Державний торговельно-економічний університет
Кафедра інженерії програмного забезпечення та кібербезпеки

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Двонаправлена структура управління проєктом веборієнтованого програмного продукту дистанційної платформи навчання»

Студента 2 курсу, 4м групи, спеціальності 121 «Інженерія програмного забезпечення» освітньої програми «Управління проєктами програмних продуктів» _____ підпис студента — Тарасенка Владислава Олександровича

Гарант освітньої програми PhD, доцент кафедри інженерії програмного забезпечення та кібербезпеки _____ підпис гаранта — Десятко Альона Миколаївна

Гарант освітньої програми PhD, доцент кафедри інженерії програмного забезпечення та кібербезпеки _____ підпис гаранта — Десятко Альона Миколаївна

КИЇВ – 2025

Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення та кібербезпеки
Освітній ступінь магістр
Освітня програма «Управління проєктів програмних продуктів»

Затверджую
Зав. кафедри інженерії програмного забезпечення та кібербезпеки
Криворучко О.В.
«13» грудня 2024 р.

Завдання
на кваліфікаційну роботу студентові

Тарасенку Владиславу Олександровичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи «Двонаправлена структура управління проєктом веборієнтованого програмного продукту дистанційної платформи навчання»

Затверджена наказом ректора від «14» листопада 2024 р. № 3822

2. Строк здачі студентом закінченої роботи 14 листопада 2025

3. Цільова установка та вихідні дані до роботи

Мета роботи є комплексне дослідження, теоретичне обґрунтування та практична апробація двонаправленої структури управління проєктом через розробку прототипу веборієнтованого програмного продукту – платформи дистанційного навчання.

Об'єкт дослідження виступає процес управління розробкою складних веборієнтованих програмних продуктів в умовах сучасних ринкових викликів.

Предмет дослідження є безпосередньо двонаправлена структура управління, її ключові принципи, методологічні переваги та особливості застосування в процесі створення платформи дистанційного навчання.

4. Консультанти роботи із зазначенням розділів, які консультують:

Розділ	Консультант (прізвище, ініціали)	Підпис, дата	
		Завдання видав	Завдання прийняв

5. Зміст кваліфікаційної роботи (перелік питань за кожним розділом)

ВСТУП

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ УПРАВЛІННЯ ПРОЄКТАМИ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1. Аналіз сучасних методологій управління IT-проєктами

1.2. Концепція двонаправленої структури управління проєктом

1.3. Специфіка розробки платформ дистанційного навчання

Висновки до Розділу 1

РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА ОБҐРУНТУВАННЯ ТЕХНОЛОГІЧНОГО СТЕКУ

2.1. Двофакторна структуризація проєкту веборієнтованого програмного продукту дистанційної платформи навчання

2.2. Обґрунтування вибору технологічного стеку

2.3. Організація процесу розробки згідно з двонаправленою моделлю

Висновки до Розділу 2

РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ

3.1. Налаштування середовища розробки та інфраструктури

3.2. Розробка ключових функціональних модулів

3.3. Демонстрація двонаправленого управління на прикладі розробки нової функціональності

3.4. Тестування та результати роботи

Висновки до Розділу 3

ВИСНОВКИ

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

ДОДАТКИ

6. Календарний план виконання роботи

№ пор.	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	
		за планом	фактично
1	2	3	4
1.	*Вибір теми кваліфікаційної роботи*	07.11.2024	07.11.2024
2.	*Розробка та затвердження завдання на роботу магістра (стац/заоч)*	13.12.2024	13.12.2024
3.	*Вступ та перелік літературних джерел*	22.02.2025	22.02.2025
4.	*Розробка технічного завдання*	14.03.2025	14.03.2025
5.	*Розділ 1. ТЕОРЕТИЧНІ ОСНОВИ УПРАВЛІННЯ ПРОЄКТАМИ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ*	10.04.2025	10.04.2025
6.	*Розділ 2. ПРОЄКТУВАННЯ ТА ОБҐРУНТУВАННЯ ТЕХНОЛОГІЧНОГО СТЕКУ*	23.05.2025	23.05.2025
7.	*Розділ 3 ПРАКТИЧНА РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ*	05.09.2025	05.09.2025
8.	*Розробка програми та методики тестування*	27.09.2025	27.09.2025
9.	*Написання наукової статті*	16.04.2025	16.04.2025
10.	*Керівництво користувача*	11.10.2025	11.10.2025
11.	*Висновки та пропозиції*	16.10.2025	16.10.2025
12.	*Здача кваліфікаційної роботи на кафедру (перша перевірка)*	18.10.2025	18.10.2025
13.	*Підготовка реферату та презентації доповіді*	28.10.2025	28.10.2025
14.	*Попередній захист кваліфікаційної роботи*	29.10.2025 – 31.10.2025	29.10.2025 – 31.10.2025
15.	*Здача зброшурованої кваліфікаційної роботи*	14.11.2025	14.11.2025
16.	*Зовнішнє рецензування кваліфікаційної роботи*	28.10.2025	28.10.2025
17.	*Підготовка до публічного захисту кваліфікаційної роботи*	За розкладом ЕК	

