ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка програмного забезпечення для моніторингу наукової роботи студентів»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Владислав САДЛІВСЬКИЙ
(підпис)

Виконав: здобувач вищої освіти групи ПД-44
Владислав САДЛІВСЬКИЙ

Керівник: Юрій ЗАДОНЦЕВ
к.т.н.

Рецензент: _____

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
«_____» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Садлівському Владиславу Руслановичу

1. Тема кваліфікаційної роботи: «Розробка програмного забезпечення для моніторингу наукової роботи студентів»
керівник кваліфікаційної роботи к.т.н., доцент кафедри ІПЗ Юрій ЗАДОНЦЕВ,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025 р. № 56.

2. Строк подання кваліфікаційної роботи «02» червня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про методи організації моніторингу наукової діяльності студентів, огляд сучасних веб-технологій та архітектурних підходів до побудови інформаційних систем, опис принципів розмежування ролей користувачів та організації процесу рецензування, технічна документація щодо використання фреймворку Spring Boot, засобів керування залежностями Maven, бази даних MySQL, а також бібліотек і шаблонізаторів для створення веб-інтерфейсу.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Огляд застосунку для моніторингу наукової діяльності студентів та аналіз існуючих аналогів, визначення вимог до програмного забезпечення.

2. Огляд та аналіз засобів реалізації web-додатку для моніторингу наукової діяльності студентів.
3. Проектування архітектури застосунку для моніторингу наукової діяльності студентів.
4. Програмне втілення та тестування застосунку для моніторингу наукової діяльності студентів.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до додатку.
3. Програмні засоби реалізації.
4. Діаграма варіантів використання.
5. Діаграма діяльності.
6. Діаграма класів
7. Схема бази даних
8. Екранні форми.
9. Апробація результатів дослідження

6. Дата видачі завдання «25» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	25.02.-04.03.2025	
2	Аналіз та дослідження існуючих аналогів	05.03-13.03.2025	
3	Огляд та аналіз існуючих рішень для моніторингу наукової діяльності студентів	14.03-20.03.2025	
4	Проєктування вимог до програмного забезпечення для моніторингу наукової діяльності студентів	21.03-27.03.2025	
5	Огляд засобів реалізації програмного забезпечення для моніторингу наукової діяльності студентів	28.03-17.04.2025	
6	Проєктування та розробка програмного забезпечення для моніторингу наукової діяльності студентів	18.04-28.05.2025	
7	Оформлення роботи: вступ, висновки, реферат	29.04-11.05.2025	
8	Розробка демонстраційних матеріалів	12.05-18.05.2025	
9	Попередній захист роботи	19.05-30.05.2025	

Здобувач вищої освіти _____ Владислав САДЛІВСЬКИЙ
(підпис)

Керівник
кваліфікаційної роботи _____ Юрій ЗАДОНЦЕВ
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 70 стор., 5 табл., 23 рис., 39 джерел.

Мета дослідження — вдосконалення процесу моніторингу наукової діяльності студентів.

Об'єкт дослідження — процес організації та цифрового моніторингу наукової діяльності студентів у закладах вищої освіти.

Предмет дослідження — веб-застосунок для реєстрації, перегляду та оцінювання наукових робіт студентів із підтримкою ролей користувачів.

Короткий зміст роботи: у роботі проаналізовано сучасні підходи до моніторингу наукової діяльності студентів у закладах вищої освіти, виділено основні проблеми організації контролю та супроводу студентських наукових робіт. Запропоновано власний веб-застосунок, що дозволяє автоматизувати процес створення, рецензування, коментування та адміністрування студентських наукових робіт через зручний веб-інтерфейс. В основі реалізації використано архітектуру Spring Boot + MySQL + Tailwind CSS, розроблено механізми реєстрації користувачів, розмежування ролей, коментування окремих фрагментів тексту роботи та відстеження статусу проєкту. Практична реалізація показала можливість підвищення ефективності взаємодії між студентами та науковими керівниками, а також забезпечення прозорого контролю за процесом підготовки та перевірки наукових робіт.

Сферою використання застосунку є організація та моніторинг наукової діяльності студентів.

КЛЮЧОВІ СЛОВА: МОНІТОРИНГ НАУКОВОЇ ДІЯЛЬНОСТІ, СТУДЕНТСЬКІ РОБОТИ, КОМЕНТУВАННЯ, ВЕБ-ЗАСТОСУНОК, РОЛЬОВА МОДЕЛЬ, JAVA, MAVEN, SQL, SPRING BOOT.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API — Інтерфейс програмування додатків (Application Programming Interface).

DTO — Об’єкт передавання даних (Data Transfer Object).

LaTeX — Мова розмітки даних та пакет макросів для набору наукових текстів.

MEU — Метод корисності очікування (Maximum Expected Utility).

NB — Наївний підхід Баєса (Naive Bayes).

OS — Операційна система (Operating System).

UI — Користувацький інтерфейс (User Interface).

UX — Досвід користувача (User Experience).

WebGL — Веб-графічна бібліотека (Web Graphics Library).

HTML — Мова гіпертекстової розмітки (HyperText Markup Language).

CSS — Каскадні таблиці стилів (Cascading Style Sheets).

JS — Мова програмування JavaScript.

ER-діаграма — Діаграма зв'язків сутностей (Entity-Relationship Diagram).

UML — Уніфікована мова моделювання (Unified Modeling Language).

MVC — Архітектурний шаблон “модель–подання–контролер” (Model–View–Controller).

DB — База даних (Database).

SQL — Мова структурованих запитів (Structured Query Language).

CRUD — Операції створення, читання, оновлення, видалення (Create, Read, Update, Delete).

REST — Стиль архітектури програмного інтерфейсу (Representational State Transfer).

JWT — Стандартизований формат токену доступу (JSON Web Token).

ВСТУП

Актуальність: сучасний стан розвитку освіти висуває нові вимоги до організації наукової діяльності студентів у закладах вищої освіти. В умовах цифровізації дедалі більшої актуальності набувають автоматизовані засоби моніторингу, що дозволяють підвищити якість супроводу курсових, бакалаврських і дипломних робіт, забезпечити прозорість оцінювання та ефективну взаємодію між студентами й науковими керівниками. Традиційні підходи до оцінювання й контролю часто не відповідають потребам сучасного освітнього процесу — вони трудомісткі, малогнучкі та не дозволяють своєчасно реагувати на зміни у підготовці наукових проєктів.

Запровадження веб-застосунку для централізованого зберігання, коментування та рецензування наукових робіт дозволяє автоматизувати ключові процеси, підвищити ефективність і прозорість взаємодії між усіма учасниками освітнього процесу.

Об'єктом дослідження є процес організації та цифрового моніторингу наукової діяльності студентів у закладах вищої освіти.

Предметом дослідження виступає веб-застосунок для реєстрації, перегляду, коментування та оцінювання наукових робіт студентів із підтримкою ролей користувачів (студент, викладач, адміністратор).

Метою роботи є вдосконалення процесу моніторингу наукової діяльності студентів шляхом створення та впровадження спеціалізованого веб-застосунку, що дозволяє автоматизувати всі основні етапи — від реєстрації до рецензування та контролю статусу роботи.

Методами дослідження стали аналіз і порівняння сучасних освітніх платформ та інструментів, формалізація вимог, побудова UML- та ER-діаграм, розробка веб-застосунку на основі Java, Spring Boot, MySQL, Tailwind CSS та проведення тестування за участю реальних користувачів.

Науковою новизною роботи є створення веб-застосунку з підтримкою гнучкої ролевої моделі, механізмами коментування окремих фрагментів тексту, відстеженням статусів роботи та прозорою історією змін. Запропоновано архітектурне рішення, якого бракує в більшості існуючих освітніх платформ, з орієнтацією на реальні сценарії взаємодії студентів і наукових керівників.

Практична значущість результатів полягає у можливості впровадження розробленої системи в закладах вищої освіти для супроводу студентських наукових робіт. Це дозволяє підвищити якість комунікації, зменшити адміністративне навантаження, забезпечити контроль дедлайнів і прозорість оцінювання.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Актуальність проблеми

Наукова діяльність студентів є невід'ємною частиною освітнього процесу. Саме вона дозволяє не лише поглибити фахові знання, а й сформувати навички самостійного дослідження, критичного аналізу та системного мислення.

Курсова, дипломна та інша дослідницька робота передбачає постійну взаємодію між студентом і науковим керівником, а також потребує точного обліку прогресу, оперативного обміну інформацією та об'єктивного оцінювання результатів. Водно час способи, якими це організовується сьогодні, часто є застарілими: паперові звіти, електронна пошта чи таблиці створюють додаткові труднощі, уповільнюють роботу, ускладнюють доступ до даних і призводять до плутанини в комунікації. У сучасному інформаційному середовищі ці проблеми особливо відчутні. Виникає очевидна потреба в рішенні, яке дозволить перевести ці процеси в цифрову площину.

Створення веб-застосунку, що може взяти на себе функції моніторингу, обміну коментарями, збереження й оновлення інформації про наукові роботи, дозволить істотно підвищити якість організації навчального процесу. Така система повинна бути зрозумілою для користувачів з різним рівнем технічної підготовки. Важливо, щоб спілкування між студентами та викладачами відбувалося швидко та без зайвих бар'єрів. При цьому захист особистих даних повинен залишатися пріоритетом, адже мова йде про конфіденційну інформацію, яка стосується і навчання, і персональних результатів. Не менш важливим є те, щоб усі дії в системі були зручними, а реагування на зміни в роботах — максимально оперативним.

Отже, в умовах цифровізації освіти розробка подібного веб-застосунку виглядає не лише доречною, а й необхідною. Вона дасть змогу осучаснити процес підготовки студентських наукових робіт, зробити його більш прозорим і результативним, а також сприятиме підвищенню загальної якості освітніх послуг,

в умовах цифровізації освіти розробка подібного веб-застосунку виглядає не лише доречною, а й необхідною. Вона дасть змогу осучаснити процес підготовки студентських наукових робіт, зробити його більш прозорим і результативним, а також сприятиме підвищенню загальної якості освітніх послуг. На рисунку 1.1 представлено загальний життєвий цикл веб-застосунку

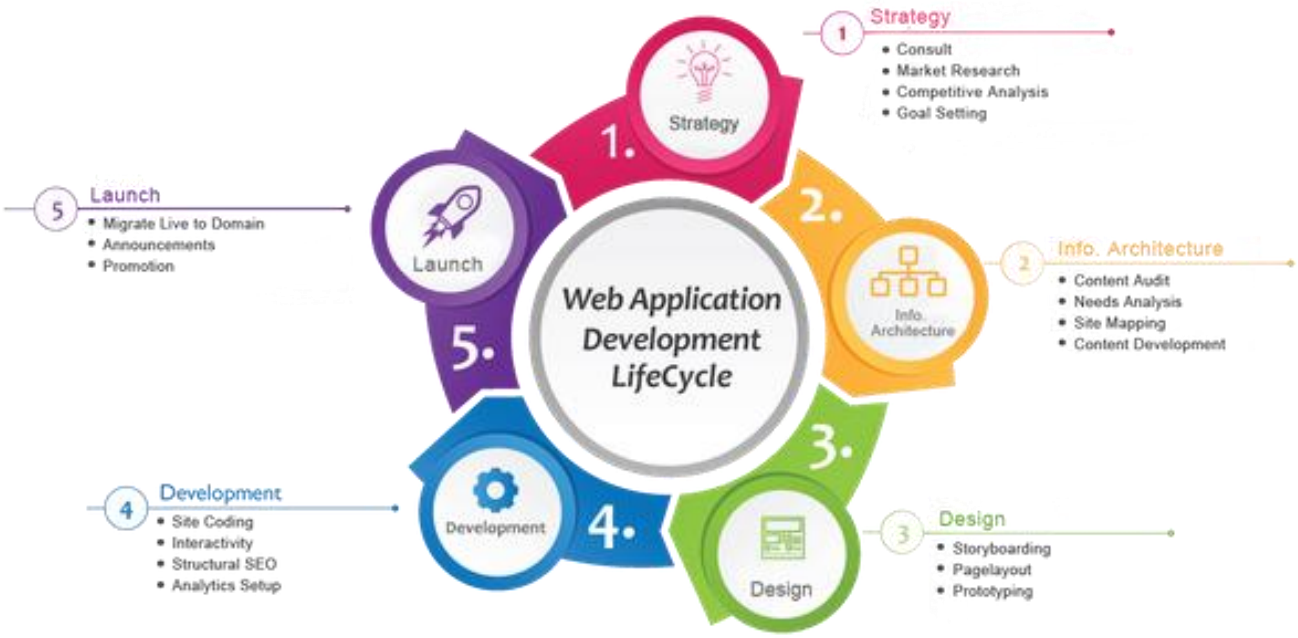


Рис. 1.1 Загальний життєвий цикл веб-застосунку

Отже, організація моніторингу наукової роботи студентів — це, по суті, проєкт зі створення злагодженої системи, яка дозволяє зберігати, обробляти та аналізувати всю інформацію, пов’язану з дослідницькою діяльністю. Цей процес охоплює не лише технічний бік — як-от облік тем, версій документів чи хронології змін, — а й питання зручності користування та взаємодії між усіма учасниками: студентами і викладачами.

Особливу складність становить великий обсяг даних, які постійно змінюються: нові теми, доповнення, зауваження від керівників, зміни статусів. Усе це потрібно не просто зберігати, а ще й представляти в такому вигляді, щоб кожен користувач міг швидко знайти потрібну інформацію. І тут постає завдання створити систему, яка була б інтуїтивно зрозумілою навіть для тих, хто не має

спеціальної технічної підготовки. Спілкування в межах такої платформи також повинно бути організоване так, щоб викладачі могли зручно залишати коментарі до окремих частин роботи, вносити пропозиції чи зауваження без необхідності складних дій з боку студентів. Не можна оминути увагою й питання безпеки. Платформа, яка зберігає дані користувачів та документи освітнього характеру, має бути захищеною від витоків, несанкціонованого доступу та випадкових втрат[1]. Система повинна надійно фіксувати всі дії: хто і коли створив чи змінив роботу, який статус вона має, що коментували викладачі — і все це з можливістю простого аналізу.

У межах такої системи кожна роль матиме свої можливості: студенти зможуть створювати та оновлювати роботи а викладачі — перевіряти та коментувати їх. На рисунку 1.2 зображено схему взаємодії ролей користувачів у розробленому веб-застосунку.

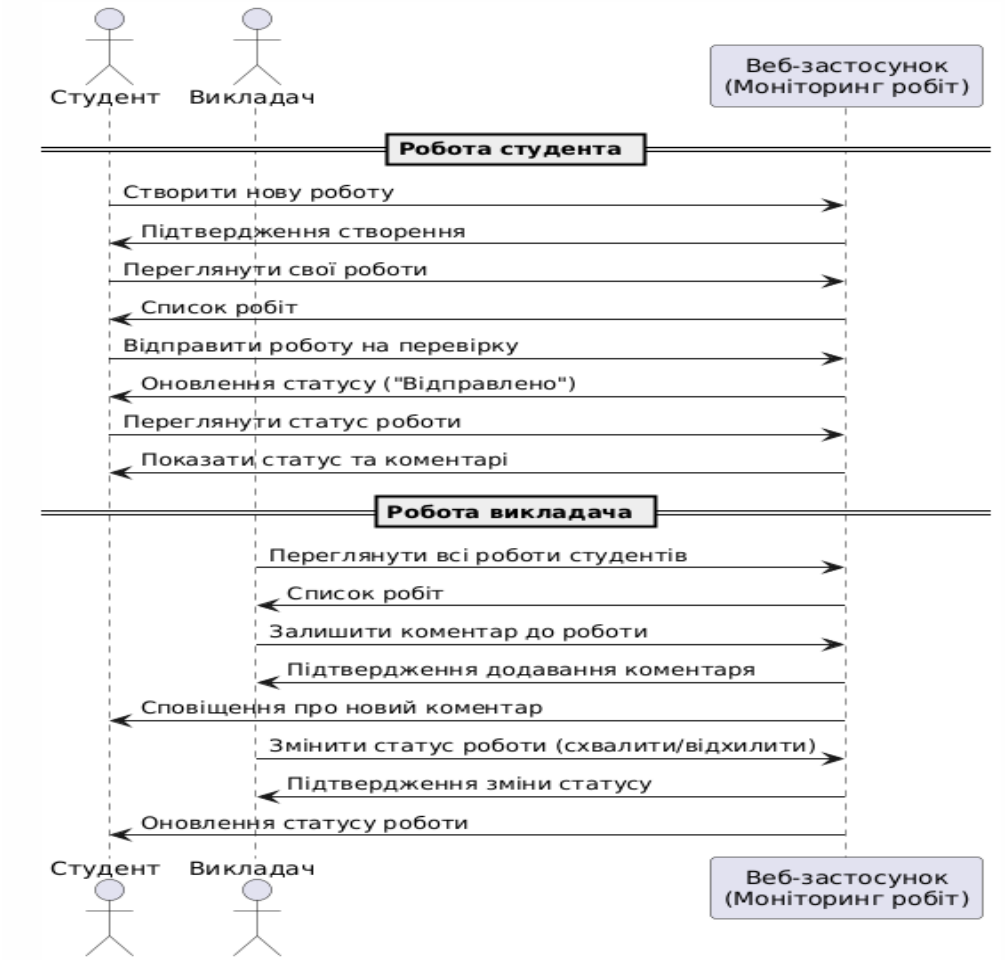


Рис. 1.2 Схема взаємодії ролей користувачів у веб-застосунку

У результаті впровадження такого веб-застосунку дозволяє не лише автоматизувати рутинні процеси, а й створити середовище, в якому взаємодія між усіма учасниками навчання буде комфортною, ефективною і справді сучасною.

1.2 Аналіз аналогів

У сучасному освітньому середовищі існує широкий вибір цифрових інструментів, які частково вирішують завдання з управління студентською науковою діяльністю — від обліку та перевірки робіт до спільного редагування та комунікації. Проте жоден із них не забезпечує повний комплекс функцій, необхідних для ефективного контролю та супроводу студентських досліджень.

Для розробки спеціалізованого веб-застосунку доцільно провести огляд найбільш популярних систем, оцінити їх можливості, сильні сторони та обмеження. У цьому розділі розглянуто чотири інструменти — Google Classroom, Notion, Overleaf і Miro — які частково можуть бути використані для організації роботи з науковими проєктами студентів.

Основними критеріями для оцінювання були:

— підтримка ролей користувачів;

— коментування текстів або елементів контенту;

— структура даних і зручність навігації;

— захист персональних даних;

— функції спільної роботи;

— інтерфейс і легкість використання;

— можливість впровадження в освітнє середовище

Google Classroom [2] - одна з найпоширеніших сучасних платформ для організації освітнього процесу, створена компанією Google спеціально для шкіл, університетів, коледжів та онлайн-освіти. Її головна мета — зробити взаємодію між викладачами та студентами в цифровому середовищі максимально простою та зрозумілою. За допомогою цього інструменту можна легко створювати курси, роздавати завдання, перевіряти їх, виставляти оцінки, а також підтримувати

спілкування між усіма учасниками навчального процесу. Платформа дозволяє зручно організовувати навчальні матеріали, обговорювати завдання у коментарях, швидко інформувати студентів через оголошення та зберігати всі файли централізовано. Завдяки простоті, безкоштовності та інтеграції з іншими сервісами Google Classroom набув великої популярності у закладах освіти по всьому світу. Його легко опановують як студенти, так і викладачі, незалежно від рівня технічної підготовки. На рисунку 1.3 зображено, як виглядає інтерфейс цієї платформи.

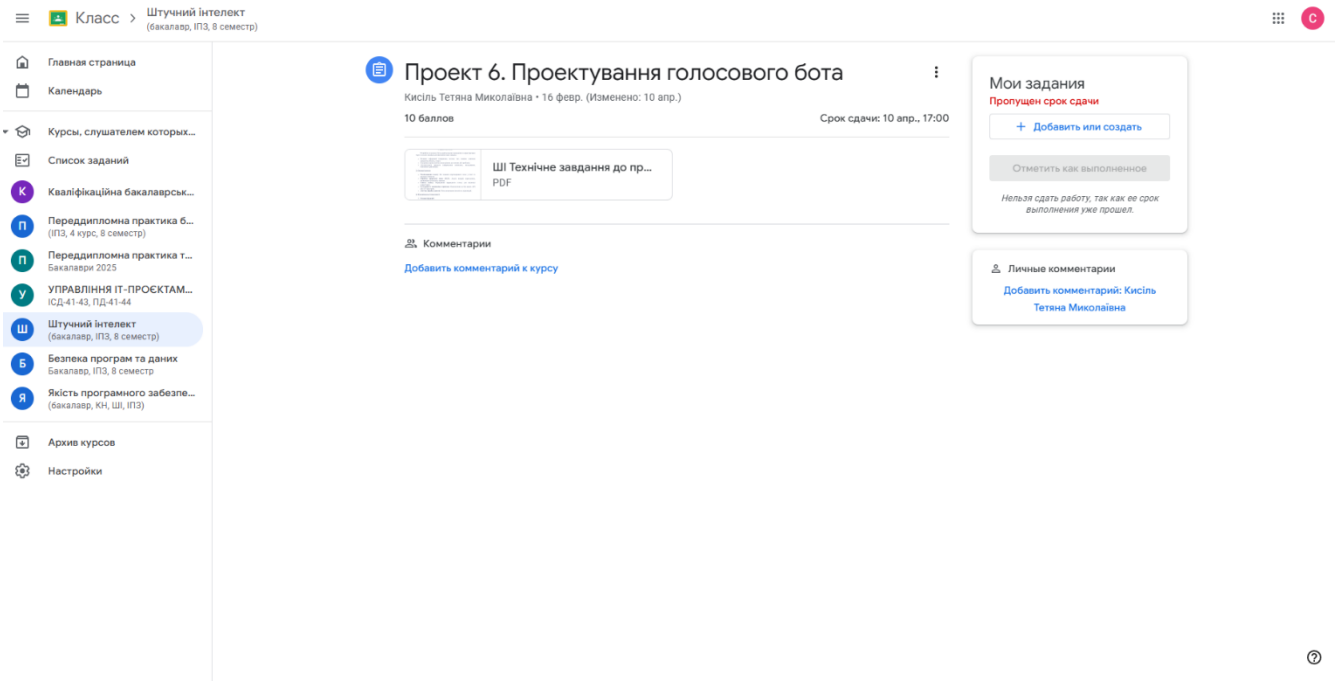


Рис. 1.3 Візуальний інтерфейс застосунку

Таким чином, при розгляді Google Classroom у контексті моніторингу наукової діяльності студентів, варто виокремити такі помітні особливості:

- Створення окремих курсів з матеріалами, завданнями та оголошеннями — усе чітко структуровано для кожної групи.

- Повна інтеграція з Google Drive, Docs, Gmail та Calendar, що забезпечує зручну роботу з файлами, автоматичне додавання подій у календар і спільне редагування документів.

- Інтерфейс знайомий усім користувачам Google, тож освоїтись у платформі не становить труднощів навіть для новачків.

- Можна встановлювати дедлайни, швидко виставляти оцінки, надсилати повідомлення як усій групі, так і окремим студентам.

- Платформа доступна як у веб-версії, так і через мобільні додатки, що дозволяє залишатися на зв’язку з навчанням у будь-який момент.

- Завдяки можливості групового обговорення й коментування завдань підтримується постійний діалог між усіма учасниками навчання.

- Є простий поділ ролей: викладачі та студенти, кожен із чітко визначеними правами доступу.

