає хронологію змін. Окрім цього, на платформі було організовано командний доступ із зазначенням авторів змін та інтеграцією з іншими сервісами. Рис. 3.2 демонструє дашборд користувача з двома основними репозиторіями проекту:

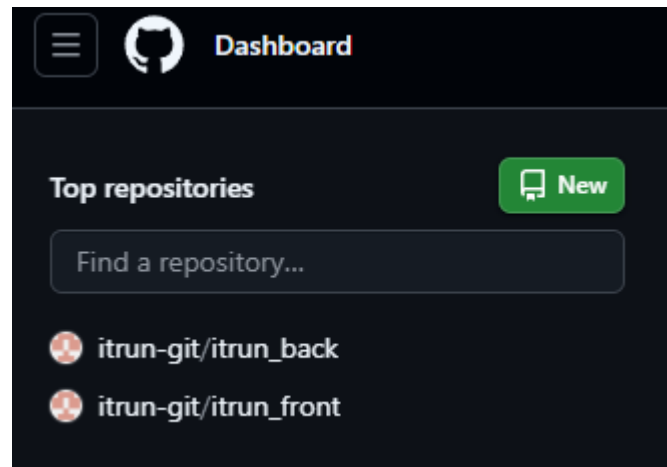


Рисунок 3.2 – Панель GitHub з підключеними репозиторіями *itrunkit_front* та *itrunkit_back*

У репозиторії *itrunkit_front* (рис. 3.3) чітко структуровано логіку клієнтської частини застосунку: виокремлено конфігураційні файли (**Dockerfile**, **docker-compose.yml**), каталоги з вихідним кодом (**src**) та публічні ресурси (**public**). Крім того, до складу репозиторію входять файли конфігурацій для лінтерів, залежностей та веб-сервера (**nginx.conf**, **package.json**), що підтверджує застосування сучасних практик розробки, відповідність вимогам до масштабованості та підтримки CI/CD. Наявність документаційного файлу **README.md** забезпечує швидке ознайомлення з архітектурою проєкту нових учасників команди.

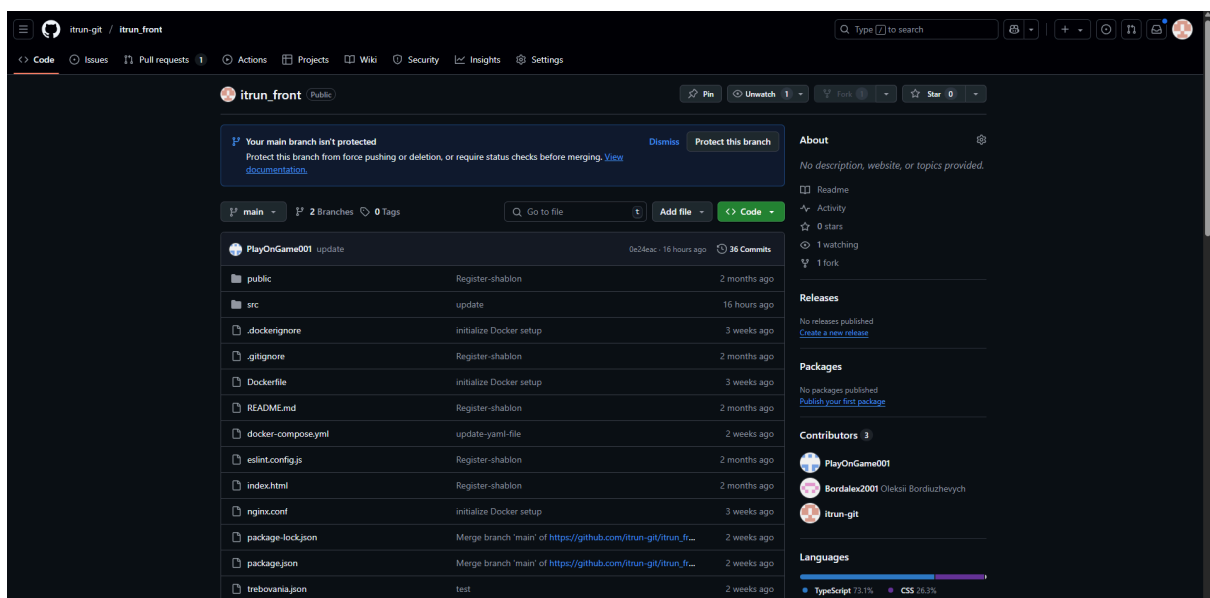


Рисунок 3.3 – Структура репозиторію клієнтської частини на GitHub

Окрему увагу було приділено перевірці організації середовища знань у Notion. У рамках проєкту було створено централізовану структуру з базами даних для API-документації, архітектури, задач, ретроспектив і чеклістів. Була протестована можливість одночасного редагування сторінок командою та використання шаблонів для прискорення формування нових сторінок. Оновлення в режимі реального часу

працювали стабільно, що підтвердило надійність сервісу для спільного ведення документації.