7. Дата видачі завдання «13» грудня 2024 р.

8. Науковий керівник кваліфікаційної роботи Десятко А.М.
(прізвище, ініціали, підпис)

9. Гарант освітньої програми Десятко А.М.
(прізвище, ініціали, підпис)

10. Завдання прийняв до виконання студент Тарасенко В.О.
(прізвище, ініціали, підпис)

11. Відгук керівника кваліфікаційної роботи

Науковий керівник кваліфікаційної роботи

_____

(підпис, дата)

Відмітка про попередній захист _____

(ПІБ, підпис, дата)

12. Висновок про кваліфікаційну роботу

Кваліфікаційна робота студента _____Тарасенко В.О._____

(прізвище, ініціали)

може бути допущена до захисту екзаменаційній комісії.

Гарант освітньої програми _____Десятко А.М._____

(прізвище, ініціали, підпис)

Завідувач кафедри _____Десятко А.М.____

(підпис, прізвище, ініціали)

«_____»_____20 ___ р.

АНОТАЦІЯ

Кваліфікаційна робота присвячена дослідженню, проєктуванню та практичній реалізації двонаправленої структури управління проєктом веборієнтованого програмного продукту дистанційної платформи навчання. У роботі проведено системний аналіз сучасних методологій управління IT-проєктами – каскадних, гнучких (Agile) та гібридних – із визначенням їх переваг і недоліків у контексті створення складних EdTech-рішень. Запропоновано концепцію двонаправленої моделі управління, яка поєднує стратегічне планування «зверху-вниз» (Top-Down) з емпіричним зворотним зв'язком «знизу-вгору» (Bottom-Up), забезпечуючи баланс між стабільністю і гнучкістю розробки. Кваліфікаційна робота містить 82 сторінки, 38 рисунків, 2 таблиці, перелік використаних джерел налічує 50 найменувань.

Розроблено концептуальну архітектуру та прототип платформи дистанційного навчання із використанням сучасного технологічного стеку (Node.js, NestJS, React, PostgreSQL, Docker, GitHub Actions). Реалізовано двофакторну структуризацію проєкту – за роботами (WBS) і за ролями команди (RACI-матриця) – що дозволило забезпечити прозорість процесів, узгодженість між стратегічними цілями та технічною реалізацією. Практична частина демонструє застосування двонаправленої моделі на прикладі розробки ключових модулів системи (аутентифікація користувачів, управління курсами, аналітика навчального прогресу).

Результати роботи мають науково-практичне значення для вдосконалення підходів до управління IT-проєктами та можуть бути використані для створення корпоративних і комерційних EdTech-продуктів, підвищення ефективності командної взаємодії та якості освітніх цифрових платформ.

Ключові слова: двонаправлена структура управління, Agile, веборієнтований програмний продукт, платформа дистанційного навчання, EdTech, Node.js / React / PostgreSQL

ABSTRACT

The qualification paper is devoted to the research, design, and practical implementation of a bidirectional project management structure for a web-based software product – a distance learning platform. The study provides a systematic analysis of modern IT project management methodologies – waterfall, agile, and hybrid – identifying their advantages and limitations in the context of developing complex EdTech solutions. The paper proposes the concept of a bidirectional management model, which combines top-down strategic planning with bottom-up empirical feedback, ensuring an effective balance between stability and flexibility in software development. The qualification paper contains 82 pages, 38 figures, 2 tables, and a list of references comprising 50 sources.