Переваги:

- Підключення до курсу здійснюється в кілька кліків через Google-акаунт — не потрібно проходити складні реєстрації.

- Усі файли зберігаються в хмарі Google Drive, що гарантує доступність матеріалів навіть при зміні пристрою.

- Завдяки синхронізації з Google Calendar студенти бачать дедлайни, а викладачі можуть планувати роботу на тиждень чи місяць наперед.

- Миттєві повідомлення про нові завдання чи зміни дозволяють оперативно реагувати на оновлення в курсі.

Недоліки:

- Відсутність гнучкої системи перевірки або рецензування: платформа не підтримує багатоступеневе затвердження проєктів або погодження кількома викладачами.

- Неможливість залишати коментарі безпосередньо в тексті роботи — лише загальні зауваження до всього завдання.

- Немає контролю версій і журналу змін — не можна відстежити, як змінювалась робота з часом або хто вніс правки.

- Система не дозволяє створювати індивідуальні маршрути перевірки або інтегруватися з іншими платформами для більш складної взаємодії.

Після проведеного аналізу можна зробити висновок, що Google Classroom чудово підходить для загальної організації навчального процесу та роботи з типовими завданнями. Але для супроводу саме наукової діяльності студентів — з

детальними перевірками, глибокими коментарями та контролем версій — функціоналу цієї платформи недостатньо. Саме тому все більше закладів звертаються до альтернативних рішень, наприклад, Notion або спеціалізованих застосунків, які краще адаптовані під індивідуальну наукову роботу студентів.

Notion [3] - це універсальна цифрова платформа нового покоління, яка поєднує функціонал нотатника, бази даних, таск-менеджера, календаря та навіть wiki-системи.

Однією з найбільших переваг платформи є можливість створювати власну систему — саме під свої потреби: від трекера особистих завдань до повноцінного робочого середовища для супроводу наукових проєктів. Notion дає змогу структурувати інформацію у вигляді баз даних, сторінок, списків літератури, таблиць зі статусами, тегами та коментарями.

Однак важливо розуміти, що Notion не створювався спеціально для освітнього середовища. У ньому відсутні такі речі, як ролевий поділ користувачів (керівник, рецензент, студент), формальні етапи затвердження робіт чи вбудовані механізми освітнього контролю.

На рисунку 1.4 представлений інтерфейс платформи Notion

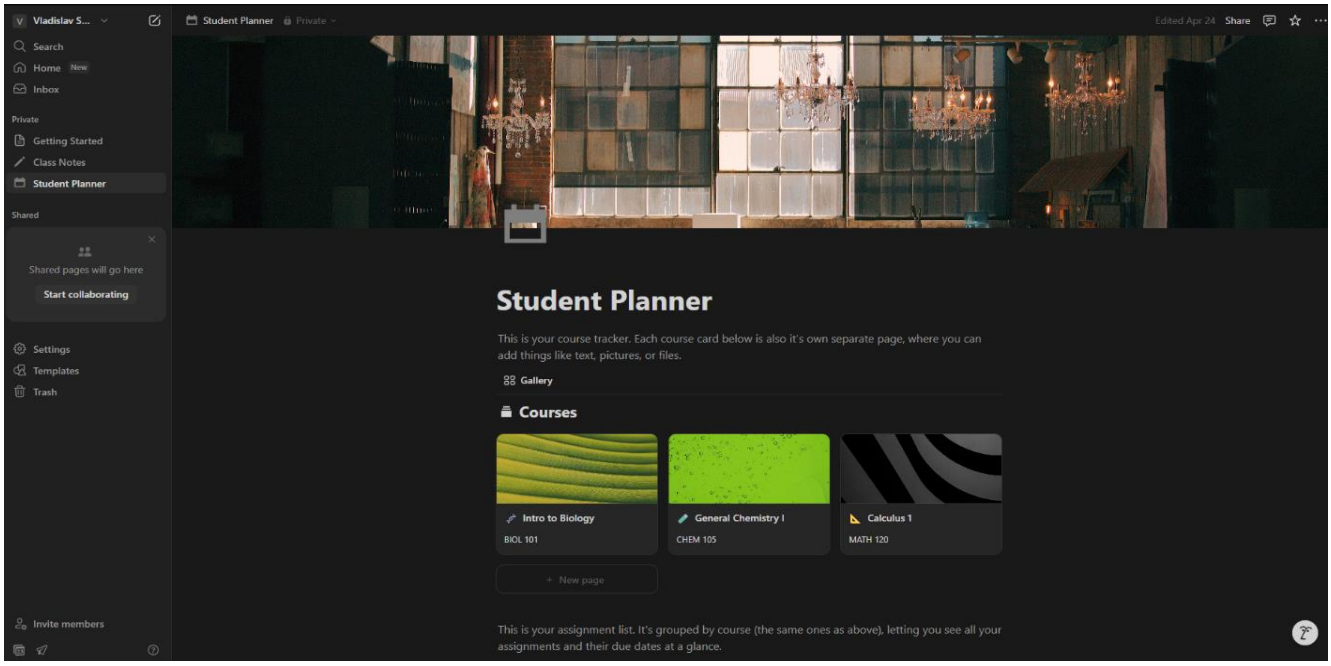


Рис. 1.4 Візуальний інтерфейс застосунку Notion

Опираючись на це, можна виокремити кілька помітних особливостей використання Notion у межах науково-освітнього процес:

- Повна свобода структурування — користувачі можуть самостійно створювати сторінки, таблиці, бази даних і зв’язувати їх будь-яким чином, що особливо корисно для складних і багаторівневих проєктів.

- Власна система баз даних із різними режимами перегляду: таблиця, дошка (канбан), календар, галерея — з підтримкою фільтрів, формул та автоматизацій.

- Широкий вибір готових шаблонів для планування досліджень, обліку завдань, наукового трекінгу, а також можливість створити свої шаблони з повторюваними подіями.

- Інтеграція з Google Drive, Slack, Zoom, GitHub та іншими сервісами, що дозволяє об’єднувати все в одному місці.

- Коментарі до окремих блоків сторінки — зручно вести дискусії прямо в контексті, не виходячи з робочого простору.

- Можливість командної роботи в реальному часі: одночасне редагування, коментування, фіксація прогресу та обговорення ідей.

- Кросплатформеність — доступ із браузера, мобільних додатків та десктопних клієнтів, включаючи розширення для Chrome.

Переваги:

- Надзвичайна гнучкість — Notion легко адаптується під будь-який формат роботи, дозволяє масштабувати проєкт або швидко змінювати його структуру.

- Величезна кількість шаблонів, інтеграцій і порад від спільноти допомагає швидко стартувати навіть новачкам.

- Простота освоєння інтерфейсу та можливість налаштувати все самостійно — без знання програмування.

- Спільна робота в реальному часі, збереження коментарів, призначення відповідальних і ручне ведення історії змін.

- Мобільність — доступ з будь-якого пристрою у будь-який момент.

Недоліки:

- Відсутність чіткої освітньої логіки: немає ролей типу «викладач», «студент», «рецензент», як і механізмів погодження етапів чи статусів.

- Немає автоматичного журналу змін або повноцінної історії версій — це ускладнює роботу над науковими текстами, особливо колективними.

- Необхідність налаштування «з нуля» — немає готових освітніх рішень, тому користувачам потрібно самостійно створювати всю структуру.

- Обмежена відповідність вимогам до захисту персональних даних, що може бути критичним у контексті офіційного навчального процесу.

- Зі збільшенням кількості сторінок та елементів складно підтримувати порядок і знаходити потрібну інформацію без додаткових правил організації.

Опираючись на наведений аналіз, можна зробити висновок, що Notion — надзвичайно гнучкий інструмент, який чудово підходить для самоорганізації, особистих досліджень або неформальної командної роботи. Але як платформа для офіційного супроводу студентських наукових робіт він має суттєві обмеження. Бракує освітньо орієнтованої логіки, ролей, автоматизації перевірок і безпечної інфраструктури, тому для формальної роботи в університетах частіше обирають інші інструменти — наприклад, Overleaf або вузькоспеціалізовані платформи з освітнім фокусом.

Overleaf [4] — це одна з провідних онлайн-платформ для створення та спільного редагування наукових текстів у форматі LaTeX. Вона активно використовується студентами, викладачами й дослідниками по всьому світу, особливо у сферах, де важливе чітке дотримання академічних стандартів оформлення — наприклад, при підготовці курсових, дипломних, статей або дисертацій. Основна ідея Overleaf — надати користувачам зручне, хмарне середовище для підготовки наукових документів із підтримкою спільної роботи. Платформа дозволяє редагувати код LaTeX прямо в браузері, одразу переглядати результат у PDF, коментувати окремі частини тексту, відстежувати історію змін і використовувати готові шаблони для наукових публікацій.

Саме завдяки цій увазі до точності, оформлення та командної взаємодії Overleaf став стандартом для технічних і природничих спеціальностей. Водночас, він не був створений як інструмент для освітнього супроводу, тому його функціонал має певні обмеження для університетського моніторингу студентських робіт. На рисунку 1.5 зображено інтерфейс платформи Overleaf

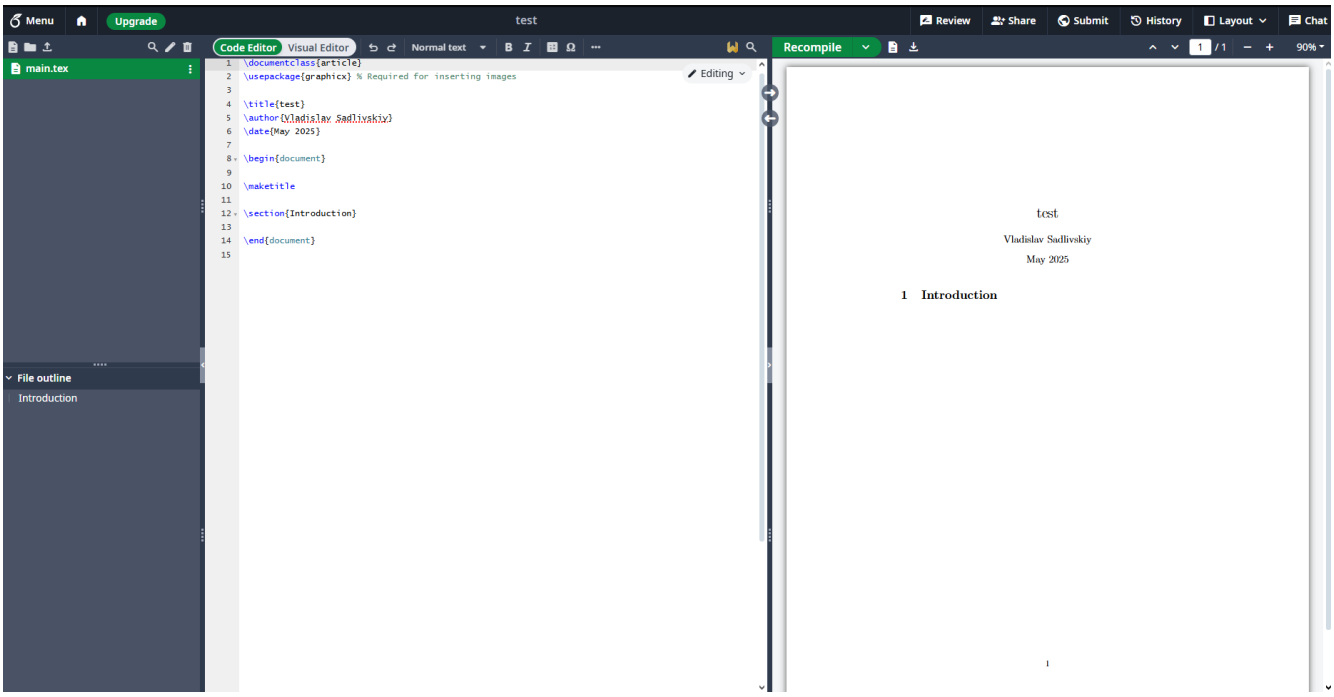


Рис. 1.5 Візуальний інтерфейс застосунку Overleaf

Враховуючи це, можна окреслити кілька помітних особливостей, що визначають сильні та слабкі сторони Overleaf у межах освітнього середовища:

- Орієнтований на якісне академічне оформлення, що відповідає міжнародним вимогам журналів, конференцій і університетів.

- Підтримує спільну роботу в реальному часі — кілька користувачів можуть одночасно працювати над проєктом, бачити зміни, коментувати й обговорювати правки.

- Має широку бібліотеку шаблонів — для статей, тез, звітів, дисертацій, з урахуванням різних форматів оформлення.

- Дозволяє обговорювати зміни за допомогою вбудованого чату та коментарів до окремих рядків тексту.

- Компіляція LaTeX-коду в PDF відбувається миттєво — це допомагає одразу побачити результат і знайти помилки у верстці.

Переваги:

- Дає змогу створювати професійні наукові документи з точним дотриманням оформлення, без потреби вручну налаштовувати макети.

- Спільне редагування та повна історія змін полегшують командну роботу та контроль за внесеними правками.

- Вбудовані інструменти коментування й обговорення дозволяють викладачу або рецензенту швидко залишати зворотний зв’язок у контексті.

- Готові шаблони значно скорочують час на оформлення, особливо при роботі з публікаціями, які мають специфічні вимоги.

Недоліки:

- Необхідність знання LaTeX — для новачків це може бути серйозною перешкодою, особливо якщо вони раніше не працювали з розміткою.

- Інтерфейс платформи іноді видається складним, особливо під час роботи з великими документами або шаблонами зі складною структурою.

- Відсутність навчально-педагогічної логіки — таких як погодження теми, погодження плану, поетапна перевірка — обмежує її використання в офіційних освітніх процесах.

- Немає ролевої моделі з функціями викладача, студента, адміністратора — лише співавторство або спільний доступ.

- У безкоштовному тарифі обмежено обсяг проєкту, кількість учасників, приватність документів і доступ до історії змін — це може бути критичним для великих або довготривалих наукових проєктів.

Після аналізу можна зробити висновок, що Overleaf — ідеальний інструмент для підготовки фінального наукового документа, особливо в галузях, де використання LaTeX є стандартом. Проте з точки зору освітнього моніторингу, поетапного контролю та формальної взаємодії між студентом і викладачем, платформа не має необхідного функціоналу. Через це навчальні заклади, які шукають зручні рішення для командної роботи, погодження завдань та візуалізації

ідей, все частіше звертаються до альтернатив — таких як Miro, які пропонують зовсім інший підхід до співпраці й організації навчального процесу

Miro [5] - це сучасна онлайн-платформа для спільної роботи на візуальних інтерактивних дошках, яка здобула широку популярність серед команд, що працюють над проєктами у сфері менеджменту, дизайну, стратегічного планування та креативного мислення. Основна сила Miro — у її гнучкому, візуально орієнтованому інтерфейсі, який дозволяє створювати діаграми, інтелект-карти, блок-схеми, робочі таблиці, логічні зв'язки тощо.

У сфері освіти Miro часто використовується як допоміжний інструмент: для планування курсових і дипломних проєктів, командного мозкового штурму, візуального структурування досліджень чи презентації проміжних результатів. Проте, незважаючи на широкі можливості у візуалізації, платформа не має вбудованих механізмів для роботи з текстами, їх перевірки чи освітнього моніторингу в класичному розумінні. Тому її функціонал найбільш ефективний саме на початкових етапах наукового проєкту — коли потрібно «розкласти все по поличках» і спільно спланувати подальші кроки.

На рисунку 1.6 зображений візуальний інтерфейс застосунку.

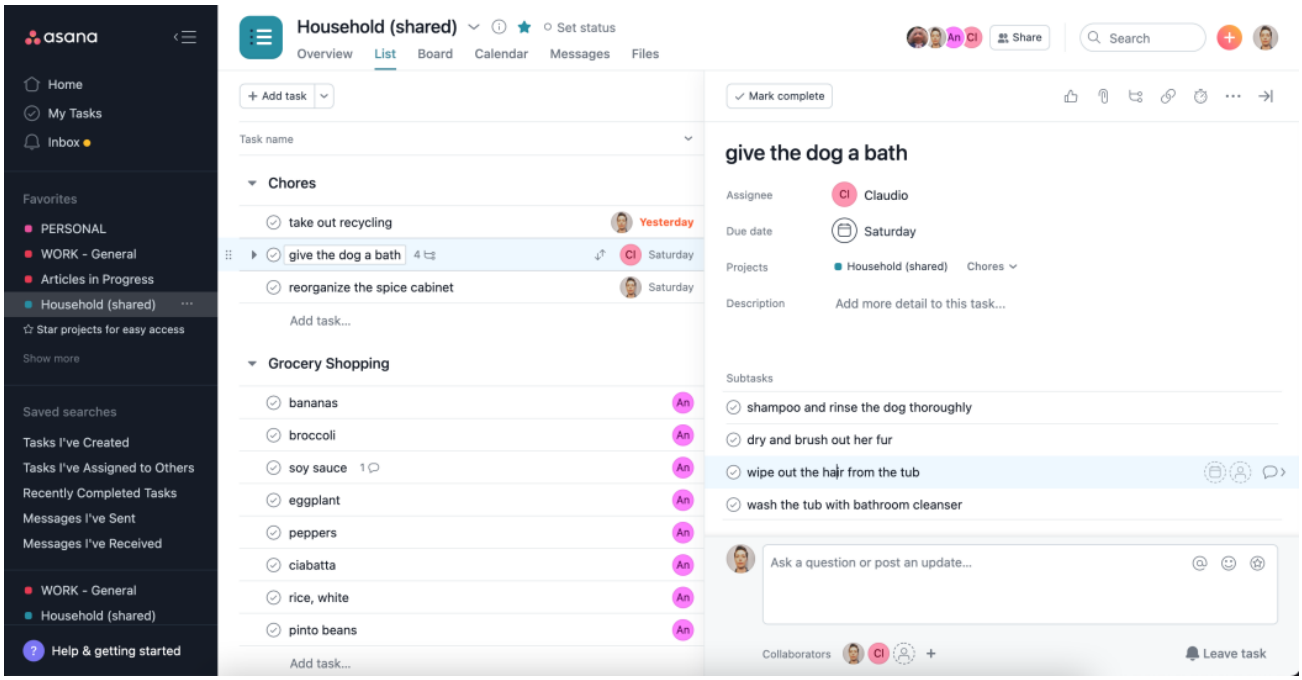


Рис. 1.6 Візуальний інтерфейс застосунку Miro

З урахуванням наведеного, можна виокремити такі помітні особливості Miro, які є характерними у контексті освітніх завдань:

- Візуальне середовище, у якому можна створювати схеми, ментальні карти, діаграми, організаційні структури — усе в режимі реального часу.

- Колективна взаємодія — кілька користувачів можуть одночасно працювати над однією дошкою, залишати коментарі, рухати об’єкти та доповнювати карту ідей.

- Бібліотека шаблонів для різних сценаріїв: від планування проєкту до аналізу проблем, ретроспектив і стратегічного мислення.

- Простота старту — не потребує додаткових програм чи складних налаштувань. Платформа працює у браузері, і до неї легко підключитися з будь-якого пристрою.

- Зручність для візуального структурування складних проєктів — від етапів дослідження до моделі взаємодії учасників.

Переваги:

- Інтуїтивно зрозумілий інтерфейс, який дозволяє швидко створювати й редагувати візуальні елементи навіть без технічної підготовки.

- Можливість синхронної роботи великої кількості користувачів — з наочним відображенням змін у режимі реального часу.

- Великий вибір готових шаблонів та інструментів для командного планування, брейнштормінгу та організації ідей.

- Повна кросплатформеність — доступ із браузера, мобільного додатку або планшета.

- Надзвичайно зручна для початкового етапу дослідження — коли потрібно визначити мету, сформулювати ключові запитання, структуру чи логіку виконання.

Недоліки:

- Відсутність функціоналу для роботи з текстовими документами — платформа не призначена для написання або перевірки змісту наукових робіт.

- Немає системи статусів, затвердження чи рецензування, тому повноцінний моніторинг виконання студентських завдань неможливий.

- Відсутній поділ ролей (керівник, студент, рецензент) та детальне управління правами доступу.
- В обмеженій (безкоштовній) версії зберігається неповна історія змін, а також відсутні функції резервного копіювання проєктів у повному обсязі.
- Не є самостійною платформою для освітнього документообігу чи зберігання фінальних версій робіт — лише додатковий візуальний інструмент

Отже, Miro чудово підходить як допоміжний інструмент на ранніх етапах підготовки дослідницької роботи. Він дозволяє ефективно структурувати ідеї, організувати групову взаємодію, спільно створити «карту проєкту» та обговорити основні завдання. Але для подальшого супроводу, перевірки, фіксації змін і формального моніторингу виконання студентських наукових робіт ця платформа не має достатньої функціональності. Тому в освітньому контексті Miro найчастіше використовується як візуальне доповнення до основних систем управління навчанням або спеціалізованих веб-застосунків.

1.3 Порівняння аналогів

В процесі аналізу популярних інструментів для організації та супроводу студентських наукових робіт були відібрані ключові характеристики, які найкраще відображають їхню придатність для освітніх цілей. Зведені результати порівняння подано у таблиці 1.1 — вона допомагає краще зрозуміти, у чому сильні й слабкі сторони кожного з розглянутих рішень. Зокрема, увагу було зосереджено на таких аспектах, як зручність користування, можливість налаштування, підтримка спільної роботи, захист даних, контроль статусів і версій, а також відповідність потребам навчального процесу.

Таблиця 1.1

Зведені результати аналізу характеристик додатків для моніторингу наукових робіт студентів

Критерій	Google Classroom	Notion	Overleaf	Miro	RAT
Підтримка ролей	+, викладач і студент	–, ролі не передбачено	+, базова рольова модель (автор, співавтор)	–, усі мають рівні права	+, чітке розмежування: студент / викладач
Коментування фрагментів робіт	–, лише загальні коментарі	+,можливість залишати коментарі до блоків	+, коментарі до частин тексту у чатах	–, коментарі лише до об'єктів, не до текстів	+, коментарі до окремих частин документа із прив'язкою до статусу
Структурування даних	+, базова прив'язка тем і дедлайнів	+, повна кастомізація баз, таблиць і сторінок	+, чітка структура LaTeX-документа	–, орієнтація на схеми, не на дані	+,структурована система етапів, тем, керівників,
Контроль версій / історія змін	–, немає	+, частковий контроль через зміни сторінок	+, професійна система версій	–, історія змін лише загальна	+, автоматичне збереження змін і журнал активності
Етапність виконання робіт	–, не передбачено	–, необхідно створювати вручну	–, підтримка лише готового тексту	–, не підтримується	+, реалізовано через статуси
Публікації / підтвердження етапів	–, немає	–, не передбачено	–, публікація лише вручну	–, не передбачено	+, затвердження через керівника
Інформаційна безпека / доступ	+, доступ через акаунт Google	–, доступ залежить від загальних налаштувань	+,приватні проекти з контролем доступу	–, відсутній контроль прав	+,багаторівнева система доступу і захист персональн
Простота використання	+, інтерфейс інтуїтивно зрозумілий	+,зрозумілий після адаптації	–, вимагає знань LaTeX	+, легкий візуальний	+, адаптований для різного рівня цифрових навичок користувачів

Загальні результати порівняння показують, що жоден із доступних сервісів не охоплює повністю весь спектр вимог до системи моніторингу студентської наукової діяльності. Одні платформи добре справляються з колаборацією чи

форматуванням текстів, інші — з адаптацією під індивідуальні потреби. Проте кожен із них має свої обмеження: хтось не підтримує повноцінну рольову модель, десь бракує контролю версій або гнучкого коментування, а деякі рішення просто не розраховані на освітній контекст. У цьому сенсі розроблений у межах цієї роботи веб-застосунок виступає як спроба об'єднати найкращі риси існуючих платформ і доповнити їх тим, чого не вистачає. Він пропонує зручний і зрозумілий інтерфейс, підтримку ролей, фіксацію статусів, контроль змін і безпечну взаємодію — тобто саме ті функції, які важливі для ефективного освітнього процесу сьогодні.