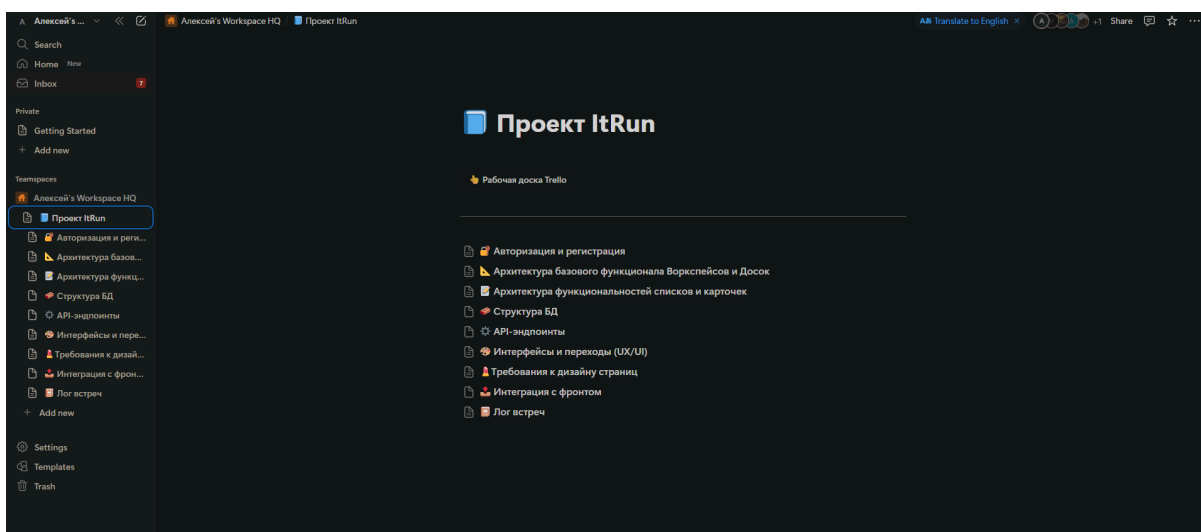


Рисунок 3.3 – Простір знань Notion для проекту ItRun

Крім цього, було здійснено апробацію інструментів синхронізації — Google Meet як платформи для мітингів, а також Discord як середовища для оперативних обговорень. Протягом спринтів проводилися щоденні стендапи, результати яких фіксувалися в окремій базі зустрічей у Notion. Голосові канали Discord продемонстрували стабільне з’єднання, можливість створення тематичних потоків і каналів для обговорення конкретних підзадач.

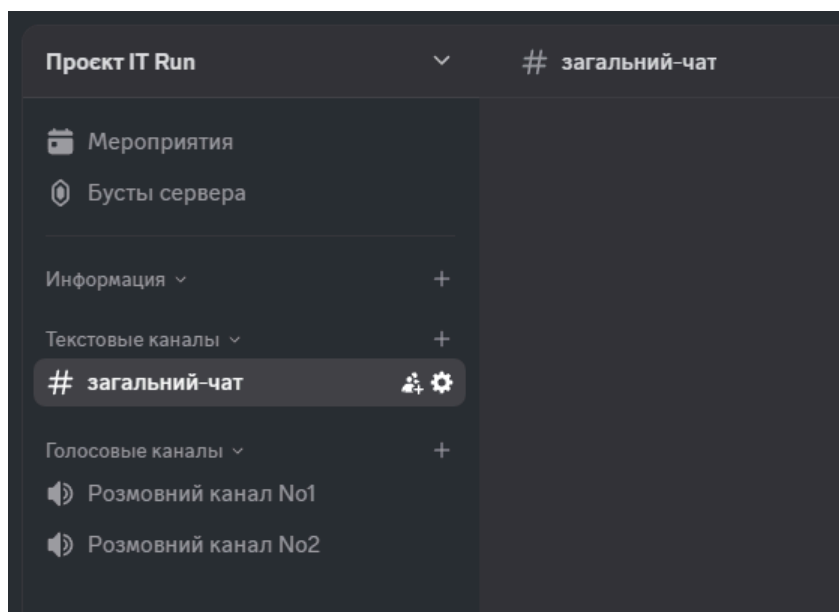


Рисунок 3.4 – Канали командної взаємодії в Discord

Таким чином, результати випробувань підтвердили ефективність розробленої інформаційної системи ItRun у контексті командної взаємодії, автоматизації процесів і підтримки знань. Усі інструменти — Trello, GitHub, Notion та Discord — функціонують у комплексі, доповнюючи один одного, що дозволяє масштабувати проєкт і забезпечувати сталу продуктивність команди незалежно від зовнішніх умов.

У результаті впровадження засобів управління завданнями, документацією та кодом — Trello, Notion, GitHub і Discord — було забезпечено повний цикл організації роботи команди розробки інформаційної системи *ItRun*. Кожен інструмент ефективно виконав покладену на нього функцію: Trello слугував візуальним центром планування задач, Notion — сховищем знань і технічної документації, GitHub — основою для контролю версій і спільної розробки, а Discord — платформою для безперервної комунікації.