A conceptual architecture and prototype of the distance learning platform were developed using a modern technology stack (Node.js, NestJS, React, PostgreSQL, Docker, GitHub Actions). The project implements a two-factor structuring approach – by tasks (WBS) and by team roles (RACI matrix) – providing process transparency and alignment between strategic objectives and technical implementation. The practical part demonstrates the application of the bidirectional model through the development of key system modules, including user authentication, course management, and learning progress analytics.

The results of the work have scientific and practical significance for improving IT project management approaches and can be applied in the creation of corporate and commercial EdTech products, enhancing team collaboration efficiency and the quality of digital educational platforms.

Keywords: bidirectional management structure, Agile, web-based software product, distance learning platform, EdTech, Node.js / React / PostgreSQL

ЗМІСТ

ВСТУП

Сучасний етап розвитку інформаційного суспільства характеризується стрімкою цифровою трансформацією, яка докорінно змінює традиційні сфери людської діяльності, зокрема освітню галузь. Глобальні виклики останніх років, прискорення темпів життя та потреба у безперервному навчанні протягом усього життя зумовили експоненційне зростання попиту на онлайн-освіту. Це, у свою чергу, стимулювало активну розробку складних веборієнтованих програмних продуктів – платформ дистанційного навчання (EdTech). Проте створення таких комплексних систем є нетривіальним завданням, що висуває високі вимоги не лише до технічної компетенції команди, а й до ефективності процесів управління проєктом.

Актуальність обраної теми полягає у розв'язанні суперечності між динамічною природою сучасних ІТ-проєктів та обмеженнями класичних моделей управління. Традиційні підходи, такі як каскадна модель, демонструють свою неефективність в умовах високої невизначеності та постійної зміни вимог з боку замовників та кінцевих користувачів. Водночас, гнучкі (Agile) методології, хоча й забезпечують необхідну адаптивність на тактичному рівні, іноді можуть призводити до розфокусування стратегічних цілей у довгострокових та масштабних проєктах. Саме тому виникає нагальна необхідність у дослідженні та впровадженні гібридних, більш досконалих підходів. Одним з таких перспективних рішень є двонаправлена структура управління проєктом, що поєднує стратегічне планування «зверху-вниз» з оперативним зворотним зв'язком та ініціативами від команди розробки «знизу-вгору», створюючи синергетичний ефект.

Метою даної кваліфікаційної роботи є комплексне дослідження, теоретичне обґрунтування та практична апробація двонаправленої структури управління проєктом через розробку прототипу веборієнтованого програмного продукту – платформи дистанційного навчання.

Об’єктом дослідження виступає процес управління розробкою складних веборієнтованих програмних продуктів в умовах сучасних ринкових викликів.

Предметом дослідження є безпосередньо двонаправлена структура управління, її ключові принципи, методологічні переваги та особливості застосування в процесі створення платформи дистанційного навчання.

Для досягнення поставленої мети необхідно вирішити такі **завдання**:

1. провести системний аналіз існуючих методологій управління проєктами розробки програмного забезпечення, зокрема каскадних, гнучких (agile) та гібридних моделей, з метою виявлення їхніх переваг та недоліків у контексті створення складних edtech-продуктів;
2. сформулювати та теоретично обґрунтувати концепцію двонаправленої структури управління проєктом, визначивши її ключові принципи, інформаційні потоки («зверху-вниз» та «знизу-вгору») та організаційні ролі;
3. здійснити аналіз предметної області платформ дистанційного навчання, дослідити ринок існуючих рішень та на основі цього сформулювати деталізовані функціональні та нефункціональні вимоги до програмного продукту, що розробляється;
4. спроєктувати архітектуру веборієнтованого програмного продукту, обґрунтувавши вибір архітектурного стилю, та розробити логічну й фізичну структуру бази даних для зберігання інформації про курси, користувачів та навчальний прогрес;
5. здійснити порівняльний аналіз сучасних програмних засобів та обґрунтувати вибір технологічного стеку для реалізації клієнтської (front-end) та серверної (back-end) частин системи, а також системи управління базами даних;
6. розробити детальний план організації процесу розробки відповідно до принципів двонаправленої моделі управління, обравши та