У процесі аналізу предметної галузі та вивчення сучасних цифрових рішень було сформовано цілісне уявлення про особливості організації моніторингу наукової діяльності студентів у закладах вищої освіти. Поглиблене дослідження дало змогу виявити низку характерних викликів, з якими щодня стикаються як студенти, так і викладачі під час підготовки, супроводу та перевірки індивідуальних наукових проєктів. До таких проблем, зокрема, належать складність підтримки ефективного зворотного зв’язку, відсутність зручних інструментів для контролю етапності виконання роботи, складність прозорого оцінювання результатів та обмеженість можливостей дотримання встановлених дедлайнів.

Окремий блок дослідження був присвячений аналізу функціональності найпоширеніших сучасних платформ — зокрема, Google Classroom, Notion, Overleaf та Miro. Аналіз продемонстрував, що попри широке поширення та технічну зрілість, жодна з цих платформ не є спеціалізованим рішенням для комплексного моніторингу студентських наукових робіт. Більшість із них орієнтовані або на загальну освітню взаємодію, або на створення персональних чи командних інформаційних просторів, але не враховують специфіку академічного оцінювання, індивідуальної відповідальності студента й багаторівневого зворотного зв’язку.

Окремо було виокремлено проблему відсутності чіткої ролевої моделі, яка б передбачала різні рівні доступу й відповідальності, можливості етапного погодження проєктів, фіксації історії змін, а також запровадження ефективного

механізму контекстного коментування — як критично важливого елементу продуктивної взаємодії між викладачем і студентом. Для систематизації результатів було складено таблицю порівняння функціональних можливостей різних платформ, що дозволило виділити їхні сильні та слабкі сторони, а також визначити перелік проблем, які залишаються поза увагою наявних інструментів.

У підсумку, проведений аналіз не лише поглибив розуміння предметної області, а й сформував методологічну основу для подальшої розробки власного веб-застосунку. Виявлені обмеження сучасних рішень чітко окреслили вектор розвитку: необхідність створення гнучкої, рольової, інтерактивної та прозорої системи, орієнтованої саме на підтримку наукової діяльності студентів. Саме ці висновки стали відправною точкою для формування технічного завдання, побудови функціональних і нефункціональних вимог, а також вибору архітектурних і технологічних рішень, що лягли в основу майбутньої платформи.

2 ПРОЄКТУВАННЯ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Визначення користувачів

Інформаційна система для моніторингу наукової діяльності студентів створювалася з урахуванням потреб реальних користувачів. Її логіка побудована таким чином, щоб кожен учасник освітнього процесу — незалежно від свого досвіду чи ролі — міг працювати з платформою зручно, швидко і результативно. У центрі уваги — люди, які щодня взаємодіють із науковими роботами: студенти, які ці роботи створюють, і викладачі, які супроводжують, оцінюють та допомагають їх вдосконалити.

Основні групи користувачів

Система передбачає роботу двох ключових категорій користувачів — студентів та викладачів (або наукових керівників). Кожна з цих груп має власні сценарії використання, які відображають її реальні функції та задачі в навчальному процесі.

Студенти — це ті, для кого створюється більшість функціоналу системи. Вони завантажують власні роботи (курсові, бакалаврські, дипломні), переглядають статус перевірки, отримують коментарі, вносять правки і спілкуються зі своїм науковим керівником. Для них надзвичайно важливо, щоб інтерфейс був зрозумілий з першого погляду, дедлайни — чітко відображалися, а історія змін дозволяла відстежити кожен крок у підготовці. Окремо варто відзначити зручність доступу до зворотного зв’язку: студенти очікують побачити не просто загальні коментарі, а конкретні зауваження до певних частин тексту, щоби точково вдосконалювати свою роботу.

Викладачі — це ті, хто здійснює супровід, контроль і рецензування наукових робіт. Вони бачать список студентів і пов’язані з ними проєкти, можуть змінювати статус кожної роботи (наприклад, “на перевірці”, “потребує доопрацювання”, “схвалено”), залишати коментарі, фільтрувати роботи за групами або етапами. Для

них особливо цінним є можливість швидко реагувати на нові завантаження, бачити зміни у роботах та мати зручну систему повідомлень, яка дозволяє не пропустити важливі події. Крім цього, викладачам потрібні базові інструменти аналітики — наприклад, для оцінки прогресу або обсягу редагувань.

Очікування і потреби користувачів

Зрозуміти, чого саме очікують користувачі від системи, — ключ до її ефективності. І хоча ролі студентів і викладачів суттєво різняться, для обох важливими є швидкість, прозорість та зручність.

Студенти хочуть мати чітку структуру особистого кабінету, де можна легко завантажити нову версію роботи, бачити її статус у режимі реального часу та своєчасно отримувати сповіщення про нові коментарі чи дедлайни. Також важливо, щоб система дозволяла зберігати історію змін і комунікації з викладачем, аби уникнути повторного редагування або непорозумінь. Інтуїтивність інтерфейсу, підтримка мобільних пристроїв і швидкий зворотний зв’язок — це ті параметри, що суттєво впливають на досвід користування.

Викладачі, зі свого боку, очікують оперативного доступу до всіх робіт підопічних студентів, з можливістю швидко фільтрувати й знаходити потрібне, бачити зміни по кожному документу, залишати коментарі до окремих фрагментів тексту, а також мати інструменти для формального виставлення оцінок. Їм важливо мати гнучку систему сповіщень, яка не буде перевантаженою, але нагадає про нову подану роботу чи відповідь студента.

Взаємодія користувачів із системою

Щоб візуалізувати цю взаємодію, доцільно використати схему, де показано, як саме інформація передається між студентом і викладачем через систему. Така структурна модель дозволяє краще зрозуміти логіку платформи навіть тим, хто не знайомий із її технічною реалізацією.

На рисунку 2.1 зображено структуру користувачів інформаційної системи у моєму застосунку

Символ	Опис
actor "Студент"	Користувач, який подає і редагує роботи, отримує фідбек, спілкується з викладачем
actor "Викладач"	Користувач, який супроводжує, перевіряє, оцінює, затверджує студентські роботи
rectangle	Центральна система моніторингу, через яку відбувається уся взаємодія

Рис. 2.1 Структура користувачів у застосунку

2.2 Ролі

У сучасних ІТ-рішеннях для освіти модель розподілу ролей є не просто технічним механізмом обмеження доступу — вона відіграє важливу роль у формуванні ефективної, зручної та безпечної взаємодії між усіма учасниками навчального процесу [33]. Саме завдяки чіткому визначенню ролей платформа набуває керованості, а користувачі отримують саме ті функції, які відповідають їх завданням, повноваженням і відповідальності. У межах розробленої системи було обрано логічну та водночас гнучку модель ролей [34]. Наразі реалізовано два основні типи користувачів: студент і викладач (науковий керівник). Такий поділ дозволяє не перевантажувати інтерфейс зайвими функціями, при цьому залишаючи достатньо простору для адаптації платформи під майбутні потреби — наприклад, додавання ролей рецензентів або адміністративного персоналу.

Загальний підхід та переваги

Коли користувач бачить у системі лише те, що йому дійсно потрібно — це знижує ризик помилок, підвищує продуктивність і формує позитивний досвід взаємодії. Натомість надмірні або невизначені повноваження, як показує практика,

призводять до плутанини, конфліктів доступу та зниження ефективності командної роботи.

У моїй системі реалізовано ролеву модель, яка чітко визначає: хто що може робити, бачити, редагувати чи змінювати. Такий підхід допомагає уникати перетину функцій, конфлікту інтересів, несанкціонованих дій. Крім того, розмежування прав доступу дає змогу централізовано впроваджувати політику доброчесності, фіксувати всі дії користувачів та швидко відновлювати події у випадку виникнення спірних ситуацій.

Подібні моделі вже давно стали стандартом у найкращих освітніх платформах світу, зокрема у внутрішніх системах західних університетів, і показали свою ефективність як у великих групах, так і при індивідуальній роботі зі студентом.

Роль «Студент»

Роль студента — це центральна роль у будь-якому освітньому рішенні, адже саме студент є безпосереднім автором, виконавцем і відповідальним за свою роботу. У нашій системі студент має простий, зручний і максимально інтуїтивний доступ до свого персонального кабінету, де він може завантажувати наукову роботу, редагувати її, стежити за статусами, отримувати коментарі від викладача та відповідати на них.

Усе побудовано так, щоб студент не губився в інтерфейсі, бачив лише ті функції, які йому справді потрібні — без ризику втручання в чужі дані чи важливі налаштування системи [. Він отримує сповіщення про будь-які дії викладача щодо своєї роботи, бачить усі зміни поетапно і має змогу переглядати історію обговорень. Завдяки логуванню дій студент захищений від випадкових видалень чи втрати даних.

Можливості студента:

- створення, редагування та надсилання власних робіт;
- перегляд поточного статусу та історії змін;
- отримання коментарів та відповіді на них;
- аналітика власного прогресу.

Обмеження студента:

- відсутність доступу до чужих робіт чи функцій викладача;

- неможливість змінювати статуси роботи або видаляти коментарі наукового керівника.

Роль «Викладач» (науковий керівник)

Викладач у системі виконує ключову функцію підтримки, супроводу, оцінювання і контролю. Його інтерфейс має ширший функціонал, який включає перегляд усіх робіт підопічних студентів, перегляд останніх змін, коментування окремих фрагментів тексту, формування зворотного зв’язку й зміну статусу проєкту (наприклад, «на доопрацуванні» або «схвалено»).

Кожен коментар, залишений викладачем, чітко фіксується та прикріплюється до конкретного місця в тексті, що спрощує подальше обговорення. Також викладач має змогу бачити динаміку роботи кожного студента та при потребі формувати звіти або оцінки. Панель повідомлень дозволяє миттєво реагувати на нові дії студентів, навіть коли робіт багато або вони надходять у різний час.

Можливості викладача:

- перегляд робіт своїх студентів;
- коментування тексту та виставлення статусів;
- рецензування робіт.
- оцінювання та перегляд змін робіт.
- написання висновків;.

Обмеження викладача:

- неможливість змінювати текст роботи студента;
- відсутність доступу до робіт інших викладачів чи конфіденційної інформації;
- можливість взаємодії лише з підопічними студентами.

На рисунку 2.2 зображено взаємодію користувачів у системі моніторингу наукової діяльності студентів

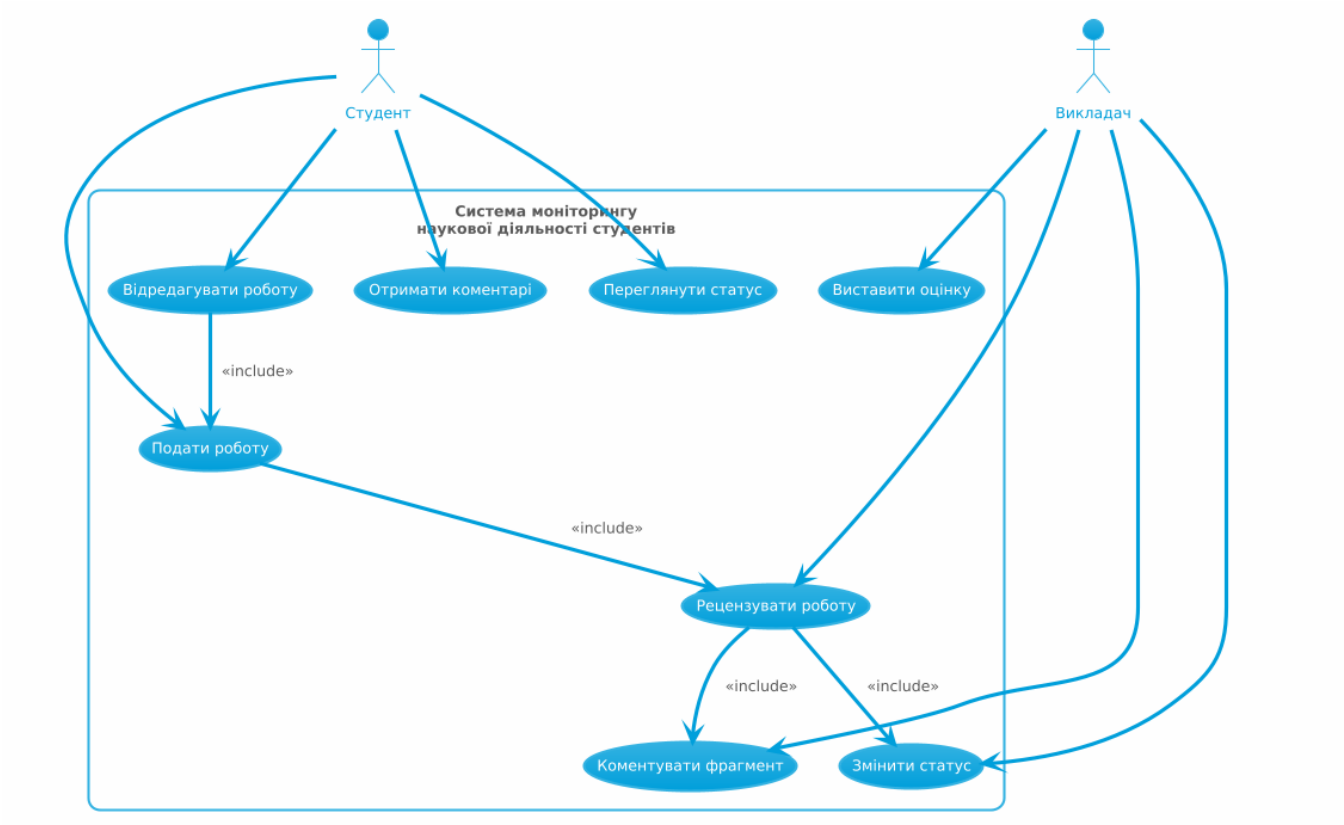


Рис. 2.2 Взаємодія користувачів у системі

Перспективи розвитку моделі ролей

Передбачено можливість розширення системи ролями рецензентів, завідувачів кафедр, зовнішніх експертів або навіть гостьових користувачів. Це дозволяє адаптувати платформу під складні багаторівневі освітні сценарії, де, наприклад, рецензент бачить роботу без імені автора, а завідувач може моніторити загальну динаміку процесу по кафедрі.

Це дозволить розширити функціональність системи під більш складні сценарії — наприклад, для захисту бакалаврських робіт, акредитації спеціальност ей або зовнішнього аудиту освітнього процесу

2.3 Функціональні вимоги

Функціональні вимоги — це основа, на якій тримається вся система. Вони визначають, якими саме інструментами повинна володіти платформа, аби повноцінно виконувати свої завдання. Саме ці вимоги окреслюють, як повинна

виглядати взаємодія з користувачем, що він може робити в системі, як швидко й ефективно буде досягнуто результат [32]. Це не просто технічна частина — це відображення реальних очікувань від системи, зібраних із реальних освітніх ситуацій [18].

Основна мета — зробити моніторинг і супровід наукових робіт не обтяжливим технічним процесом, а зручним, прозорим і зрозумілим робочим інструментом для обох сторін: студентів і викладачів. Саме тому система розроблялася з акцентом на простоту у користуванні, швидкість взаємодії, наочність усіх змін та дотримання безпеки.

Робота зі студентськими роботами

- Одним із базових елементів системи є можливість створення, редагування та збереження наукової роботи у власному середовищі. Студент може працювати над текстом поступово — це особливо цінно, коли робота велика і вимагає багато етапів перевірки. Збереження чернеток і автоматичне збереження змін формує відчуття впевненості: жоден абзац не загубиться через збій чи випадкове закриття вікна.

- Можна завантажувати нові версії роботи в будь-який момент, і викладач одразу бачить, як змінювався текст з часом. Уся історія зберігається — ніби хронологія розвитку думки. Це корисно не лише викладачу, а й самому студенту, який може повернутись до старої версії, порівняти зміни або пояснити свою логіку редагування.

- Статуси робіт — «чернетка», «на перевірці», «потребує доопрацювання», «затверджено» — допомагають чітко бачити, на якому етапі знаходиться завдання. Студенту не потрібно писати додаткових листів викладачу, щоби дізнатись: усе вже вказано у його кабінеті.

Комунікація між студентом і викладачем

- Один з найважливіших елементів системи — це зручна, логічно організована комунікація. Коментарі можна залишати прямо в тексті — до кожного окремого фрагмента. Завдяки цьому фідбек не розпливається у загальних формулюваннях, а стає конкретним і максимально прикладним.

- Уся історія переписки зберігається в одному місці: немає потреби повертатися до старих листів або шукати повідомлення у месенджерах. Весь діалог, рекомендації й відповіді завжди під рукою — навіть якщо між ними минуло кілька тижнів.

- Уточнення, обговорення правок, пояснення незрозумілих моментів — усе це відбувається безпосередньо в системі. Це значно скорочує час на зворотний зв'язок і дозволяє обом сторонам краще сфокусуватись на змісті, а не на технічних деталях.

- Система автоматично сповіщає користувача про кожен новий коментар чи зміну статусу — це означає, що важливі оновлення не залишаться непоміченими.

Рольові можливості та безпека

- Одне з основних завдань — зробити систему безпечною й логічною з точки зору прав доступу. Студенти бачать лише свої роботи, а викладачі — лише роботи тих студентів, з якими вони працюють. Такий підхід захищає приватні дані, запобігає випадковому (або навмисному) перегляду чужих документів та підтримує принцип академічної доброчесності.

- Кожна дія користувача (зміна тексту, коментар, оновлення статусу) автоматично фіксується у системному журналі. Це дозволяє відстежити повну історію взаємодії та, за потреби, виявити джерело непорозуміння або технічного збою.

- Авторизація проходить у кілька рівнів, паролі зберігаються у зашифрованому вигляді, сесії захищені від несанкціонованого доступу — усе відповідає актуальним стандартам безпеки для освітніх сервісів.

Аналітика та звітність

Для ще більшої ефективності система пропонує гнучкі інструменти аналітики. Студенти можуть бачити свій прогрес у історій змін, а викладачі — отримувати загальну картину успішності по групах. Це дозволяє об'єктивно оцінити, хто з підопічних встигає, а хто має труднощі, коли саме робота зупинилася або які коментарі мали найбільший вплив на результат.

Звіти формуються автоматично: можна швидко подивитися динаміку групи, частоту коментарів, кількість повернень на доопрацювання. Це значно спрощує і внутрішню аналітику, і звітність перед адміністрацією.

Користувацький інтерфейс

- Інтерфейс системи створено з розумінням, що не всі користувачі мають однаковий досвід роботи з ІТ. Тому головні функції — на видному місці, з підказками та простими назвами. Користувач може інтуїтивно знайти те, що шукає, навіть без інструкцій.

- Адаптивність — ще одна перевага. Користуватись платформою можна з будь-якого пристрою: вдома за ноутбуком, на парі з планшета чи дорогою з телефону. Дизайн автоматично підлаштовується під екран.

- Швидке перемикання між розділами, перегляд усіх статусів і коментарів у кілька кліків — усе це створює враження “живої” системи, яка не заважає, а допомагає.

На таблиці 2.1 зображено функціональні вимоги до мого застосунку

Таблиця 2.1

Функціональні вимоги

№	Вимога	Користувач	Опис/сценарій використання
1	Завантажити наукову роботу	Студент	Додає файл, зберігає чернетку, надсилає на перевірку
2	Переглядати статус роботи	Студент	Відображення поточного статусу роботи
3	Коментувати роботу	Обидва	Додавання та перегляд коментарів до фрагментів

Продовження Таблиці 2.1

Функціональні вимоги

№	Вимога	Користувач	Опис/сценарій використання
4	Змінювати статус роботи	Викладач	Переведення роботи у статус "доопрацювання"
5	Оцінити роботу	Викладач	Виставлення підсумкової оцінки
8	Переглядати історію змін	Обидва	Журнал усіх подій по роботі
9	Побудова аналітичних звітів	Викладач	Діаграми прогресу студентів, кількість робіт

А ось на рисунку 2.3 зображена діаграма “активності” у моєму застосунку

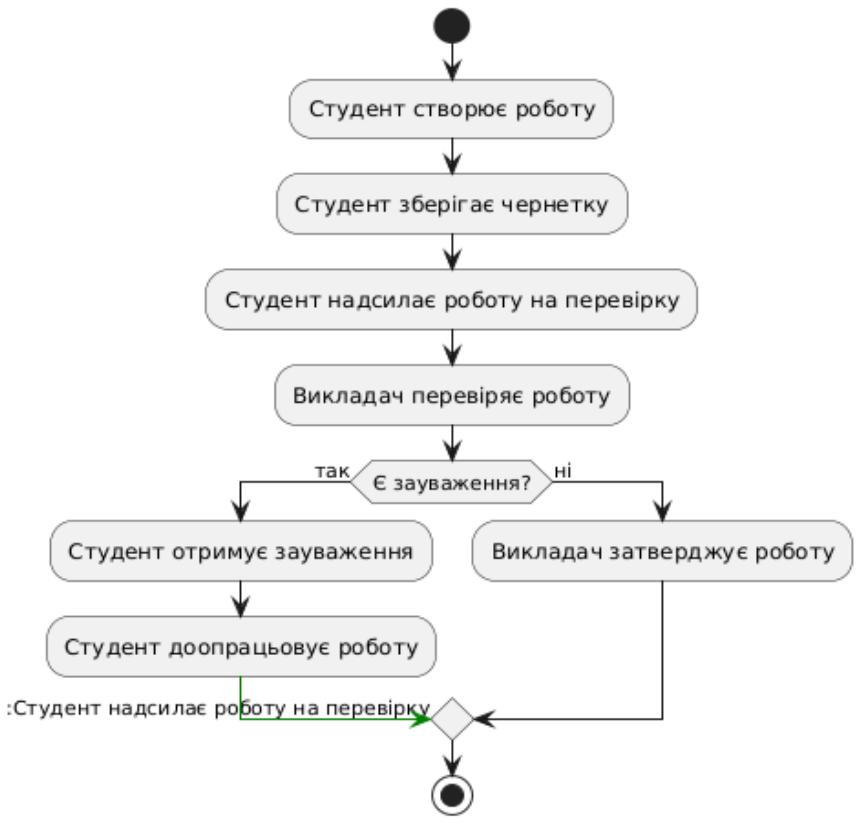


Рис. 2.3 Діаграма активності

Завдяки реалізації всіх перелічених функцій система не лише відповідає потребам студентів і викладачів, а й створює зручне, зрозуміле та гнучке середовище, яке легко адаптується до нових вимог. Саме така основа забезпечує ефективну взаємодію, прозорість процесів і підтримку академічної доброчесності в сучасному освітньому просторі.

2.4 Нефункціональні вимоги

У розробці сучасних освітніх платформ важливо не тільки забезпечити необхідний функціонал, а й створити таку систему, яка буде швидкою, стабільною, безпечною та зручною у щоденному використанні [30]. Саме нефункціональні вимоги відповідають за ті характеристики, які визначають загальний рівень якості програмного продукту: чи комфортно з ним працювати, наскільки він надійний у різних ситуаціях, чи готовий до зростання кількості користувачів або оновлення функцій у майбутньому [31]. Вони створюють базу довіри до системи та дозволяють зберігати її ефективність на всіх етапах життєвого циклу.