Проведені випробування підтвердили працездатність інтегрованого середовища розробки та управління, що відповідає сучасним вимогам до організації командної роботи в IT-проектах.

ВИСНОВКИ

У межах виконання дипломного проєкту було спроектовано, організовано та реалізовано інформаційну систему управління програмними проєктами ItRun, яка об'єднує сучасні підходи до управління задачами, документацією, комунікацією та контролем версій. Проєкт охопив усі ключові етапи життєвого циклу ПЗ — від вибору методології управління до практичного впровадження системи з подальшим тестуванням її працездатності.

На етапі проєктування було обґрунтовано вибір гнучкої методології управління Kanban, яка забезпечила прозору організацію процесів, швидку адаптацію до змін та ефективну координацію дій у невеликій команді. У ролі основного інструменту візуалізації завдань було обрано Trello, що дозволило реалізувати Kanban-дошку з підтримкою гнучкого планування, розподілу обов'язків та контролю навантаження.

У системі було реалізовано повноцінну інфраструктуру управління знаннями. Зокрема, для збереження документації, архітектурних схем, специфікацій API та технічних матеріалів використовувався сервіс Notion. Це забезпечило структуроване середовище для підтримки та передачі знань, що критично важливо для масштабування та подальшої підтримки проєкту.

Репозиторії з вихідним кодом клієнтської та серверної частин було розміщено на GitHub, що дало змогу ефективно контролювати версії, координувати колективну розробку та забезпечити прозорість у веденні історії змін. Платформа GitHub також стала основою для CI/CD-процесів, що дозволили автоматизувати оновлення і доставку коду на серверне середовище.

Командна взаємодія підтримувалась через платформу Discord, яка забезпечила швидкий обмін інформацією, організацію тематичних каналів і зручне проведення обговорень, що суттєво підвищило ефективність синхронної та асинхронної комунікації між учасниками проєкту.

Проведене комплексне тестування підтвердило працездатність усіх компонентів інформаційної системи. Інтеграція Trello, Notion, GitHub і Discord дозволила досягти високого рівня узгодженості, прозорості та технічної готовності системи до використання. ItRun продемонструвала стабільну роботу, готовність до масштабування та відповідність сучасним вимогам до організації командної розробки в IT-сфері.

Таким чином, дипломний проєкт реалізував повноцінний цикл створення, організації та підтримки інформаційної системи управління проєктами. Отримані результати підтверджують доцільність застосування вибраних технологій та методологій у розробці сучасного програмного забезпечення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ:

1. Кучеров Д. П., Артамонов Є. Б. Інженерія програмного забезпечення: навч. посібник. Київ : НАУ, 2017. 388 с.
2. Маніфест гнучкої розробки програмного забезпечення [Електронний ресурс] – Режим доступу: <https://agilemanifesto.org/iso/uk/manifesto.html>
3. Anderson D. J. *Kanban: Successful Evolutionary Change for Your Technology Business*. Blue Hole Press, 2010.
4. Kanban Tool. *Unlocking Efficiency in Six Steps*. [Електронний ресурс] – Режим доступу: <https://kanbantool.com/kanban-method>
5. Atlassian. *Kanban methodology*. [Електронний ресурс] – Режим доступу: <https://www.atlassian.com/agile/kanban>
6. TechValidate. Оцінка ефективності платформи Trello: результати опитування серед користувачів [Електронний ресурс] // SurveyMonkey: Customer stories & advocacy. — Режим доступу: <https://www.surveymonkey.com/use-cases/customer-stories-and-advocacy/>
7. Discord [Електронний ресурс] : офіційний сайт платформи для голосового, текстового та відеозв'язку. — Режим доступу: <https://discord.com/> — Назва з екрана. — Дата звернення: 22 червня 2025 р.
8. Архітектура та проектування програмного забезпечення: опорний конспект лекцій / Західноукраїнський національний університет, Факультет комп'ютерних інформаційних технологій. — [Електронний ресурс]. — Режим доступу: <https://dspace.wunu.edu.ua/bitstream/316497/24194/1/опорний%20конспект%20лекцій.pdf>
9. Кучеров Д. П., Артамонов Є. Б. Інженерія програмного забезпечення: навч. посібник. — Київ : НАУ, 2017. — 388 с.
10. Башинська І.О. SMART-підхід до визначення цілей смартизації промислового підприємства // Економіка та управління підприємствами. — Одеський національний політехнічний університет..
11. Дімура М. Notion: що це і як користуватися [Електронний ресурс]. — 14.05.2025. — Режим доступу: <https://www.site2b.ua/ua/web-blog-ua/notions-shho-ce-i-yak-koristuvatisya.html>
12. Шиманська В. Робота з Notion: це варто знати [Електронний ресурс]. — 27.10.2023. — Режим доступу: <https://www.theantmedia.com/post/robota-z-notion-ce-var-to-znati>
13. GitHub Kullanmanın 7 Önemli Faydası [Електронний ресурс]. — Режим доступу: <https://www.abakuskıtap.com/en/blog/content/7-important-benefits-of-using-github-45>