Продуктивність і швидкодія

З самого початку система проєктувалася з урахуванням високої продуктивності: будь-яка дія користувача, така як завантаження або редагування роботи, перегляд коментарів чи зміна статусу, має виконуватися миттєво — незалежно від того, скільки людей працюють у системі одночасно.

Цього вдається досягти завдяки використанню індексів у базі даних, кешування запитів, оптимізації серверних обчислень і розумному розподілу навантаження [32]. Таким чином, платформа «не гальмує», не змушує чекати та не створює ситуацій, коли користувач втрачає час на очікування відповіді.

Надійність і відмовостійкість

Користувач повинен бути впевнений, що його дані не зникнуть у найбільш критичний момент. Саме тому в системі реалізовано багаторівневу систему резервного копіювання, яка дозволяє відновити всю інформацію навіть у разі серйозного збою апаратного чи програмного забезпечення [29]. Усі зміни, дії й

події фіксуються в логах, що забезпечує повну прозорість та дає змогу розробникам або адміністраторам швидко реагувати на інциденти. Якщо виникає помилка, користувач отримує інформативне повідомлення, а відновлення функціональності відбувається автоматично — без втручання чи незручностей для студентів і викладачів.

Масштабованість

Платформа має бути здатною зростати разом із потребами навчального закладу. Для цього її архітектура побудована так, щоб нові функції, ролі, типи документів або підсистеми можна було додавати без порушення загальної логіки чи зупинки сервісу.

Система може працювати як у локальному середовищі, так і в хмарі, підтримує розгортання на кількох серверах одночасно та легко адаптується до зростання кількості користувачів чи робочого навантаження. Це дозволяє університету не обмежувати себе у майбутніх змінах або розширеннях функціоналу.

Безпека

Ураховуючи роботу з конфіденційними освітніми матеріалами та персональними даними, безпека в системі — на першому місці [28]. Усі паролі зберігаються у зашифрованому вигляді, дані передаються виключно через захищене з’єднання HTTPS, а доступ до дій суворо обмежується відповідно до ролі користувача. Система також передбачає захист від найбільш поширених кіберзагроз: SQL-ін’єкцій, міжсайтового скриптингу (XSS), підробки запитів (CSRF). Усі введені дані проходять перевірку на безпечність, а спроби несанкціонованого доступу одразу блокуються.

Доступність і зручність використання

Інтерфейс платформи продумано до дрібниць: у ньому легко орієнтуватися навіть тим користувачам, які не мають досвіду користування освітніми або адміністративними системами. Кожна дія супроводжується підказками, а структура меню інтуїтивно зрозуміла.

Особливу увагу приділено доступності — інтерфейс адаптований до потреб людей із порушеннями зору та моторики [27]. Великі шрифти, висока контрастність, підтримка навігації з клавіатури, можливість підключення екранного диктора — усе це робить платформу інклюзивною.

Завдяки адаптивному дизайну система працює однаково комфортно як на комп’ютері, так і на смартфоні, тож студенти можуть виконувати завдання, перебуваючи будь-де.

Супровід і оновлення

Для підтримки стабільності платформи в майбутньому передбачено гнучкий механізм оновлень. Нові функції впроваджуються поетапно, із попереднім тестуванням у спеціальному середовищі. У разі виникнення проблем розробники можуть “відкотити” систему до попередньої версії без втрати даних. Кожен модуль задокументований, має автоматичні тести, що значно полегшує підтримку й оновлення системи без зупинки основних процесів. Такий підхід гарантує, що система зможе змінюватися відповідно до нових вимог часу без болючих перебудов.

Інтеграція з іншими сервісами

Платформа має відкритий API, що дозволяє легко поєднувати її з іншими системами: електронними журналами, системами управління навчанням (типу Moodle, Google Classroom), сервісами пошти, календарями чи зовнішніми базами даних. Можливість експорту й імпорту даних дає змогу просто формувати звіти та ділитися інформацією.

Таким чином, нефункціональні вимоги — це те, що робить систему не просто робочою, а зручною, надійною, безпечною та здатною адаптуватися до змін. Завдяки цим характеристикам платформа залишається актуальною навіть у динамічному освітньому середовищі, де очікування користувачів постійно зростають.

2.5 Діаграма варіантів використання

Діаграма варіантів використання (Use Case Diagram) — це не просто формальна складова технічної документації, а насправді важливий етап, який дозволяє «поглянути на систему очима користувача» [26]. Саме завдяки таким схемам ще до написання жодного рядка коду можна зрозуміти, як працюватиме майбутній веб-застосунок, хто і яким чином з ним взаємодіятиме, які саме дії виконуватимуть різні учасники процесу — студенти, викладачі, адміністратори тощо.

У випадку розробки системи для моніторингу студентських наукових робіт, діаграма варіантів використання є ключовим інструментом для формалізації й візуалізації всіх типових сценаріїв. Вона допомагає чітко структурувати процес: від моменту, коли студент подає свою першу чернетку, до фінального затвердження викладачем і перегляду статистики. Таке представлення дозволяє краще оцінити, які саме модулі повинні бути реалізовані, які зв’язки існують між ними, і що буде потрібно кожному користувачу на кожному етапі роботи. Особливість подібної діаграми полягає в її універсальності: вона зрозуміла не лише розробникам або аналітикам, а й потенційним користувачам, адміністрації ЗВО, тестувальникам чи навіть керівництву факультету. Вона буквально «промовляє» мовою логіки платформи й одразу показує, чи дійсно враховано всі необхідні функції, чи немає прогалин, що можуть викликати труднощі в реальному користуванні.

Фактично, це своєрідна “дорожня карта” усіх сценаріїв, яка значно полегшує подальшу розробку, тестування, документування та супровід програмного забезпечення. Такий підхід дозволяє не лише раціонально планувати архітектуру, а й заздалегідь закласти основу для масштабування платформи в майбутньому.

З практичної точки зору, діаграма варіантів використання також дозволяє уникнути дублювання функціоналу, зайвого навантаження на інтерфейс і логіку системи [25]. Вона показує, які дії є унікальними для кожної ролі, де можливе перетинання інтересів або доступів, а також підказує, які процеси можна автоматизувати. Крім того, така візуалізація допомагає команді розробників

швидко зорієнтуватися у логіці платформи, а замовнику — переконатися, що всі очікувані можливості справді передбачені технічним рішенням.

На рисунку 2.5 зображена діаграма “варіантів використання”

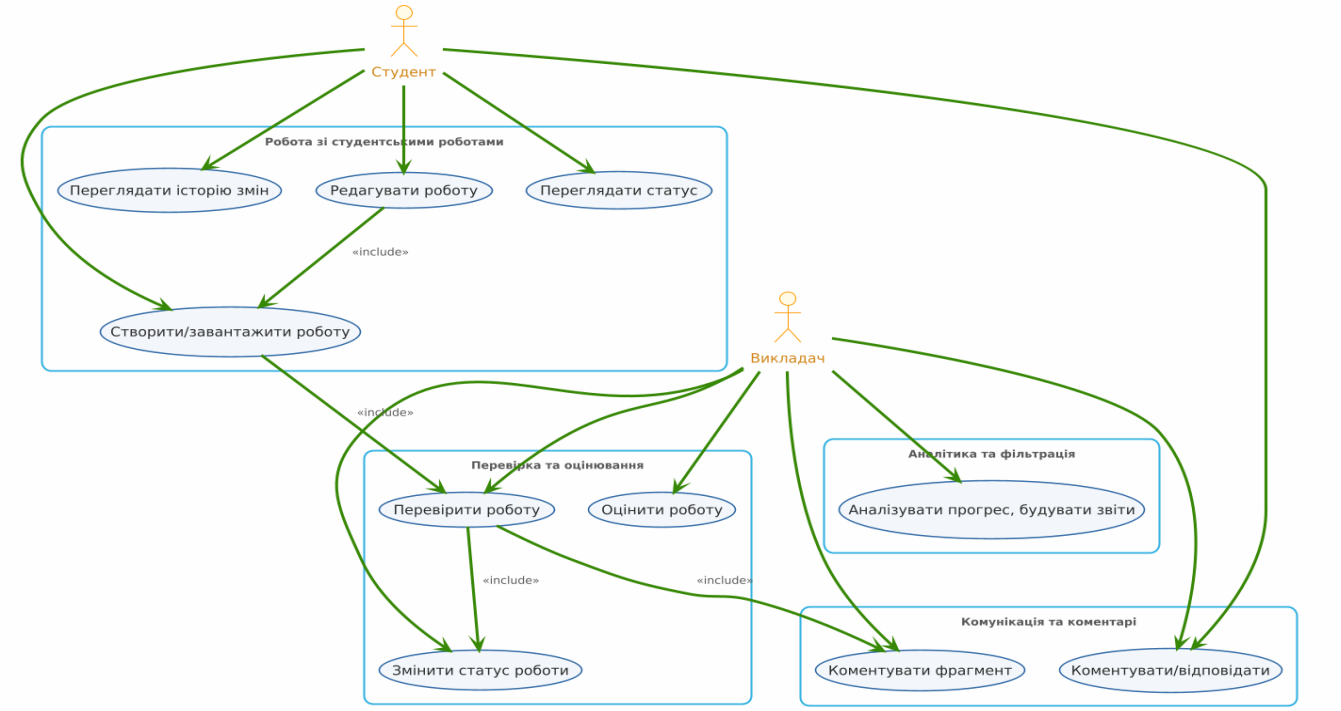


Рис. 2.5 Діаграма варіантів використання

Як випливає зі схеми, система охоплює низку ключових сценаріїв використання. Наведена нижче таблиця 2.2 розкриває зміст діаграми “варіантів використання”(Use Cases)

Таблиця 2.2

Варіантів використання (Use Cases)

Сценарій	**Опис**
Створити/завантажити роботу	Студент створює нову роботу або завантажує файл до системи
Редагувати роботу	Внесення змін до існуючого тексту чи файлу
Переглядати статус	Перегляд актуального стану роботи (“чернетка”, “на перевірці” тощо)

Продовження таблиці 2.2

Варіантів використання (Use Cases)

Сценарій	**Опис**
Переглядати історію змін	Доступ до всіх попередніх версій та ключових подій по роботі
Коментувати/відповідати	Діалог між студентом і викладачем щодо фрагментів роботи
Перевірити роботу	Викладач відкриває роботу для ознайомлення та подальших дій
Коментувати фрагмент	Локальні зауваження до окремих частин тексту
Змінити статус роботи	Оновлення стану (“доопрацювання”, “затверджено” і т.д.)
Оцінити роботу	Виставлення фінальної оцінки
Аналізувати прогрес, будувати звіти	Побудова аналітичних діаграм, перегляд активності, формування звітів

Ретельно спроєктована діаграма варіантів використання підтверджує, що розроблена система дійсно відповідає потребам усіх основних користувачів — як студентів, так і викладачів. Завдяки наочному поділу ролей та дій, кожен сценарій у системі виглядає логічно завершеним, без зайвих або, навпаки, відсутніх кроків. Це дозволяє забезпечити не лише зручність, але й безпеку та прозорість взаємодії.

Важливо підкреслити, що наявність такої діаграми ще до старту реалізації програмного коду дає змогу виявити слабкі місця логіки, уникнути дублювання функцій або ситуацій, коли певна дія виявляється неможливою для конкретної ролі [35]. Для проєктів у сфері освіти, де важлива чіткість, структурованість і контроль, це має надзвичайне значення.

Крім того, діаграма варіантів використання є потужним інструментом комунікації — вона допомагає всім учасникам розробки говорити “однією мовою”,

бути на одній хвилі й розуміти, що саме очікується від системи в межах її цілей [36].

Вона також формує основу для перевірки реалізації вимог, створення тест-кейсів, побудови інтерфейсу та написання інструкцій користувача.

У підсумку можна стверджувати, що діаграма варіантів використання — це не просто додаток до технічного завдання, а повноцінний інструмент планування, контролю і розвитку освітнього ПЗ [37]. Саме завдяки їй платформа може не тільки відповідати поточним запитам навчального процесу, а й бути гнучкою до змін і доповнень у майбутньому.

Висновки до розділу 2

У другому розділі було розгорнуто повноцінний процес формування та структурування вимог до розроблюваної системи моніторингу наукової діяльності студентів. Особливий акцент зроблено на розумінні потреб цільових користувачів — студентів, викладачів та адміністративного персоналу — а також на побудові сценаріїв їхньої взаємодії з платформою у реальному освітньому середовищі.

На основі цього аналізу вдалося чітко окреслити ключові категорії користувачів, визначити їхні ролі, функціональні обов’язки й очікування від системи.

Розробка вимог супроводжувалася глибоким опрацюванням низки критично важливих аспектів: безпеки, зручності використання, чіткого розмежування доступу, підтримки персоналізації інтерфейсу, а також технічної здатності системи до масштабування й подальшого розвитку.

Значну роль у цьому процесі відіграли діаграми варіантів використання (use case diagrams), які дозволили візуалізувати ключові сценарії роботи платформи, забезпечити комплексний підхід до моделювання функцій та уникнути ймовірності втрати важливих елементів під час реалізації.

Сформульовані у цьому розділі функціональні та нефункціональні вимоги стали не лише технічним орієнтиром для розробників, але й концептуальним фундаментом для побудови архітектури системи. Саме вони визначили логіку

структур даних, взаємозв’язки між компонентами, моделі поведінки платформи та механізми реалізації ключових функцій.

Завдяки високому рівню деталізації та системному підходу, підготовлені вимоги дозволяють ефективно перевіряти відповідність кінцевого продукту на всіх етапах його життєвого циклу — від етапу початкової розробки до тестування, розгортання та підтримки.

Це, у свою чергу, створює надійну основу для побудови стабільної, зручної та масштабованої системи, здатної відповідати потребам сучасної вищої освіти та адаптуватися до майбутніх змін.

3 ЗАСОБИ РЕАЛІЗАЦІЇ

3.1 Серверна частина

Серверна частина відіграє ключову роль у роботі всієї системи: саме тут зосереджена логіка обробки запитів, взаємодія з базою даних, перевірка прав доступу, а також загальна координація між користувачами, їхніми ролями та діями [38]. Для реалізації цієї частини було обрано перевірений стек сучасних технологій, що дозволяє створити стабільну, безпечну та масштабовану систему.

Java — одна з найпопулярніших мов програмування, що вже понад 25 років використовується для створення стабільних і безпечних систем [6]. Її головні переваги — кросплатформеність, сувора типізація та велика кількість бібліотек. Саме завдяки цим якостям Java ідеально підходить для розробки складних серверних рішень у сфері освіти, де важливо забезпечити багатокористувацький доступ, розмежування прав і захист даних.

Spring Boot — фреймворк, що значно спрощує створення Java-застосунків [7]. Він автоматизує більшість налаштувань, дозволяє будувати модульну архітектуру, де кожен компонент (контролер, сервіс, репозиторій) має свою чітку роль. Це спрощує тестування, розширення і підтримку системи.

Maven — інструмент, який допомагає впорядкувати проєкт, підключити всі необхідні бібліотеки та підтримувати єдину структуру [8]. Його використання дозволяє швидко конфігурувати середовище розробки та мінімізувати рутинну роботу з налаштуванням проєкту.

Spring Data JPA — це розширення до Spring, що реалізує ORM (Object-Relational Mapping) і дозволяє працювати з базою даних, використовуючи звичайні Java-об’єкти [10]. У результаті стандартні операції — додавання, оновлення, видалення — виконуються без написання SQL, що зменшує кількість помилок і прискорює розробку.

Spring Security — фреймворк, відповідальний за безпеку веб-застосунків [16]. Він дозволяє реалізувати автентифікацію, авторизацію, обмежити доступ до функцій залежно від ролі користувача, а також забезпечує захист від поширених атак.

MVC (Model-View-Controller) — архітектурна модель, що логічно розділяє систему на три основні частини [17]. Model відповідає за збереження і обробку даних, View — за відображення інформації користувачу, а Controller — за зв’язок між цими двома частинами. Такий підхід спрощує розробку, тестування і підтримку, дозволяючи працювати з кодом більш структуровано і зрозуміло.

Усі ці компоненти в сукупності створюють надійну й гнучку серверну архітектуру, яка легко адаптується під зміни, дозволяє розвивати систему без значного переписування коду та забезпечує стабільну роботу в умовах навчального процесу з великою кількістю користувачів.

3.2 Клієнтська частина

Користувацький інтерфейс — це саме те, з чим взаємодіє студент або викладач щодня. Він повинен бути зручним, інтуїтивно зрозумілим і доступним на будь-якому пристрої. Для створення такого інтерфейсу були обрані сучасні веб-технології, які забезпечують адаптивність, привабливий вигляд та зручну навігацію.

HTML5 — стандарт розмітки, який забезпечує логічну структуру веб-сторінок та коректне відображення на різних пристроях [11]. Завдяки йому реалізовано каркас інтерфейсу: особисті кабінети, список робіт, модальні вікна, форми коментування. HTML5 також дозволяє легко поєднувати контент із мультимедійними елементами, інтегрувати JavaScript та забезпечувати зручну навігацію всередині системи.

CSS3 — мова стилів, що відповідає за візуальне оформлення інтерфейсу [12]. Вона підтримує адаптивність, плавні переходи, анімації та медіа-запити, що робить зовнішній вигляд сучасним і приємним для користувача. У проєкті CSS3 працює у

парі з Tailwind, що дало змогу значно пришвидшити розробку та забезпечити стилістичну цілісність.

Tailwind CSS — утилітарний фреймворк для стилізації інтерфейсу, який базується на підході "utility-first" [13]. Замість написання окремих CSS-файлів, розробник застосовує готові класи прямо у HTML, що зменшує обсяг коду, спрощує підтримку стилів і пришвидшує внесення змін. Завдяки Tailwind створено чистий, гнучкий дизайн, який легко адаптується до будь-якого пристрою.

JavaScript — мова, що відповідає за динамічність і логіку на боці користувача [14]. Вона забезпечує інтерфейсну взаємодію без перезавантаження сторінок: відкриття модальних вікон, обробку подій, виділення тексту для коментування, асинхронну відправку даних на сервер через fetch/AJAX. Таким чином, користувач може взаємодіяти з системою комфортно, без затримок і зайвих оновлень.

Thymeleaf — шаблонізатор, що використовується для генерації HTML-сторінок на сервері з урахуванням динамічних даних [15]. Він дозволяє персоналізувати інтерфейс відповідно до ролі користувача, показувати статуси робіт, повідомлення про помилки, фільтрувати або сортувати дані. Завдяки інтеграції з Spring Boot, Thymeleaf забезпечує гнучкий контроль відображення кожної сторінки і дає змогу будувати зручну багатосторінкову структуру.

Разом ці технології дозволили створити інтерфейс, який виглядає сучасно, працює швидко та легко сприймається користувачами різного рівня підготовки. Клієнтська частина не тільки забезпечує зовнішній вигляд, а й активно взаємодіє з сервером, формуючи єдиний комфортний простір для ефективної співпраці між студентами й викладачами.

3.3 База даних

Жодна система не може повноцінно працювати без надійного механізму зберігання й організації даних. Саме базу даних можна вважати серцем застосунку, де фіксуються усі суттєві елементи — користувачі, наукові роботи, статуси перевірки, коментарі викладачів, ролі та їхні права. Від ефективності побудови БД

залежить не тільки швидкість роботи системи, а й зручність обробки, аналізу та захисту даних.

MySQL — це одна з найпопулярніших реляційних систем управління базами даних, що поєднує високу продуктивність із стабільністю роботи [9]. Вона активно застосовується у веб-проєктах різного масштабу, завдяки відкритому коду, хорошій документації й підтримці транзакцій. У нашій системі MySQL дозволяє логічно структурувати всі дані, підтримувати зв’язки між таблицями через зовнішні ключі (foreign keys), а також забезпечити швидкий пошук і надійність зберігання навіть при великому обсязі інформації.

ER-діаграма — це візуальна модель структури бази даних, яка допомагає зрозуміти логіку зв’язків між сутностями. У запропонованому проєкті виділено кілька ключових таблиць: User, Work, Comment, Role, WorkStatus. Кожна з них відповідає окремому об’єкту: наприклад, Work містить інформацію про наукову роботу, має зв’язки з автором (User), статусом (WorkStatus) та коментарями (Comment). Такий підхід дає змогу зберігати структуру логічною, а дані — цілісними. ER-діаграма особливо корисна на етапі обговорення, тестування або доопрацювання системи, бо дозволяє швидко зорієнтуватися в її архітектурі.

Робота з даними через ORM реалізована за допомогою Spring Data JPA — потужного модуля, який автоматично перетворює таблиці у Java-класи (entity), створює схеми під час запуску застосунку, генерує запити на основі назв методів у репозиторіях. Завдяки ORM зменшується потреба у написанні "ручного" SQL-коду, підвищується безпека (запити перевіряються на коректність і не допускають SQL-ін’єкцій), а сама розробка стає значно швидшою й зрозумілішою.

Побудована модель бази даних забезпечує необхідний баланс між швидкодією, гнучкістю і безпекою. Завдяки використанню MySQL у поєднанні зі Spring Data JPA вдалося створити структуру, яка легко підтримується, масштабується, та дозволяє швидко додавати нові функції, не змінюючи фундаментальні принципи роботи із даними.

3.4 Інтеграція компонентів

Щоб система працювала злагоджено, всі її частини мають бути добре взаємопов’язаними — як єдиний механізм, де кожна шестерня виконує свою роль. У даному проєкті реалізовано багаторівневу архітектуру, що поєднує серверну логіку, базу даних та клієнтський інтерфейс. Такий підхід дозволяє підтримувати чіткий розподіл обов’язків між компонентами та забезпечити стабільну і прогнозовану взаємодію.

REST API і серверна логіка реалізовані на базі Java з використанням Spring Boot. Усі запити, які надходять від користувача — перегляд, додавання, оновлення або коментування наукової роботи — обробляються контролерами. Ці контролери відповідають за перевірку прав доступу, взаємодію із бізнес-логікою та передачу команд до бази даних через відповідні сервіси та репозиторії. Щоб забезпечити безпечний доступ до функцій системи, використовується Spring Security, який розділяє права між студентами, викладачами та адміністраторами.

Генерація сторінок на сервері відбувається за допомогою шаблонізатора Thymeleaf. Він дозволяє динамічно формувати вміст сторінок з урахуванням ролі користувача, поточного статусу роботи, наявних коментарів тощо. Такий підхід забезпечує персоналізацію інтерфейсу — наприклад, студент бачить тільки свої документи і коментарі до них, а викладач має доступ до всіх робіт, які перевіряє.

Інтерактивність на клієнті забезпечується за допомогою JavaScript. Завдяки скриптам реалізовано обробку дій користувача без перезавантаження сторінки — відкриття вікон, підсвічування фрагментів тексту, перевірка форм, асинхронна взаємодія із сервером. Такий підхід суттєво покращує досвід користувача (UX) і робить роботу в системі більш динамічною та зручною.

DTO (Data Transfer Object) виступають сполучною ланкою у передачі даних між користувачем і сервером. Вони дозволяють чітко контролювати, яка саме інформація потрапляє на фронтенд, не відкриваючи зайвих внутрішніх деталей реалізації. Це важливо як для безпеки, так і для гнучкості — будь-які зміни у структурі бази не вимагають негайної перебудови всього клієнтського коду.

Можливість масштабування також закладена на рівні архітектури. Завдяки використанню стандартних протоколів та розділенню обов’язків, до системи в майбутньому легко можна буде додати нові модулі — наприклад, електронну розсилку, інтеграцію з навчальними платформами (LMS) або розширені інструменти для аналітики. Це не потребуватиме кардинальної перебудови всього проєкту й дозволить зберегти стабільність при зростанні навантаження.

У результаті, така модель побудови інтеграції дозволяє усім компонентам працювати узгоджено, забезпечуючи гнучкість для майбутніх оновлень, зручність у підтримці та надійність у щоденній роботі системи.

3.5 Інструменти тестування та розгортання

Щоб будь-яка інформаційна система працювала стабільно, без збоїв і непередбачуваних зупинок, важливо не лише її правильно створити, а й належним чином перевірити та впровадити [39]. Саме тестування і розгортання визначають якість фінального продукту та готовність системи до щоденного використання в реальному освітньому середовищі.

Тестування серверної частини виконується для зручності за допомогою ретельно відібраних інструментів: Spring Boot Test, JUnit і Mockito. Завдяки Spring Boot Test вдається моделювати повноцінне середовище з усіма налаштуваннями і перевіряти, як система поводиться в умовах, наближених до бойових. JUnit відповідає за юніт-тести — тобто перевірку окремих функцій, методів або компонентів: контролерів, сервісів, репозиторіїв. А Mockito, у свою чергу, дозволяє «відрізати» частини логіки за допомогою імітацій і сфокусуватися на тестуванні конкретного фрагмента. Такий підхід дозволяє швидко виявляти помилки, мінімізувати ризики регресій і зберігати впевненість у стабільності після кожного оновлення.

Користувацька частина тестується переважно вручну, оскільки саме тут взаємодія користувача з інтерфейсом має бути максимально природною і зручною. Перевіряється коректність усіх форм, функціональність інтерфейсу коментування,

відображення статусів робіт, реакція системи на помилкові введення, адаптивність до різних пристроїв і браузерів. У цього процесу є можливість автоматизуватися за допомогою бібліотек типу Selenium чи Cypress. Вони дозволяють знімати навантаження на етапі перевірки та моделювати поведінку користувача.

Процес розгортання організовано із застосуванням принципів контейнеризації. Зокрема, використання Docker дає змогу запускати систему в ізольованому середовищі — незалежно від того, яка операційна система використовується на сервері. Це робить розгортання передбачуваним і зменшує кількість помилок, пов’язаних із різними налаштуваннями середовища.

Таким чином, поєднання ретельного тестування й автоматизованого розгортання дозволяє не лише зберігати якість продукту на високому рівні, а й оперативно впроваджувати нові зміни, зберігаючи при цьому довіру користувачів до системи.

Висновки до розділу 3

У третьому розділі дипломної роботи зосереджено увагу на ключових рішеннях, що стосуються вибору інструментів та засобів реалізації веб-платформи для моніторингу наукової діяльності студентів. Це був важливий етап, на якому технічна ідея проєкту почала набувати реальних обрисів, а концепція — перетворюватися на повноцінний програмний продукт. У контексті стрімкого розвитку ІТ-сфери та великої кількості доступних технологій, особливо важливо було не лише вибрати сучасні засоби, а й обґрунтувати їхню доцільність саме для задач системи моніторингу.

У межах серверної частини платформи ключовим рішенням стало використання мови програмування Java в поєднанні з фреймворком Spring Boot. Такий вибір обумовлений вимогами до стабільності, масштабованості та безпеки системи.

Для роботи з базою даних було впроваджено Spring Data JPA — потужний інструмент для організації об’єктно-реляційного зв’язку (ORM), який спростив доступ до даних і зменшив кількість "ручного" SQL-коду. У ролі СКБД використано перевірену MySQL, що гарантує швидкість обробки запитів,

цілісність даних і масштабованість навіть у випадку зростання обсягів інформації. Важливу роль у зручності збирання, запуску та підтримки залежностей відіграла система керування проєктом Maven, яка також сприяє підтримці модульності та чистоти структури проєкту.

На клієнтському рівні було обрано набір сучасних веб-технологій — HTML5, CSS3, JavaScript, шаблонізатор Thymeleaf і бібліотеку стилів Tailwind CSS. Такий стек забезпечив створення адаптивного, логічно структурованого й візуально привабливого інтерфейсу, зручного як для студентів, так і для викладачів. Tailwind CSS особливо допоміг при розробці уніфікованих стилів і прискорив верстку, а інтеграція з Thymeleaf дозволила гнучко керувати вмістом сторінок, відображати динамічні дані та регулювати доступ до функцій на основі ролей користувачів.

Окрему увагу було приділено взаємодії між усіма компонентами системи. Важливо було не лише реалізувати кожен із модулів, а й налаштувати їхню коректну роботу як єдиного цілого — від браузера користувача до серверної логіки й бази даних. Налагодження стабільної обробки HTTP-запитів, передача даних через DTO, обробка помилок і забезпечення зворотного зв’язку — все це стало основою для надійного функціонування платформи.

Також розглянуто засоби тестування та автоматизації, що дозволили перевіряти працездатність розроблених модулів на різних етапах і швидко ідентифікувати можливі помилки. Цей підхід сприяв підвищенню загальної якості коду та стабільності системи в реальному використанні.

Підсумовуючи, можна зазначити, що третій розділ став технічною “серцевиною” проєкту. Завдяки обґрунтованому вибору технологій, їхній сумісності та послідовній інтеграції було закладено надійну технологічну основу, яка повністю відповідає функціональним вимогам системи. Така архітектура не лише забезпечує зручність поточного використання, а й відкриває широкі можливості для подальшого розвитку платформи у відповідь на нові виклики освітнього середовища.

4 ПРОЄКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Архітектура

Як вже було показано у попередніх розділах, створена система моніторингу наукової діяльності студентів ґрунтується на сучасному підході до побудови програмного забезпечення — багаторівневій архітектурі, що поєднує зручність, стабільність і можливість подальшого розвитку [19]. Та попри окремий опис кожного з технічних компонентів, лише архітектурна схема дозволяє по-справжньому комплексно побачити загальну картину. Саме вона показує, як пов’язані між собою різні частини системи, якими шляхами рухаються дані та як відбувається взаємодія між користувачами й програмною логікою. Ця схема — не просто технічна формальність, а один із ключових етапів розробки, адже вона створює міцну основу для всієї подальшої реалізації. Вона слугує «архітектурним кресленням», яке дозволяє візуалізувати і структуру, і взаємозв’язки, і логіку роботи системи. Такий підхід дає змогу уникнути плутанини, підвищити продуктивність розробки та забезпечити прозоре тестування. А головне — він дозволяє легко масштабувати й оновлювати систему в майбутньому без необхідності повністю переписувати її структуру.

Для більш наочного розуміння архітектурного підходу до проєктування платформи, у цьому розділі представлено схему, яка відображає всі основні складові системи, їхні зв’язки та функціональні ролі [20]. Цей графічний матеріал дозволяє швидко оцінити, які саме компоненти беруть участь у кожному процесі, як забезпечується цілісність даних, безперебійна взаємодія та відповідність вимогам масштабованості й безпеки.

В моєму застосунку є такі основні рівні архітектури:

Клієнтський рівень (Frontend)

Цей рівень — це те, що бачить і чим користується студент чи викладач. Веб-інтерфейс відповідає за відображення сторінок, форм, таблиць, вікон коментування та всіх інших елементів взаємодії. Тут реалізовано сучасний адаптивний дизайн,

завдяки якому система однаково зручно працює як на комп’ютері, так і на планшеті чи смартфоні. Швидка реакція на дії користувача й інтуїтивна структура дозволяють зосередитися не на технічних деталях, а на суті — роботі з науковими текстами.

Серверний рівень (Backend)

Це «мозок» системи, де відбувається обробка всіх запитів, перевірка доступу, зміна статусів, збереження нових версій робіт, управління правами користувачів. Тут реалізовані всі бізнес-процеси, контролери та сервіси [22]. Саме цей рівень визначає, як саме працює система — які дії дозволені, що і в який момент можна робити, як реагувати на певні сценарії. Він не лише координує обмін даними, а й гарантує стабільність і безпеку платформи.

Рівень доступу до даних (Data Access Layer)

Цей рівень — зв’язуюча ланка між серверною логікою та самою базою даних. За допомогою ORM (зокрема Spring Data JPA) система перетворює таблиці в об’єкти, з якими розробники можуть працювати зручно та без необхідності вручну писати складні SQL-запити. Це значно знижує ризик помилок, прискорює розробку й дозволяє зосередитись на логіці, а не на технічних деталях взаємодії з БД.

База даних (MySQL)

Центральне сховище всіх ключових елементів системи: облікових записів, наукових робіт, коментарів, статусів, історії змін [21]. Структура бази оптимізована під швидкий пошук і обробку даних. Завдяки використанню реляційної моделі легко встановлюються зв’язки між сутностями, забезпечується цілісність даних, а також високий рівень захисту від втрат і помилок.

На рисунку 4.1 зображена архітектурна схема системи, яка візуально демонс трує логіку побудови, розподіл відповідальностей між компонентами та шлях руху інформації — від введення даних користувачем до їх обробки й збереження на сервері.

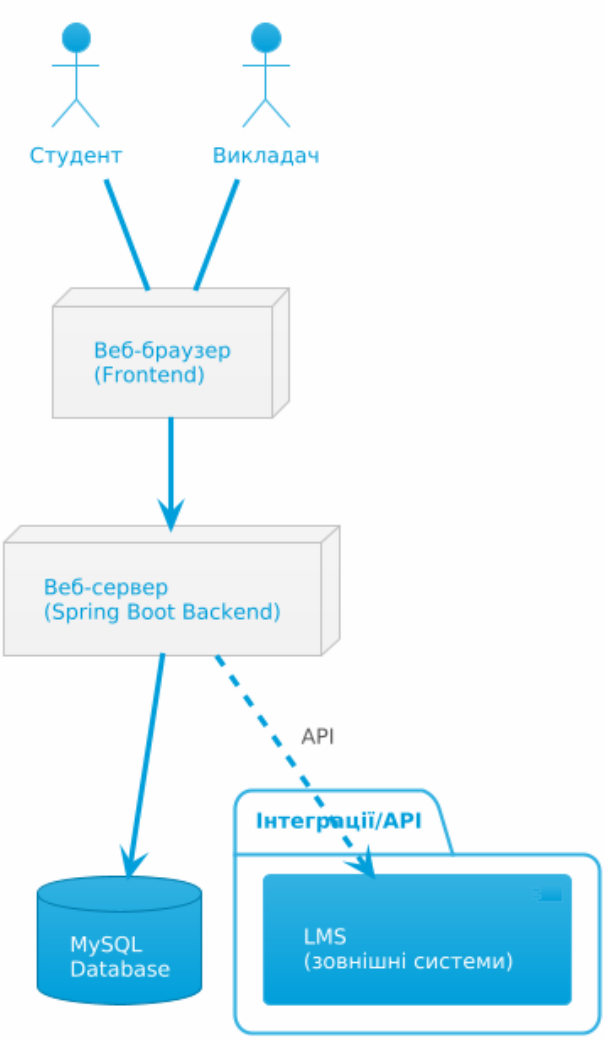


Рис. 4.1 Архітектурна схема системи

Архітектурна модель, наведена на діаграмі, дозволяє побачити цілісну картину функціонування системи — від дій користувачів до збереження й обробки даних на серверному рівні. Такий підхід дає змогу не лише впорядкувати логіку роботи платформи, але й закладає надійну основу для її стабільності, безпеки та розвитку.

4.2 Структури даних, моделі, DTO

Для будь-якої інформаційної системи структура даних — це не просто набір таблиць або схем. Це її внутрішній "скелет", навколо якого вибудовується вся логіка, функціональність і життєвий цикл. Саме від якості проєктування моделей залежить, наскільки ефективно система працюватиме під навантаженням, як легко

буде реалізовувати нові функції, забезпечувати безпеку чи масштабувати платформу з плином часу.

У системі моніторингу студентської наукової діяльності підхід до побудови моделей та передачі даних не був випадковим або вторинним. Навпаки — йому приділялася особлива увага ще на ранньому етапі розробки [23]. Це дозволило створити надійну, логічно завершену та технологічно грамотну основу, яка не лише відповідає поточним потребам користувачів, а й має достатній запас гнучкості на майбутнє.

В основі цієї реалізації лежить набір чітко визначених сутностей — тобто «ключових гравців» системи. Кожна з них виконує свою роль: хтось зберігає інформацію про користувача, інша відповідає за наукову роботу, ще інша — за історію змін чи коментарі.

Розробка моделей розпочалася зі створення ER-діаграми (Entity-Relationship diagram) [24]. Це був своєрідний “архітектурний кресленик”, на якому поступово окреслювалися всі основні об’єкти системи та зв’язки між ними. Такий підхід дозволив не тільки сформувати повну картину структури, а й уникнути дублювання інформації, логічних суперечностей чи втрати важливих даних під час складних операцій.

Одним із завдань було забезпечити референційну цілісність, тобто гарантувати, що зв’язки між сутностями не “висять у повітрі”, а логічно підтримуються — наприклад, робота не може існувати без автора, а коментар — без конкретної роботи або фрагмента. Завдяки цьому вдалося побудувати структуру, яка відповідає реальному сценарію користування платформою — від реєстрації користувача до оцінювання проєкту.

Основні моделі системи

- Користувач (User) — центральна точка взаємодії з системою. Зберігає логін, email, роль, історію входів. Від цієї моделі “розходяться” всі подальші дії: завантаження роботи, коментування, перегляд статистики.

- Робота (Work або ResearchWork) — ключова модель, що описує науковий проєкт. У ній міститься не лише заголовок і короткий опис, а й статус, дедлайни,

версії файлів та інші атрибути, важливі для супроводу повного життєвого циклу роботи.

- Коментар (Comment) — реалізує живу взаємодію між студентом і викладачем. Може бути загальним або точковим (прив’язаним до певного фрагмента). Такі коментарі стають основою зворотного зв’язку, допомагають покращити якість роботи ще до її остаточного захисту.

- Роль (Role) — окрема, але критично важлива сутність, яка регулює, що саме дозволено робити кожному користувачеві. Цей механізм дозволяє підтримувати чітку політику доступу, захист від несанкціонованих дій та контроль за прозорістю взаємодії.

- Статус роботи (WorkStatus) — відображає поточну фазу проєкту: від чернетки до затвердженої версії. Завдяки цьому і студент, і викладач завжди знають, на якому етапі перебуває робота, що ще потрібно зробити й хто має ініціативу на даний момент.

- Історія змін (Event або History) — несе відповідальність за фіксацію всіх важливих змін у проєкті. Це може бути зміна статусу, нова версія файлу, доданий коментар. Такий “щоденник” дій — один з елементів, що підвищує довіру до системи та забезпечує академічну доброчесність.

Окрім самих моделей, у системі активно використовуються DTO-об’єкти — спеціальні «транспортні контейнери», які дозволяють передавати дані між клієнтом і сервером без ризику витоку чутливої інформації. Вони зменшують навантаження на API, спрощують перевірку даних і знижують кількість помилок у процесі інтеграції компонентів.

Приклади DTO:

- UserDto — передає лише базову інформацію про користувача, без паролів чи службових полів.

- WorkDto — надає дані про тему роботи, опис, статус, автора, дедлайни.

- CommentDto — включає текст, дату створення, автора та позицію у тексті, до якої прив’язано коментар.

- LoginDto, StatusGradePayload — використовуються в авторизації, зміні статусів, виставленні оцінок.

На рисунку 4.2 зображена UML-діаграма класів у розроблюваному застосунку, яка є логічною основою реалізації всієї бізнес-логіки, збереження інформації та побудови користувацьких сценаріїв. Саме ця діаграма дозволяє побачити “всю картину” — звідки й куди рухаються дані, що з чим пов’язано і як це реалізується в реальній роботі системи.

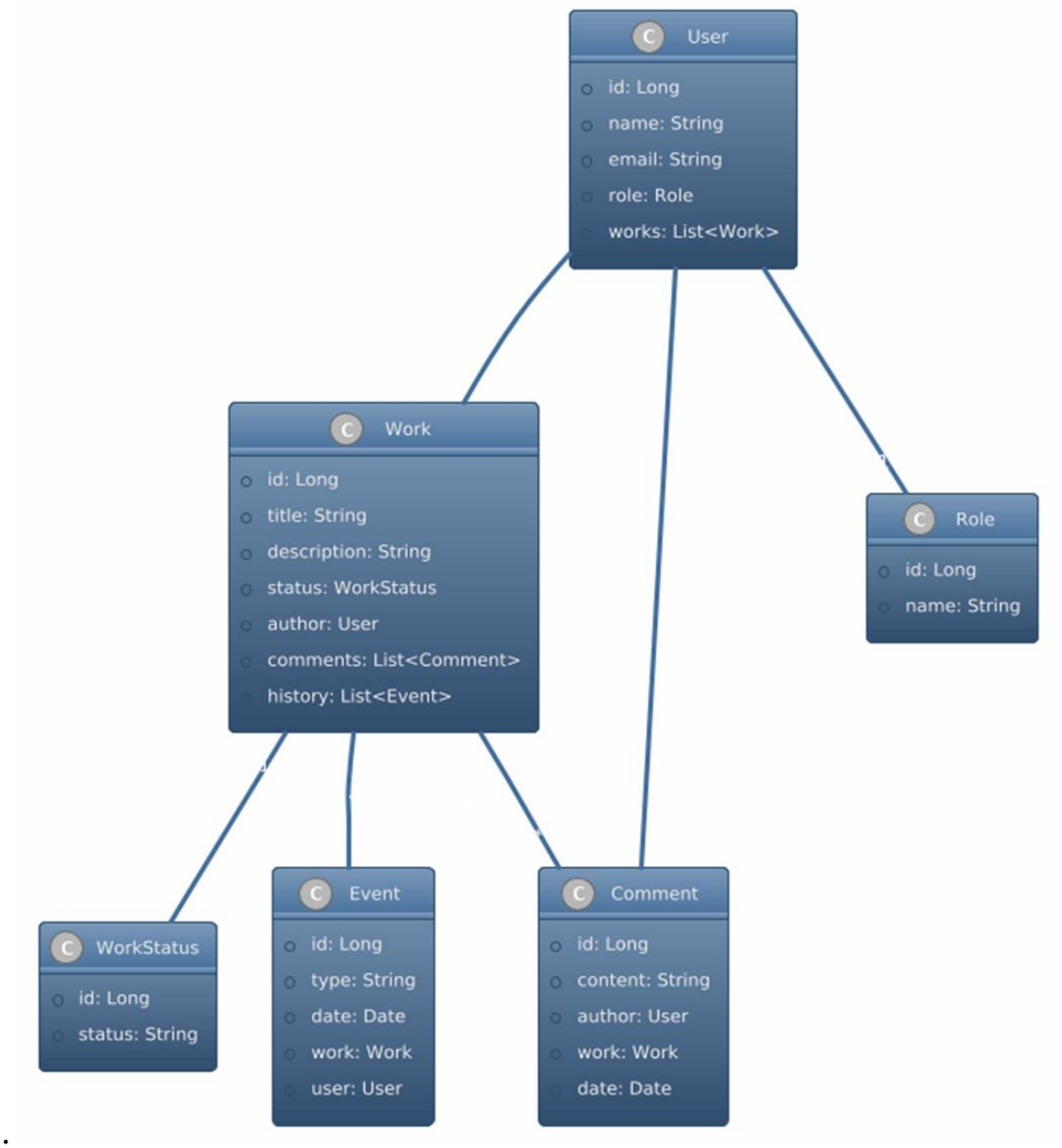


Рис. 4.2 UML-діаграма класів

Підсумовуючи, такий підхід до структурування даних дозволяє не лише ефективно організувати всі ключові бізнес-процеси системи, а й гарантує логічну узгодженість, безпечну передачу інформації між компонентами й легкість у підтримці. Завдяки добре спроєктованим моделям і DTO-схемам, система готова до масштабування, зміни бізнес-логіки й інтеграції з іншими платформами у майбутньому.

4.3 Ключові контролери

У кожній сучасній веб-платформі важливо не лише реалізувати необхідні функції, а й грамотно організувати внутрішню логіку їх виконання. У цьому ключі контролери відіграють одну з найважливіших ролей — саме вони відповідають за те, як система реагує на дії користувача, як обробляє запити та як забезпечує взаємодію між зовнішнім інтерфейсом і бізнес-логікою. У створеній системі контролери — це свого роду "мозок" усіх користувацьких сценаріїв, який працює у зв’язці з іншими компонентами (сервісами, базою даних, механізмами безпеки) та забезпечує стабільну й передбачувану поведінку додатка в будь-яких ситуаціях.

В основі побудови контролерів використано принцип розділення відповідальностей (Separation of Concerns). Це означає, що кожен контролер виконує лише ті завдання, які прямо стосуються його сфери: автентифікація, робота з науковими проєктами, обробка коментарів тощо. Усі складні операції (на кшталт перевірки прав доступу, взаємодії з базою, обробки файлів, логування дій) винесено в окремі сервіси, що дозволяє зробити код більш чистим, а систему — більш гнучкою й легкою у підтримці. У разі потреби додати нову функцію або змінити вже реалізовану логіку, достатньо змінити відповідний контролер, не торкаючись інших частин системи.

Реалізовано декілька основних контролерів:

AuthController (контролер автентифікації)

Цей контролер відповідає за процеси входу, реєстрації та виходу користувача з системи. Тут реалізовано обробку логіна, перевірку пароля, створення сесій або

токенів (JWT) для доступу до захищених сторінок. Завдяки чітким механізмам автентифікації кожен користувач отримує доступ лише до тих розділів, які дозволяє його роль, що забезпечує безпеку й контроль доступу.

UserController (контролер користувачів)

Цей контролер працює з персональними профілями: дозволяє переглядати й оновлювати особисту інформацію, змінювати налаштування, переглядати історію робіт або активностей.

WorkController (контролер робіт)

Один із найважливіших компонентів системи. Він обробляє всі дії, пов’язані з науковими роботами: від створення нової роботи до її рецензування та затвердження. Подача, редагування, перегляд, завантаження версій, зміна статусу (“чернетка”, “потребує доопрацювання” тощо) — усе це відбувається саме тут. Контролер також опрацьовує запити на фільтрацію, сортування й взаємодію з коментарями.

CommentController (контролер коментарів)

Відповідає за двосторонню комунікацію між студентом і викладачем. Додавання нових коментарів, редагування, видалення, перегляд за часом або фрагментом тексту — усе це реалізовано через цей контролер. Він тісно інтегрується з WorkController, оскільки кожен коментар належить до певної роботи й конкретного місця в ній.

AdminPageController (контролер адміністративних сторінок)

Цей компонент забезпечує керування налаштуваннями системи: від перегляду статистики й активності користувачів до зміни політик безпеки або запуску резервного копіювання.

HomeController, WorkPageController, CommentsPageController

Ці контролери відповідають за загальну навігацію в системі. Вони керують відображенням основних сторінок: списків робіт, розділів коментарів, форм пошуку й фільтрації. Саме через них реалізується логіка роботи фронтенду, тобто взаємодії користувача із візуальними елементами інтерфейсу.

На таблиці 4.1 зображені основні endpoint-и застосунка.

Таблиця 4.1

Основні endpoint-и

Контролер	**Основні endpoint-и**	**Опис функції**
AuthController	/login,/logout, /register	Логін, вихід, реєстрація
UserController	/users/{id}, /users/me	Перегляд профілю, власного кабінету
WorkController	/works, /works/{id}, /works/{id}/status	Створення, перегляд, редагування, зміна статусу
CommentController	/comments, /comments/{id}	Додавання/редагування/видалення коментарів

На рисунку 4.3 зображено діаграму контролерів у розробленому застосунку.

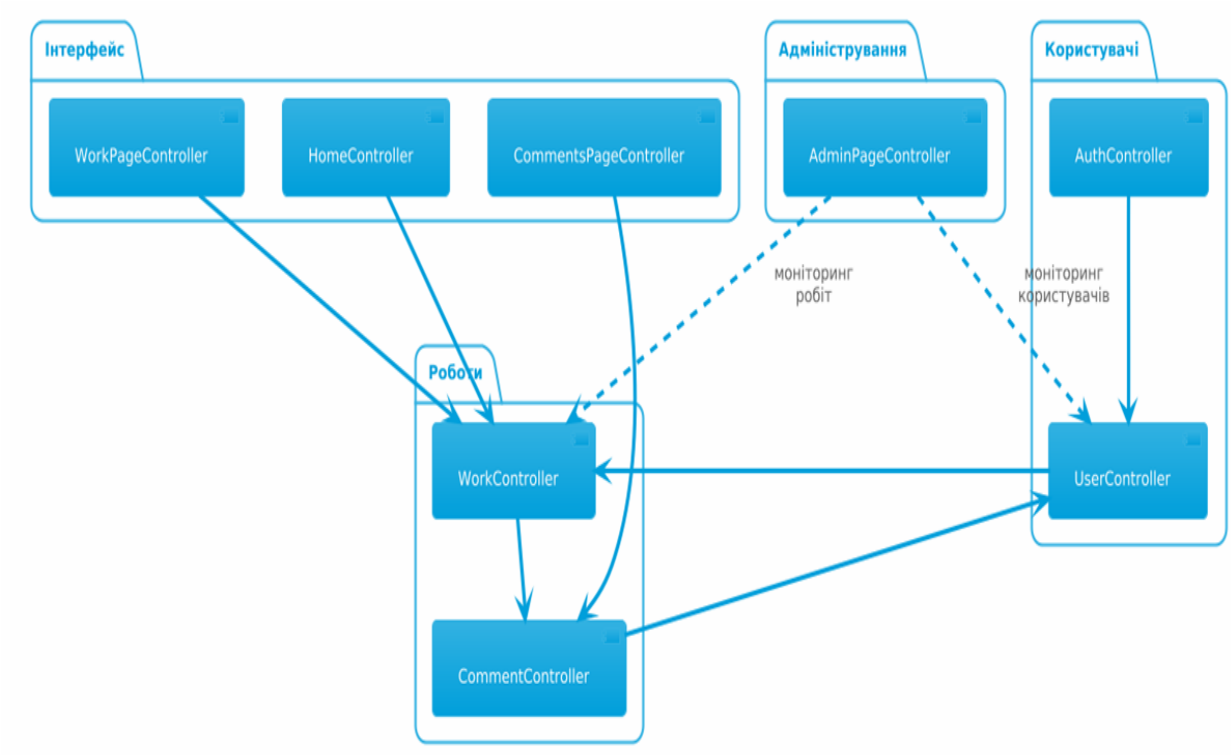


Рис. 4.3 Діаграма контролерів

У цілому така структурована організація контролерів дозволяє не тільки ефективно підтримувати код та розширювати систему у майбутньому, а й забезпечує передбачувану, безпечну й зручну роботу всіх модулів платформи.

4.4 Користувацький інтерфейс

Користувацький інтерфейс — це не просто графічне оформлення платформи. Це — міст між людиною та цифровою системою, через який відбувається вся взаємодія, приймаються рішення, обмінюється інформація. Від того, наскільки інтуїтивно зрозумілим, логічним і комфортним буде цей інтерфейс, безпосередньо залежить і успішність використання системи, і рівень задоволеності її користувачів.

Саме тому при проєктуванні інтерфейсу даної системи особливу увагу було приділено деталям: розташуванню елементів, зручності переходів, кольоровій палітрі, реакції на дії користувача. Студент має відчути, що все знаходиться “під рукою”, а викладач — що жодна функція не захована надто глибоко.

Реалізовано декілька основних принципів UX/UI-дизайну платформи:

Навігаційне меню побудовано за принципом “мінімум кроків — максимум результату”. Основні розділи виділено в окремі кнопки або вкладки: “Мої роботи”, “Коментарі”, “Новий проєкт” тощо. Іконки, підказки та логічне групування допомагають швидко зорієнтуватися навіть при першому вході.

Tailwind CSS більш гармонічний і ідеально підходить для реалізації інтерфейса щоб він виглядав більш сучасно та послідовно: всі кнопки, повідомлення, заголовки виконані в єдиному стилі. Використання контрастних кольорів дозволяє акцентувати увагу на важливих елементах.

Інтерфейс автоматично підлаштовується під будь-який розмір екрана. Робота з платформи однаково зручна як з ноутбука, так і зі смартфона. Це особливо важливо для студентів, які часто працюють “на ходу” або з дому.

Усі форми мають підказки, автоматичну валідацію даних, сповіщення про помилки й підтвердження виконаних дій. Це дозволяє уникнути помилок ще до того, як вони потраплять у базу.

Кожна сторінка акцентує увагу користувача на головному. Наприклад, кнопки “Подати на перевірку” або “Залишити коментар” завжди розташовані у видимій частині екрану, не потребують довгого пошуку і підсвічуються для зручності.

Перейдемо до макетів основних сторінок. Для прикладу виберемо декілька реалізованих сторінок застосунку, а саме: Сторінка Регестрації та Логіну, Головна сторінка, Роботи студента, Адмін панель, Форма коментування та Редагування роботи. Почнемо по порядку:

1. Сторінка реєстрації та логіну:

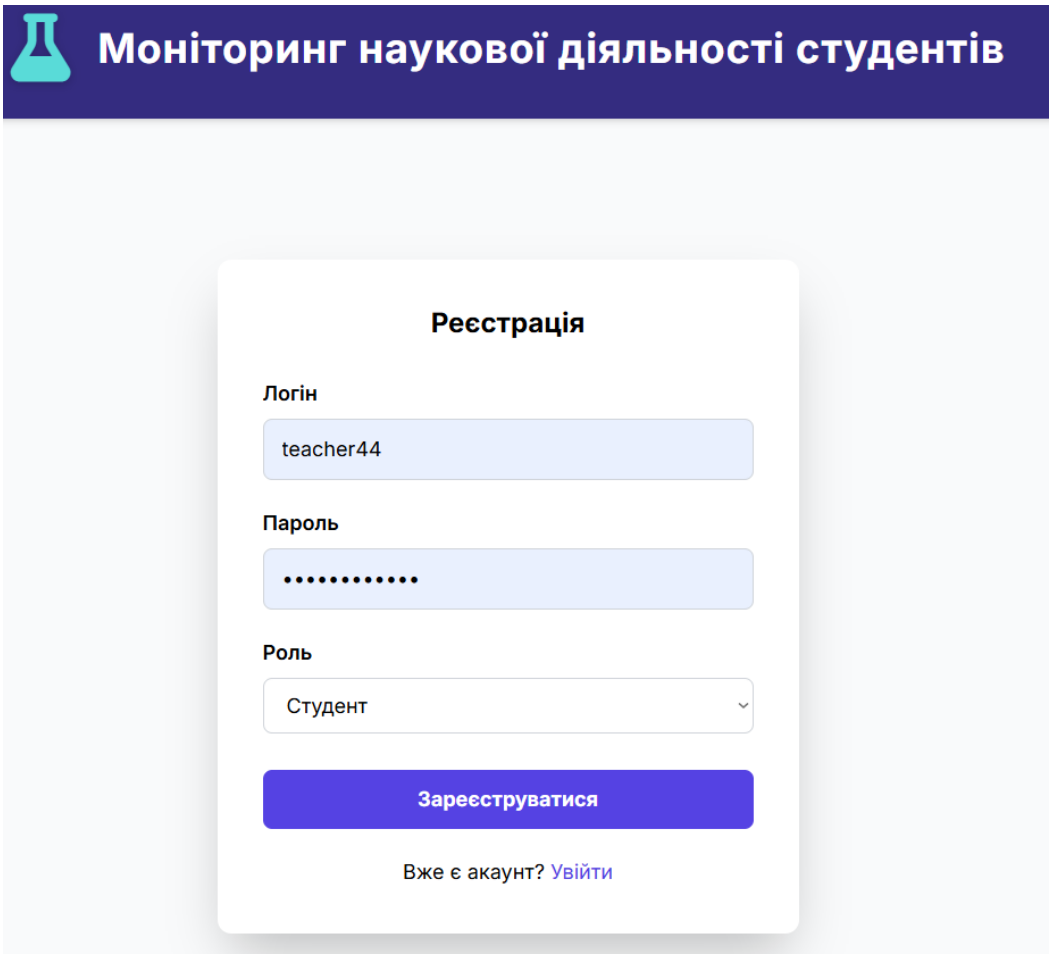


Рис. 4.4 Форма реєстрації нового користувача у системі моніторингу наукової діяльності студентів.

Передбачено введення логіна, пароля та вибір ролі (студент чи викладач) для подальшої роботи з платформою.

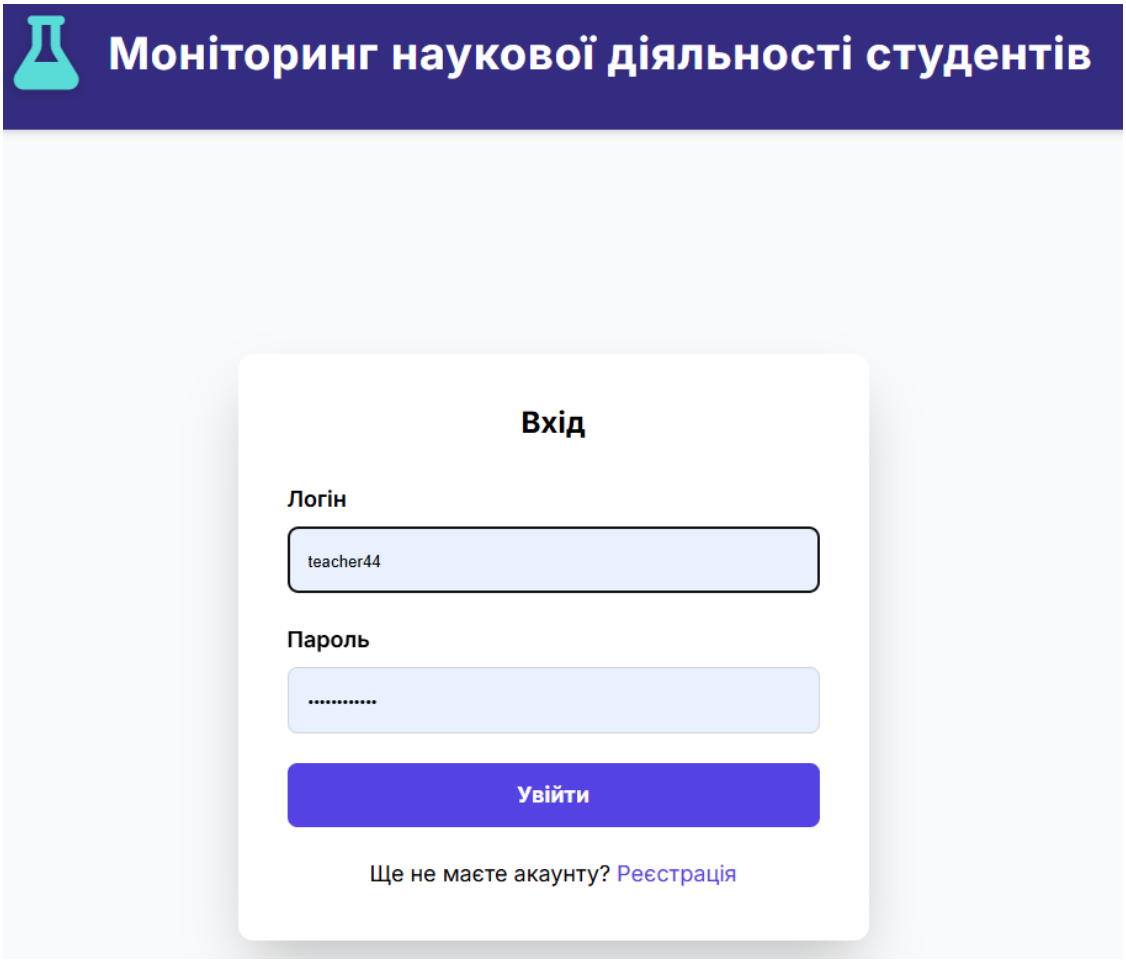


Рис. 4.5 Форма входу користувача до системи.

На сторінці передбачено поля для введення логіна й пароля, а також посилання на реєстрацію для нових користувачів

2. Головна сторінка:

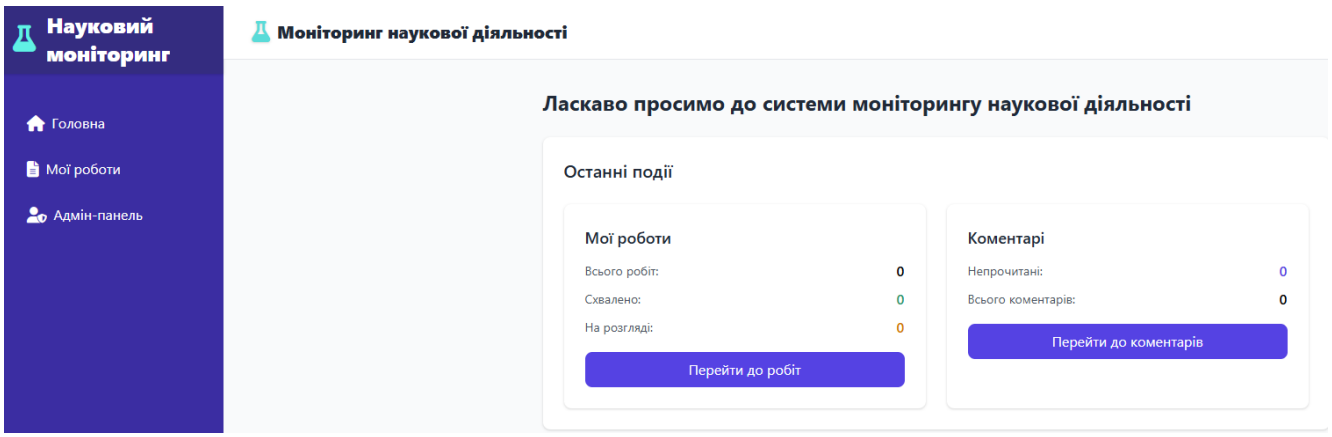


Рис. 4.6 Головна сторінка користувача в системі моніторингу наукової діяльності студентів.

На сторінці відображаються останні події (оновлення робіт, нові коментарі), загальна статистика по роботах і коментарях, а також швидкі кнопки для переходу до відповідних розділів.

3. Роботи студента:

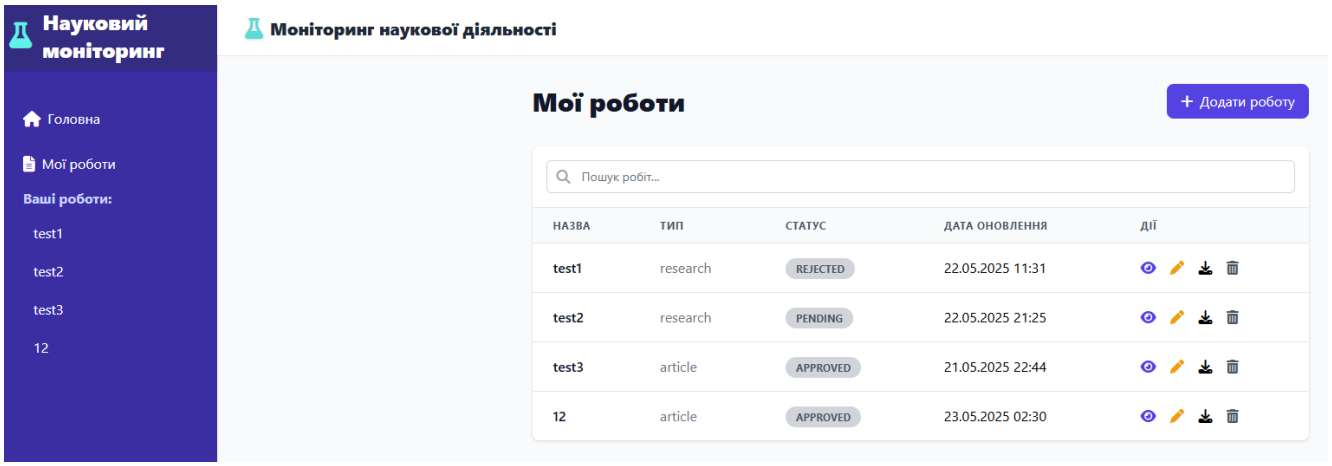


Рис. 4.7 Кабінет студента з переліком його наукових робіт.

На сторінці відображаються основні атрибути кожної роботи (назва, тип, статус, дата оновлення), а також кнопки для перегляду, редагування, завантаження чи видалення роботи.

4. Адмін панель:

На сторінці відображаються основні показники системи: кількість користувачів, кількість робіт на перевірці та кількість нових коментарів. Нижче представлений журнал останніх дій студентів та викладачів для контролю активності.

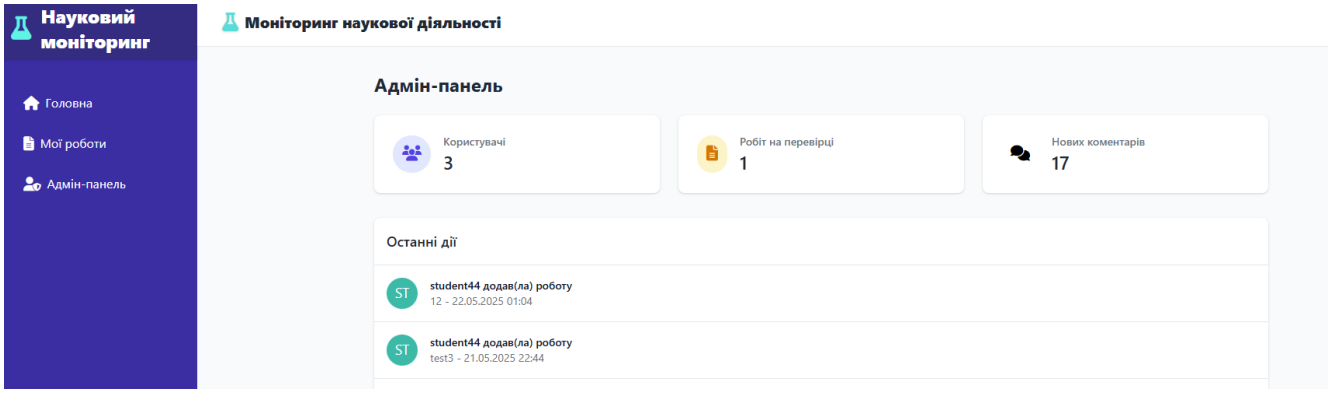


Рис. 4.8 Адміністраторська панель моніторингу наукової діяльності студентів.

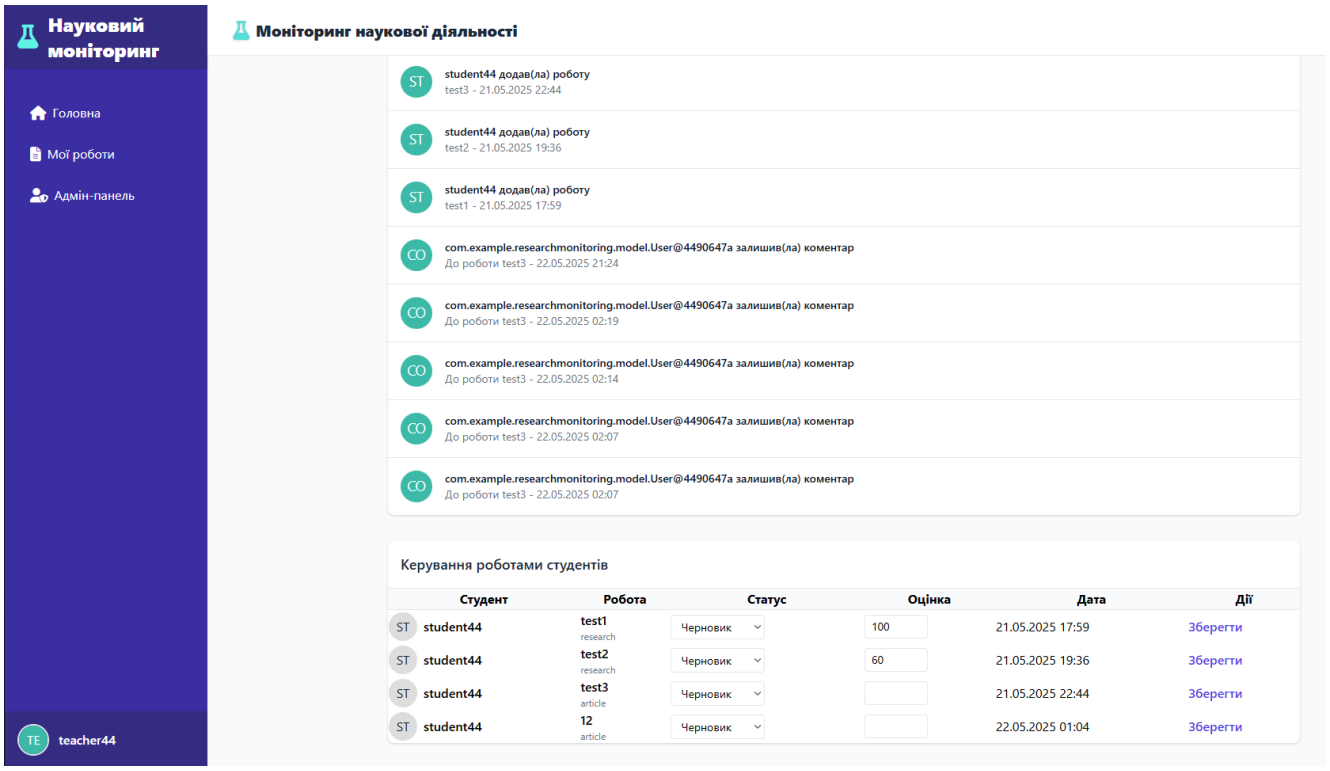


Рис. 4.9 Модуль керування роботами студентів у адмін-панелі.

Адміністратор або викладач може переглядати роботи всіх студентів, змінювати їхній статус, виставляти оцінки та зберігати результати.

5. Форма коментування та редагування роботи:

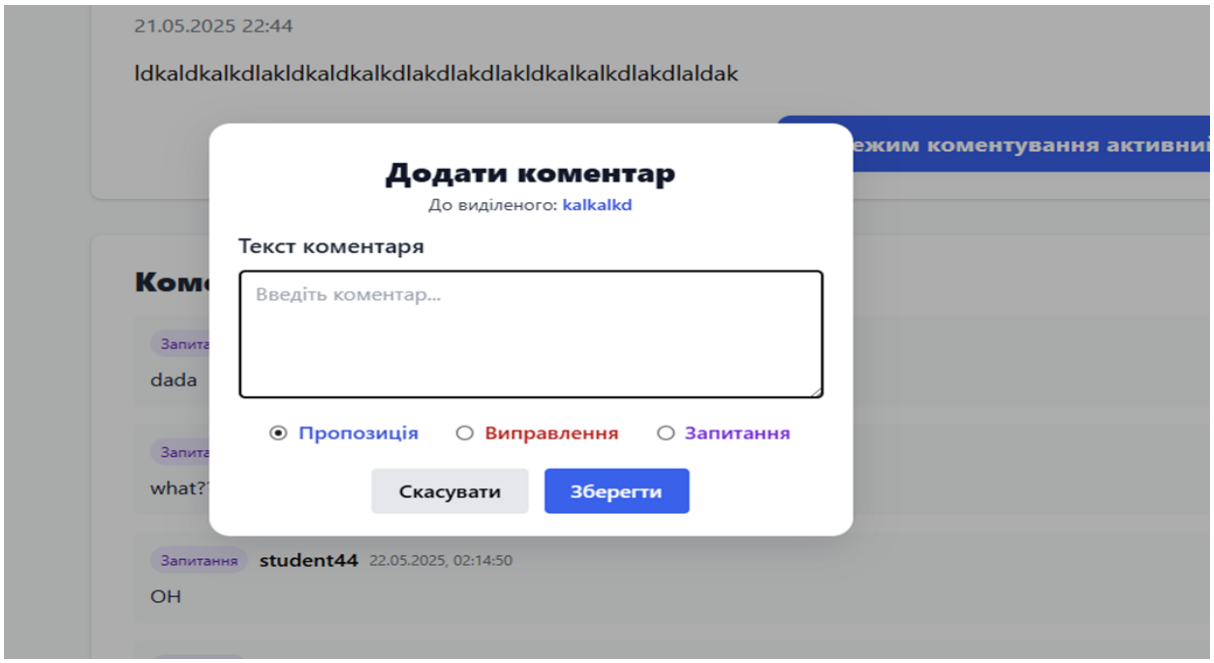


Рис. 4.10 Вікно додавання коментаря до фрагмента наукової роботи.

Користувач може ввести текст коментаря, обрати тип коментаря (пропозиція, виправлення, запитання) та зберегти або скасувати дію.

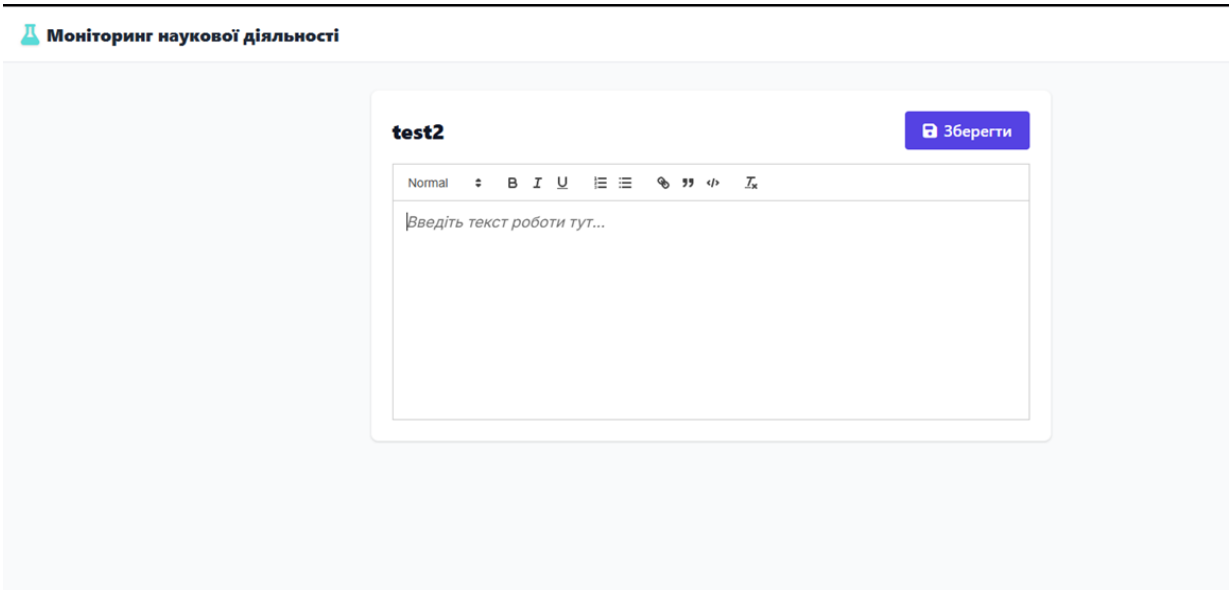


Рис. 4.11 Форма редагування тексту студентської роботи.

Передбачено інтуїтивний текстовий редактор з базовим форматуванням, що дозволяє студенту додавати або змінювати зміст своєї роботи перед подачею на перевірку.

Отже, ретельне опрацювання користувацького інтерфейсу — це один із головних чинників, який забезпечує просту адаптацію до платформи навіть для тих, хто не має глибоких технічних знань. У поєднанні з сучасним дизайном, адаптивністю та продуманими сценаріями, система пропонує не просто набір функцій, а повноцінне цифрове середовище, в якому кожен користувач почувається впевнено, захищено й ефективно реалізує свої навчальні завдання.

4.5 Функціонал коментування

Залишення коментарів до студентських робіт у системі моніторингу наукової діяльності — це не просто додатковий функціонал, а повноцінний інструмент для живої, гнучкої та змістовної взаємодії між учасниками навчального процесу. Його наявність дозволяє не лише обмінюватися думками в межах платформи, а й суттєво покращити якість освітнього супроводу та зменшити кількість “втраченої комунікації”, що часто трапляється в електронній пошті чи месенджерах.

Основні можливості коментування

Коментарі до роботи:

Кожен користувач, залежно від своєї ролі (студент чи викладач), може залишити коментар як до всього тексту роботи, так і до його окремого фрагмента. Для цього достатньо виділити потрібну частину тексту, після чого відкривається форма для введення коментаря — короткого текстового повідомлення із зазначенням контексту.

Вибір типу коментаря:

Система передбачає можливість обрати тип коментаря з-поміж трьох варіантів: “Пропозиція”, “Виправлення” або “Запитання”. Це дозволяє одразу зрозуміти інтонацію повідомлення, його ціль і бажаний зворотний зв’язок. Такий підхід не тільки структурує спілкування, а й підвищує ефективність взаєморозуміння між сторонами.

Відображення коментарів:

Усі коментарі до роботи виводяться в інтерфейсі у вигляді стрічки — у хронологічному порядку, з іменем автора, датою й точкою прив’язки до конкретного місця в тексті (якщо воно було вказане). Це значно спрощує перегляд і дає можливість швидко зорієнтуватися, які частини роботи потребують уваги.

Редагування та видалення:

Кожен коментар може бути змінений або видалений його автором — наприклад, якщо є потреба уточнити формулювання чи прибрати застаріле зауваження. Це створює динамічний і “живий” канал зворотного зв’язку, який не прив’язаний до статичних документів.

Особливості реалізації

Уся логіка коментування реалізована через прив’язку до конкретної роботи, збереження в базі даних усіх атрибутів: автора, дати, типу, тексту. Це забезпечує повну історію змін і можливість подальшого аналізу.

Інтерфейс форми коментування побудований максимально просто: є текстове поле, вибір типу коментаря й кнопки для збереження або скасування дії. Це дає змогу швидко залишати зауваження без зайвих кроків чи перемикань.

Наразі в системі не реалізовано автоматичне надсилання сповіщень про нові коментарі або їхнє сортування, що, втім, зберігає простоту реалізації та концентрує увагу на безпосередньому процесі взаємодії в межах перегляду роботи.

Типовий користувацький сценарій

1. Користувач відкриває наукову роботу, над якою хоче залишити відгук.
2. Виділяє текст або переходить до загального блоку коментарів.
3. Заповнює коротку форму, обирає тип повідомлення, натискає “Зберегти”.
4. Коментар одразу стає видимим у стрічці, доступний для редагування та спільного перегляду.

Інтерфейс додавання коментаря та відображення списку коментарів до роботи представлено на рисунку 4.10 у підрозділі 4.4.

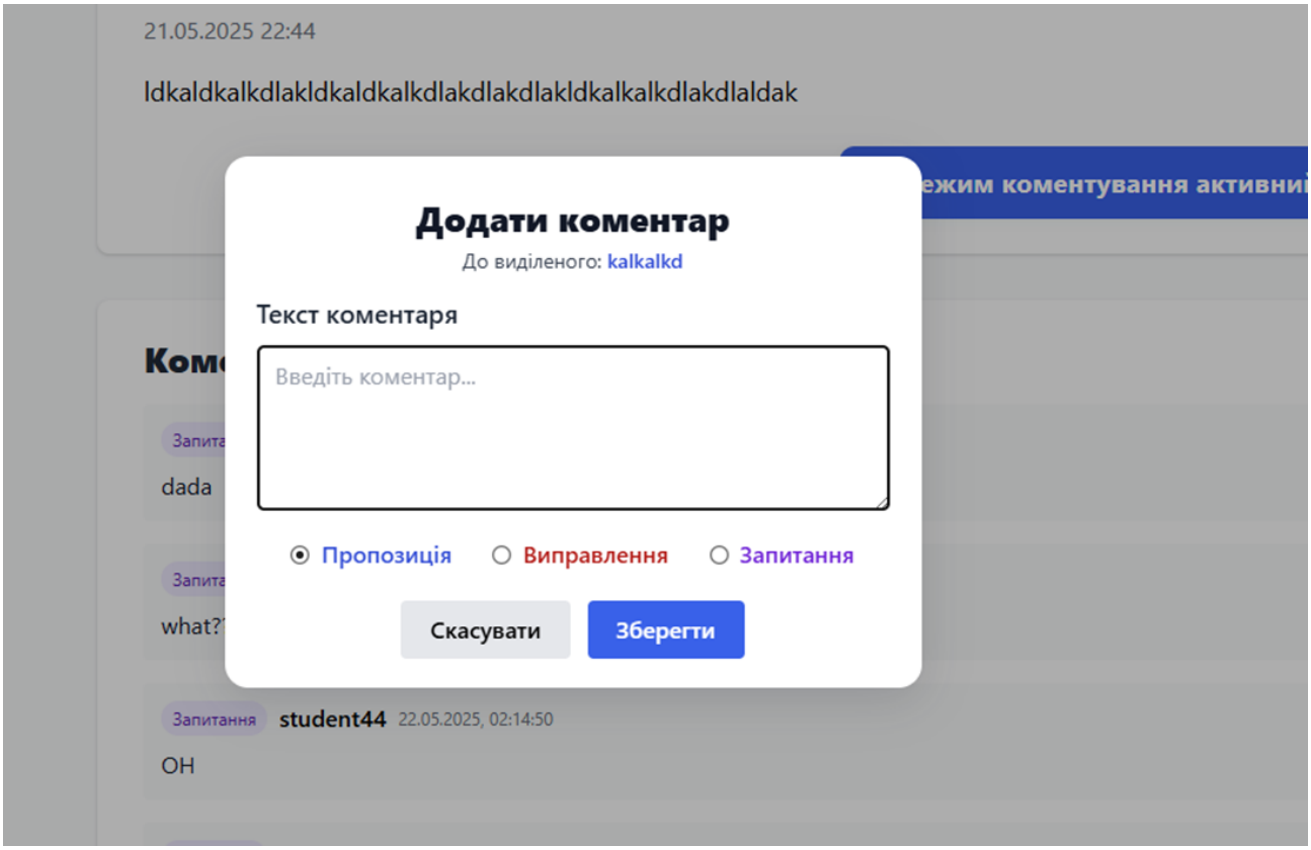


Рис. 4.10 Вікно додавання коментаря до фрагмента наукової роботи.

Такий підхід забезпечує зручність для користувача й дозволяє швидко орієнтуватися у контексті зауважень.

Таким чином, функціонал коментування працює не як пасивна примітка, а як інтерактивний канал зворотного зв’язку. Він формує основу для якісного діалогу,

дозволяє студенту оперативно враховувати зауваження викладача, а викладачу — систематично відслідковувати зміни, не втрачаючи загального контексту роботи. Усе це робить процес підготовки наукових проектів більш прозорим, контрольованим і взаємозрозумілим.

4.6 Тестування

Якість будь-якої сучасної інформаційної системи значною мірою залежить не лише від архітектури чи набору функцій, а й від того, наскільки ретельно вона була протестована ще до запуску. У випадку з платформою для моніторингу наукової діяльності студентів питання тестування відіграє критично важливу роль, оскільки система має бути не просто функціональною, а й надійною, передбачуваною та безпечною для великої кількості користувачів — як студентів, так і викладачів.

Основною метою тестування розробленої платформи стало гарантування її коректної роботи в реальних умовах. Це включало не лише перевірку працездатності окремих функцій, але й оцінку цілісності логіки системи, її здатності витримувати навантаження та надійно захищати персональні дані користувачів.

Тестування проводилося за двома напрямами:

- Ручне тестування (manual), яке дозволяє “очима користувача” пройти ключові сценарії, відчути інтерфейс, оцінити логіку взаємодії.

- Автоматизоване тестування (auto), що дало змогу охопити ширший спектр ситуацій, особливо на рівні API, логіки та безпеки.

Перейлемо до ручного тестування. Цей етап дозволив переконатися, що платформа працює логічно, зручно й без збоїв у стандартних сценаріях. Найбільша увага приділялася тим етапам, які є критичними для кінцевого користувача:

Реєстрація та авторизація

Перевірено створення облікових записів з різними ролями (студент, викладач), успішний і неуспішний вхід, обробку помилок при введенні неправильних даних, повторні спроби входу тощо.

Робота з науковими роботами

Протестовано створення нових робіт, оновлення існуючих, зміну статусу (“чернетка”, “на перевірці”, “доопрацювання”, “затверджено”), а також видалення робіт. Перевірено, як система реагує на дії користувача і чи правильно відображається кожен етап.

Коментування

Особливу увагу приділено коментуванню: перевірено можливість залишати коментарі до певних фрагментів тексту, редагувати або видаляти їх, вибирати тип зауваження (запитання, пропозиція, виправлення) та запобігати додаванню порожніх коментарів.

Інтерфейс

Виконано огляд усіх ключових сторінок: головна, особистий кабінет, список робіт, сторінка перегляду коментарів, адміністративна панель. Окремо протестова но адаптивність інтерфейсу на різних екранах і коректне відображення повідомле нь, статусів та індикаторів.

Адміністрування

Перевірено доступ адміністратора до аналітики, управління користувачами, перегляду статистики та взаємодії з системними даними. Особливу увагу приділено стабільності роботи панелі адміністратора під час одночасного доступу кількох користувачів.

На таблиці 4.2 зображено приклад тестового сценарію у застосунку

Таблиця 4.2

Приклад тестового сценарію

№	Дія	Вхідні дані	Очікуваний результат
1	Реєстрація нового користувача	login: student01	Обліковий запис створено, перехід на сторінку входу
2	Додавання нової роботи	Назва:“Тема 1”	Роботу додано, відображається у списку
3	Додавання коментаря до фрагмента	Текст: “Ок!”	Коментар з’являється у відповідному блоці
4	Видалення роботи	test2	Робота зникає зі списку, з’являється повідомлення про успішне видалення

А тепер перейдемо до автоматизованого тестування. Для забезпечення надійності ядра системи — її API, логіки та безпеки — було впроваджено автоматизоване тестування із застосуванням інструментів JUnit (для серверної частини), а також можливістю smoke-тестів через Swagger або Postman.

Тестування контролерів

Автоматично перевірялися відповіді на запити до REST API: чи правильний статус (200 OK, 404, 401 тощо), чи коректні тіла відповідей, як обробляються помилки та невалідні запити.

Тестування сервісів

Тести охопили бізнес-логіку зміни статусів, перевірку доступу до функцій, логіку створення, оновлення та видалення коментарів, унікальність логінів під час реєстрації.

Перевірка безпеки

Виконувались запити до захищених ендпоінтів без авторизації — система коректно блокувала такі дії. Також перевірялась правильність обмеження прав доступу залежно від ролі користувача.

Рисунок 4.13 підтверджує, що бек чує запит и додає його у базу

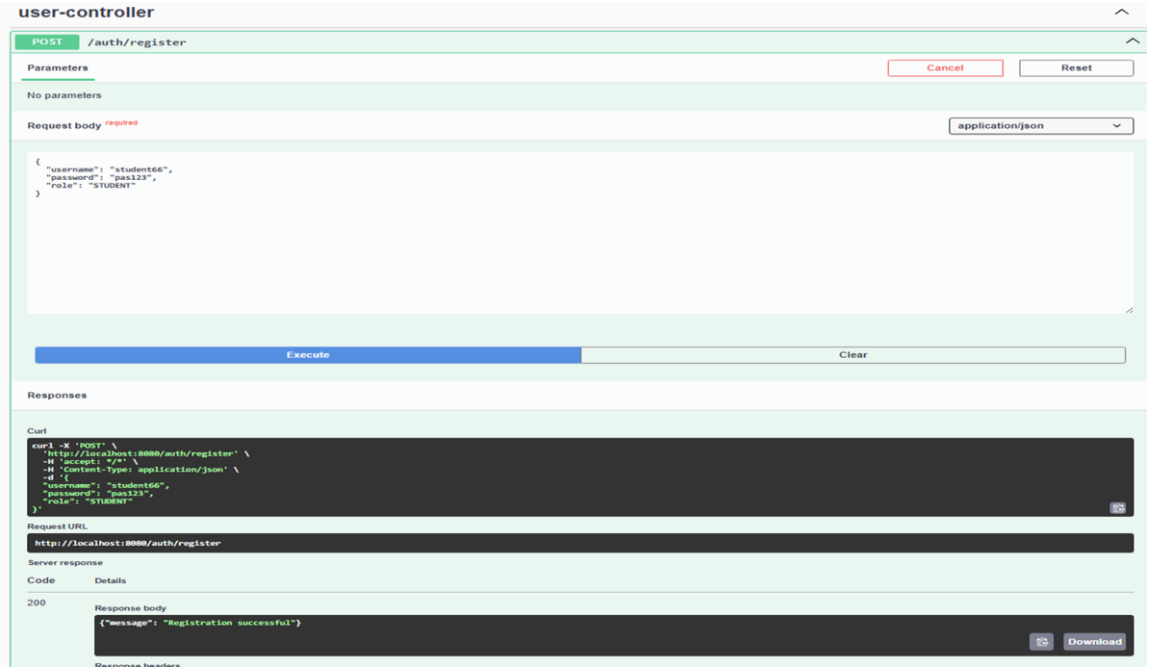


Рис. 4.13 Успішна реєстрація у Swagger UI.

Таким чином, тестування стало не просто етапом перевірки, а повноцінним елементом розробки, який забезпечив упевненість у тому, що всі заплановані функції реалізовані правильно, система не має критичних вразливостей, а користувачі можуть зручно, безпечно і стабільно працювати з платформою щодня. Своєчасне виявлення проблем дозволило не лише уникнути помилок у реальному використанні, а й зробити інтерфейс та функціонал більш зручними й логічними.

Висновок до розділу 4

У четвертому розділі дипломної роботи детально розглянуто процес практичної реалізації веб-платформи для моніторингу наукової діяльності студентів. Це саме той етап, де попередньо визначені концепції, вимоги й архітектурні підходи починають втілюватися в конкретні технічні рішення, логіку й інтерфейси. Завдяки вже сформованому технологічному підґрунтю, розробка системи стала не лише послідовною, а й технічно обґрунтованою.

Перш за все, була розроблена багаторівнева архітектура, яка дозволила чітко розмежувати відповідальності між клієнтською, серверною та базовою логікою, а також передбачити можливість розширення системи без її суттєвої перебудови.

Такий підхід не тільки оптимізує процес розробки, а й забезпечує зручність у супроводі й оновленні коду в майбутньому.

Важливим кроком стало моделювання даних: визначено ключові сутності, між якими встановлено логічні зв’язки, створено відповідні моделі та DTO для безпечної й контрольованої передачі інформації. Завдяки цьому вдалося забезпечити надійність зберігання, уникнути дублювання та налагодити ефективну взаємодію між компонентами платформи.

У межах розділу розглянуто й реалізацію ключових контролерів, які відповідають за основні функції — від автентифікації користувачів до керування роботами, коментування й адміністративного моніторингу. Кожен контролер реалізує логіку в рамках своєї області відповідальності, що дозволяє легко масштабувати систему й у разі потреби — оновлювати лише окремі модулі.

Особливий акцент було зроблено на створенні користувацького інтерфейсу. Важливо було не просто реалізувати всі необхідні функції, а зробити їх максимально зрозумілими, доступними й адаптованими під реальні сценарії взаємодії. Інтерфейс орієнтований на користувача: незалежно від того, студент він чи викладач, робота з платформою не викликає труднощів навіть без попереднього досвіду.

Окремим блоком розглянуто реалізацію функціоналу коментування — одного з найважливіших елементів системи, який покликаний налагодити продуктивну комунікацію між студентами та викладачами. Завдяки можливості залишати зауваження до окремих фрагментів роботи, обирати тип коментаря й редагувати повідомлення, платформа підтримує живий зворотний зв’язок і сприяє формуванню довіри та взаєморозуміння в навчальному процесі.

Не залишено без уваги й тестування: ручні перевірки сценаріїв взаємодії, автоматизовані тести для бізнес-логіки та API, а також перевірка безпеки дали змогу виявити потенційні вразливості ще до запуску платформи. Це дозволило забезпечити не лише стабільну, а й безпечну роботу всієї системи.

Завершальним етапом стало розгортання системи на сервері, налаштування її компонентів і підготовка до використання в реальному навчальному середовищі.

Розроблена документація дає змогу адміністраторам легко керувати платформою, здійснювати оновлення, резервне копіювання та підтримку користувачів.

У результаті виконаної роботи було не лише реалізовано всі функціональні вимоги, а й створено гнучку, масштабовану основу для подальшого розвитку. Платформа демонструє готовність до інтеграції з іншими освітніми сервісами, адаптації під нові сценарії використання та вдосконалення відповідно до потреб конкретного закладу освіти.

ВИСНОВКИ

Завершення дипломної роботи стало результатом глибокого занурення в актуальні виклики, пов’язані з організацією, підтримкою та моніторингом наукової діяльності студентів у цифровому середовищі. У межах дослідження вдалося не лише окреслити проблемні зони, але й запропонувати цілісне технічне рішення, що охоплює всі ключові аспекти — від ідеї та моделі до готового функціонального продукту.

Першим кроком стала аналітична частина, у межах якої було вивчено особливості супроводу студентських робіт у сучасній вищій освіті, виявлено труднощі зворотного зв’язку, контролю дедлайнів та забезпечення прозорості оцінювання. Зібрані висновки дозволили сформувати вимоги до майбутньої платформи та краще зрозуміти очікування майбутніх користувачів.

Особливу цінність мало вивчення вже наявних рішень — таких як Google Classroom, Notion, Overleaf, Miro. Порівняльний аналіз показав, що хоча кожна з цих платформ має свої переваги, жодна з них повною мірою не відповідає специфіці моніторингу індивідуальних студентських проєктів. Саме цей розрив і став підґрунтям для створення власної системи.

Було проведено формалізацію всіх необхідних елементів: визначено ролі користувачів, окреслено функціональні та нефункціональні вимоги, побудовано діаграми варіантів використання, структури даних та моделі сценаріїв роботи. Це дало змогу чітко вибудувати логіку системи та забезпечити її відкритість до подальших змін.

З технологічної точки зору реалізовано багаторівневу архітектуру, яка поєднує сучасні рішення у сфері веброзробки: фронтенд реалізовано за допомогою HTML5, CSS3, Tailwind, JavaScript і шаблонізатора Thymeleaf; серверна частина побудована на базі Java з використанням Spring Boot, Spring Security, JPA, а для зберігання даних обрано MySQL.

Особлива увага приділялася моделюванню даних. Створено зручну та гнучку систему сутностей, де враховано всі ключові аспекти: користувачі з різними ролями, роботи з версіями та статусами, система коментарів, історія змін. Для безпечної взаємодії між шарами було впроваджено DTO, що дозволяє ефективно працювати з даними без порушення принципів безпеки.

Логіка платформи реалізована через набір контролерів, які чітко розмежовують відповідальність і забезпечують обробку основних сценаріїв: авторизація, реєстрація, подання роботи, коментування, змінення статусу, адміністрування. Такий підхід дозволив досягти стабільності, гнучкості та масштабованості всієї системи.

Завдяки сучасному адаптивному дизайну інтерфейс платформи виявився доступним навіть для користувачів без досвіду роботи з подібними системами. Навігація логічна, інтерфейс інтуїтивний, а можливості налаштування — гнучкі. Реалізовано сценарії взаємодії для обох категорій користувачів — студентів і викладачів, з урахуванням їхніх типових потреб.

Окремим важливим напрямом стало впровадження функціоналу коментування. Користувачі можуть залишати точкові коментарі до фрагментів тексту, вказувати тип зауваження, редагувати та видаляти повідомлення. Це не лише спрощує зворотний зв’язок, а й перетворює взаємодію в системі на повноцінний освітній діалог.

На етапі тестування проведено перевірку всіх основних сценаріїв як вручну, так і автоматизовано. Це дозволило не тільки виявити помилки ще до впровадження системи, а й забезпечити її стабільну роботу в умовах реального навантаження.

Фінальним етапом стала підготовка до розгортання: описано процес установки, налаштування, резервного копіювання та оновлення системи. Додатково визначено можливі напрямки розвитку — інтеграція з LMS, підтримка мобільних додатків, розширення аналітики, автоматизація оцінювання.

У підсумку, реалізована система є повноцінним рішенням для організації цифрового моніторингу студентських наукових робіт у ЗВО. Вона об’єднує

простоту використання, широкий функціонал, гнучкість налаштувань та сучасний інтерфейс. Система готова до масштабування, інтеграції з іншими освітніми середовищами та подальшого розвитку, відкриваючи нові можливості для цифрової трансформації освіти.

Робота пройшла апробацію. За її результатами було опубліковано наступні тези доповідей:

1. Садлівський В.Р., Задонцев Ю.В. Веб-розробка застосунку для моніторингу наукової діяльності студентів. Матеріали: VI Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в ІКТ», 24.04.25, ДУІКТ, Київ, Україна, с.206-207.

2. Садлівський В.Р., Задонцев Ю.В. Забезпечення інформаційної безпеки у програмному забезпеченні для моніторингу наукової роботи студентів. Матеріали: XII Всеукраїнської науково-практичної конференції молодих учених «Інформаційні технології 2025», 15.05.25, КСУ імені Бориса Грінченка, Київ, Україна, с.322.

ПЕРЕЛІК ПОСИЛАНЬ

1. Інформаційні моделюючі технології, системи та комплекси [Електронний ресурс]. URL: https://er.chdtu.edu.ua/bitstream/ChSTU/4873/1/Book_IMTCK_2024.pdf#page=133 (дата звернення: 10.06.2024).

2. Google Classroom [Електронний ресурс]. URL: https://edu.google.com/workspace-for-education/products/classroom/ (дата звернення: 10.06.2024).

3. Notion [Електронний ресурс]. URL: https://www.notion.com/ (дата звернення: 10.06.2024).

4. OverLeaf [Електронний ресурс]. URL: https://www.overleaf.com/project (дата звернення: 10.06.2024).

5. Miro [Електронний ресурс]. URL: https://miro.com/app/dashboard/ (дата звернення: 10.06.2024).

6. Java [Електронний ресурс]. URL: https://www.java.com/en/ (дата звернення: 10.06.2024).

7. Spring Boot [Електронний ресурс]. URL: https://spring.io/projects/spring-boot (дата звернення: 10.06.2024).

8. Maven [Електронний ресурс]. URL: https://maven.apache.org (дата звернення: 10.06.2024).

9. MySQL [Електронний ресурс]. URL: https://dev.mysql.com/doc/ (дата звернення: 10.06.2024).

10. Spring Data JPA [Електронний ресурс]. URL: https://spring.io/projects/spring-data-jpa (дата звернення: 10.06.2024).

11. HTML5 [Електронний ресурс]. URL: https://html.spec.whatwg.org (дата звернення: 10.06.2024).

12. CSS3 [Електронний ресурс]. URL: https://www.w3.org/Style/CSS/ (дата звернення: 10.06.2024).

13. Tailwind CSS [Електронний ресурс]. URL: https://tailwindcss.com (дата звернення: 10.06.2024).

14. JavaScript [Електронний ресурс]. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (дата звернення: 10.06.2024).

15. Thymeleaf [Електронний ресурс]. URL: https://www.thymeleaf.org (дата звернення: 10.06.2024).

16. Spring Security [Електронний ресурс]. URL: https://docs.spring.io/spring-security/reference/ (дата звернення: 10.06.2024).

17. MVC (Model-View-Controller) [Електронний ресурс]. URL: https://martinfowler.com/eaaDev/uiArchs.html (дата звернення: 10.06.2024).

18. Freeman E., Robson E., Bates B., Sierra K. Head First Design Patterns. 2nd ed. O'Reilly Media, 2021. 670 p.

19. Blaha M., Rumbaugh J. Object-Oriented Modeling and Design with UML. 3rd ed. Pearson, 2021. 576 p.

20. Myers G.J., Sandler C., Badgett T. The Art of Software Testing. 4th ed. Wiley, 2024. 384 p.

21. Sommerville I. Software Engineering. 11th ed. Pearson, 2020. 792 p.

22. Pressman R.S., Maxim B.R. Software Engineering: A Practitioner's Approach. 9th ed. McGraw-Hill, 2020. 936 p.

23. Larman C. Applying UML and Patterns. 4th ed. Pearson, 2023. 736 p.

24. Nielsen J. Usability Engineering. Morgan Kaufmann, 2020. 362 p.

25. Bass L., Clements P., Kazman R. Software Architecture in Practice. 4th ed. Addison-Wesley, 2021. 624 p.

26. Fowler M. Refactoring: Improving the Design of Existing Code. 2nd ed. Addison-Wesley, 2021. 448 p.

27. Newman S. Building Microservices. 2nd ed. O'Reilly Media, 2021. 616 p.

28. Sadalage P.J., Fowler M. NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. 2nd ed. Addison-Wesley, 2022. 230 p.

29. Pautasso C., Alonso G. RESTful Web Services: Principles, Patterns, Emerging Technologies. Springer, 2021. 400 p.

30. Cohn M. Succeeding with Agile: Software Development Using Scrum. Addison-Wesley, 2022. 304 p.

31. Li J., Wang S., Wang Q., Chen X. Java Web Development Illuminated. Jones & Bartlett Learning, 2023. 588 p.

32. Tavares R. Spring Security in Action. Manning Publications, 2020. 384 p.

33. Diogo F. Hands-On RESTful API Design Patterns and Best Practices. Packt Publishing, 2021. 344 p.

34. Okoli Ch., Schabram K. Designing a Research Project: A Guide for Computer Science Students. Springer, 2023. 250 p.

35. Kurniawan B. Java 17 for Absolute Beginners. Apress, 2022. 789 p.

36. Bower J.L., Foster R.N. Agile Methodologies in Software Engineering. Routledge, 2021. 280 p.

37. Pal A., Mohapatra D.P. Automated Software Testing with Selenium. CRC Press, 2022. 350 p.

38. Shahriari H.R., Georgiadis C.K. Modern Database Management. 14th ed. Pearson, 2023. 528 p.

39. Evans E., Vernon V. Domain-Driven Design Distilled. 2nd ed. Addison-Wesley, 2022. 192 p.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка програмного забезпечення для моніторингу наукової роботи студентів

Виконав студент 4 курсу
групи ПД-44
Садлівський Владислав Русланович
Керівник роботи
к.т.н, доц, доцент кафедри ІПЗ Задонцев Юрій Вікторович

Київ – 2025

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета дослідження: вдосконалення процесу моніторингу наукової діяльності студентів.

Об’єкт дослідження: процес організації та цифрового моніторингу наукової діяльності студентів у закладах вищої освіти.

Предмет дослідження: веб-застосунок для реєстрації, перегляду та оцінювання наукових робіт студентів із підтримкою ролей користувачів.

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Провести аналіз предметної галузі та існуючих програмних рішень для обліку та моніторингу наукової діяльності студентів та оцінки їхньої функціональності.
2. Сформулювати вимоги до розроблюваного веб-застосунку на основі аналізу предметної галузі та існуючих програмних рішень.
3. Дослідити та обрати сучасні технології та інструменти, які можуть бути використані для створення веб-застосунку.
4. Провести моделювання та розробити архітектуру веб-застосунку для моніторингу наукової діяльності студентів.
5. Розробити веб-застосунок для моніторингу наукової діяльності студентів.
6. Провести тестування веб-застосунку.

3

АНАЛІЗ АНАЛОГІВ

Критерій	Google Classroom	Notion	Overleaf	Miro	RAT
Підтримка ролей	-	-	+	-	+
Коментування фрагментів роботи	-	+	+	-	+
Структурованість даних	+	+	+	-	+
Контроль версій / історія змін	-	+	+	-	+
Етапність виконання робіт	-	-	-	-	+
Публікація / підтвердження етапів	-	-	-	-	+
Інформаційна безпека / доступ	+	-	+	-	+
Простота використання	+	+	-	+	+

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:

1. Реєстрація та аутентифікація користувачів.
2. Створення, редагування та видалення наукових робіт.
3. Перегляд та коментування робіт науковим керівником.
4. Розмежування ролей.
5. Експорт звітів про наукову активність.

Нефункціональні вимоги:

1. Веб-доступність.
2. Документування API.
3. Захист даних (JWT).
4. Підтримка розширення функціоналу.
5. Підтримка резервного копіювання.

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

6

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ

7

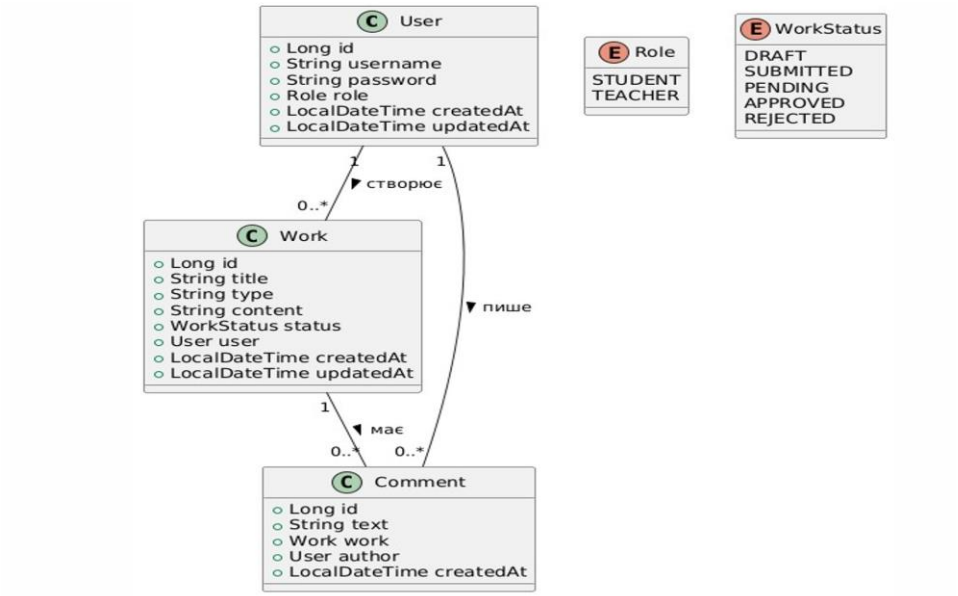
ДІАГРАМА ДІЯЛЬНОСТІ

Основний сценарій для студента: створення та відправка роботи

Основний сценарій для викладача: залишення коментаря до студентської роботи

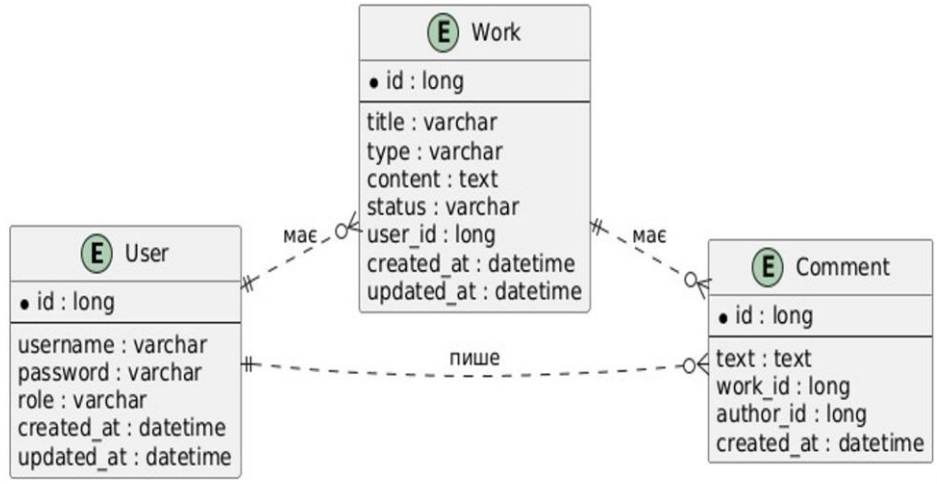
8

ДІАГРАМА КЛАСІВ



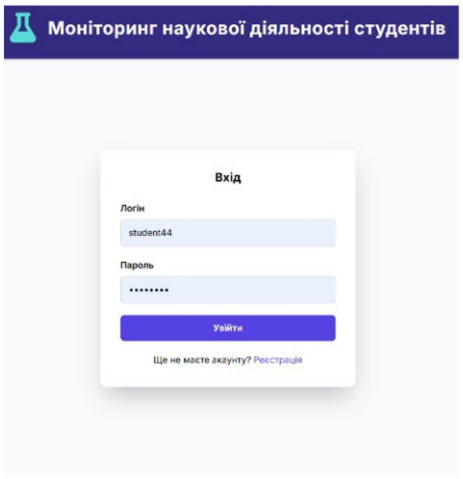
9

СХЕМА БАЗИ ДАНИХ

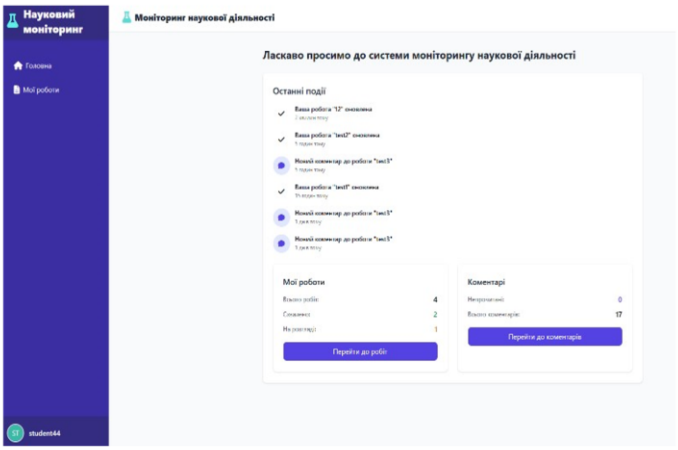


10

ЕКРАННІ ФОРМИ



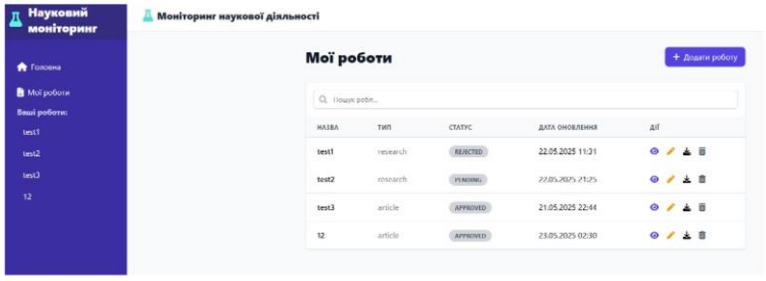
Сторінка логіну



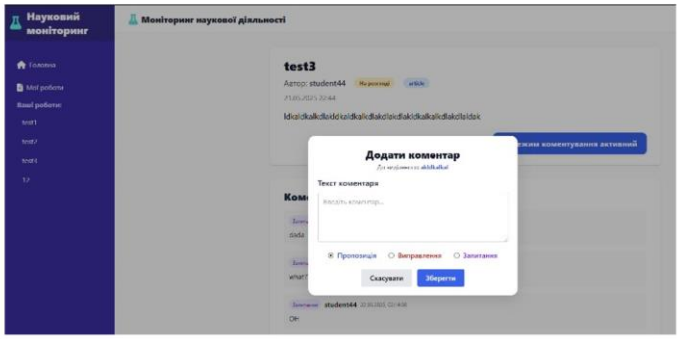
Головна сторінка

11

ЕКРАННІ ФОРМИ



Сторінка «Мої роботи»



Сторінка перегляду змісту роботи
та процес створення коментаря

12

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Садлівський В.Р., Задонцев Ю.В. Веб-розробка застосунку для моніторингу наукової діяльності студентів. Матеріали: Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в ІКТ», 24.04.25, ДУІКТ, Київ, Україна, с.206-207.

2. Садлівський В.Р., Задонцев Ю.В. Забезпечення інформаційної безпеки у програмному забезпеченні для моніторингу наукової роботи студентів. Матеріали: XII Всеукраїнської науково-практичної конференції молодих учених «Інформаційні технології – 2025», 15.05.25, КСУ імені Бориса Грінченка, Київ, Україна, с.322.

13

ВИСНОВКИ

1. Проведено аналіз предметної галузі та існуючих програмних рішень для обліку та моніторингу наукової діяльності студентів, в результаті визначено їх основні недоліки та неповноту функціоналу.
2. Сформульовано вимоги до розроблюваного, таким чином визначено функції для реалізації у застосунку та нефункціональні вимоги до нього.
3. Досліджено сучасні технології та інструменти для створення веб-застосунку, завдяки чому було визначено засоби його розробки.
4. Проведено моделювання та розроблено архітектуру веб-застосунку для моніторингу наукової діяльності студентів, з використанням обраних технологій та інструментів.
5. Реалізовано веб-застосунок для моніторингу наукової діяльності студентів за визначеними архітектурою та функціоналом.
6. Проведено тестування веб-застосунку, в результаті було виправлено виявлені помилки.

14

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

ResearchMonitoringApplication.java

```
@SpringBootApplication
public class ResearchMonitoringApplication {
    public static void main(String[] args) {

SpringApplication.run(ResearchMonitoringApplication.class, args);
    }
}
```

User.java

```
Package com.example.researchmonitoring.model;
import jakarta.persistence.*;
import org.hibernate.annotations.CreationTimestamp;
import org.hibernate.annotations.UpdateTimestamp;
import java.time.LocalDateTime;

@Entity
public class User{
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long id;
  @Column(nullable = false, unique = true)
  private String username;
  @Column(nullable = false)
  private String password;
  @Enumerated(EnumType.STRING)
  private Role role;
  @CreationTimestamp
  private LocalDateTime createdAt;
  @UpdateTimestamp
  private LocalDateTime updatedAt;
  public Long getId() { return id; }
  public void setId(Long id) { this.id = id; }
  public String getUsername() { return username; }
  public void setUsername(String username) { this.username = username; }
  public String getPassword() { return password; }
  public void setPassword(String password) { this.password = password; }
  public Role getRole() { return role; }
  public void setRole(Role role) { this.role = role; }
  public LocalDateTime getCreatedAt() { return createdAt; }
  public void setCreatedAt(LocalDateTime createdAt) { this.createdAt = createdAt; }
  public LocalDateTime getUpdatedAt() { return updatedAt; }
  public void setUpdatedAt(LocalDateTime updatedAt) { this.updatedAt = updatedAt; }
}
```

Comment.java

```
Package com.example.researchmonitoring.model;
import jakarta.persistence.*;
import org.hibernate.annotations.CreationTimestamp;

import java.time.LocalDateTime;
@Entity
public class Comment {
  @Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
```

```
    private Long id;
    @ManyToOne(fetch = FetchType.LAZY)
    private Work work;
    @ManyToOne(fetch = FetchType.LAZY)
    private User author;
    @Column(name = "text", columnDefinition = "TEXT")
    private String text;
    @Column(name = "is_read")
    private Boolean isRead = false;
    @CreationTimestamp
    @Column(name = "created_at", updatable = false)
    private LocalDateTime createdAt;
    public boolean isRead() {
        return isRead;
    }
    public void setRead(boolean read) {
        isRead = read;
    }
    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }
    public Work getWork() { return work; }
    public void setWork(Work work) { this.work = work;}
    public User getAuthor() { return author; }
    public void setAuthor(User author) { this.author = author; }

    public String getText() { return text; }
    public void setText(String text) { this.text = text; }
    public LocalDateTime getCreatedAt() { return createdAt; }
    public void setCreatedAt(LocalDateTime createdAt) { this.createdAt = createdAt; }
}
```

Role.java

```
package com.example.researchmonitoring.model;
public enum Role { STUDENT, TEACHER }
```

Work.java

```
package com.example.researchmonitoring.model;

import jakarta.persistence.*;
import org.hibernate.annotations.CreationTimestamp;
import org.hibernate.annotations.UpdateTimestamp;
import java.time.LocalDateTime;

@Entity
@Table(name = "work")
public class Work {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String title;

    @Column(columnDefinition = "TEXT")
    private String description;

    @Column(columnDefinition = "TEXT")
    private String content;
    @Enumerated(EnumType.STRING)
    private WorkStatus status = WorkStatus.DRAFT;
    private Integer grade;      // Оценка работы, если есть
    private String filePath;    // Ссылка на файл (если используется)

    @ManyToOne(fetch = FetchType.LAZY)
```

```
    @JoinColumn(name = "user_id") // важно! user_id вместо student_id, если у тебя в БД user_id
    private User user;
    @CreationTimestamp
    @Column(name = "created_at", updatable = false)
    private LocalDateTime createdAt;

    @UpdateTimestamp
    @Column(name = "updated_at")
    private LocalDateTime updatedAt;

    // --- Геттеры и сеттеры ---

    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }

    public String getTitle() { return title; }
    public void setTitle(String title) { this.title = title; }

    public String getDescription() { return description; }
    public void setDescription(String description) { this.description = description; }

    public String getContent() { return content; }
    public void setContent(String content) { this.content = content; }

    public String getType() { return type; }
    public void setType(String type) { this.type = type; }
    public WorkStatus getStatus() { return status; }
    public void setStatus(WorkStatus status) { this.status = status; }

    public Integer getGrade() { return grade; }
    public void setGrade(Integer grade) { this.grade = grade; }

    public String getFilePath() { return filePath; }
    public void setFilePath(String filePath) { this.filePath = filePath; }

    public User getUser() { return user; }
    public void setUser(User user) { this.user = user; }

    public LocalDateTime getCreatedAt() { return createdAt; }
    public void setCreatedAt(LocalDateTime createdAt) { this.createdAt = createdAt; }

    public LocalDateTime getUpdatedAt() { return updatedAt; }
    public void setUpdatedAt(LocalDateTime updatedAt) { this.updatedAt = updatedAt; }
}
```

WorkStatus.java

```
package com.example.researchmonitoring.model;
public enum WorkStatus {
    DRAFT,      // черновик – видит только студент
    SUBMITTED,  // отправлено научруку на проверку
    REJECTED,   // научрук вернул с ошибками
    APPROVED,   // работа принята и оценена
    PENDING
}
```