налаштувавши відповідні інструменти для ведення завдань, контролю версій та командної взаємодії;

7. здійснити практичну реалізацію програмного прототипу платформи, розробивши ключові функціональні модулі, зокрема систему автентифікації користувачів, управління навчальним контентом та механізм проходження курсів;
8. продемонструвати ефективність застосування двонаправленої моделі на практичному кейсі розробки нової функціональності, детально описавши цикли постановки завдання, виконання, тестування та інтеграції зворотного зв'язку для коригування планів;
9. провести комплексне тестування розробленого прототипу з метою перевірки його працездатності та відповідності раніше сформульованим функціональним і нефункціональним вимогам.

Практична цінність отриманих результатів полягає в тому, що запропонована модель управління та розроблений на її основі прототип платформи можуть бути використані як методологічна та технологічна основа для створення реальних комерційних або корпоративних освітніх проєктів. Це дозволить підвищити ефективність процесу розробки, забезпечити кращу відповідність кінцевого продукту потребам ринку та гнучко реагувати на зміни, що є критично важливим для успіху в конкурентному середовищі цифрової освіти.

РОЗДІЛ 1.
ТЕОРЕТИЧНІ ОСНОВИ УПРАВЛІННЯ ПРОЄКТАМИ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1. Аналіз сучасних методологій управління IT-проєктами

Ефективність розробки сучасних інформаційних систем, зокрема веборієнтованих програмних продуктів, значною мірою залежить від обраної методології управління проєктом. Вибір підходу визначає не лише швидкість та вартість розробки, але й здатність кінцевого продукту відповідати динамічним вимогам ринку та очікуванням користувачів [31, 33]. В еволюції інженерії програмного забезпечення сформувалися два фундаментально відмінні парадигмальні підходи: класичний (предиктивний) та гнучкий (адаптивний), які згодом стали основою для створення численних гібридних моделей.

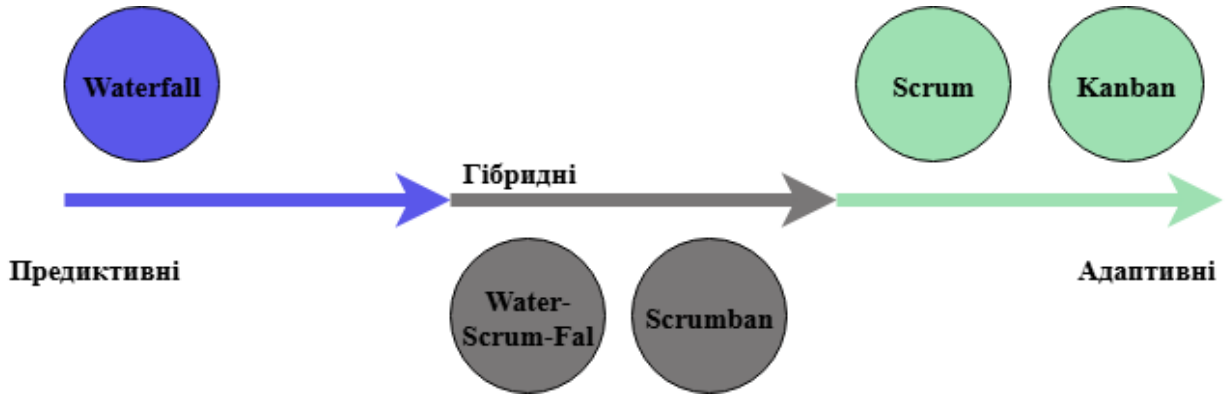


Рис. 1.1 – **Карта парадигм управління: предиктивна ↔ адаптивна**

Джерело: побудовано автором на основі [31, 33]

Класичний підхід до управління проєктами, що знайшов своє формалізоване вираження у стандарті PMBOK [1] та фундаментальних працях з інженерії програмного забезпечення [7], найчастіше асоціюється з каскадною моделлю (Waterfall). Дана модель характеризується суворою послідовністю етапів: аналіз вимог, проєктування, реалізація, тестування, впровадження та супровід. Перехід до наступного етапу можливий лише після повного

завершення попереднього, що створює лінійний та детермінований характер процесу розробки [16].

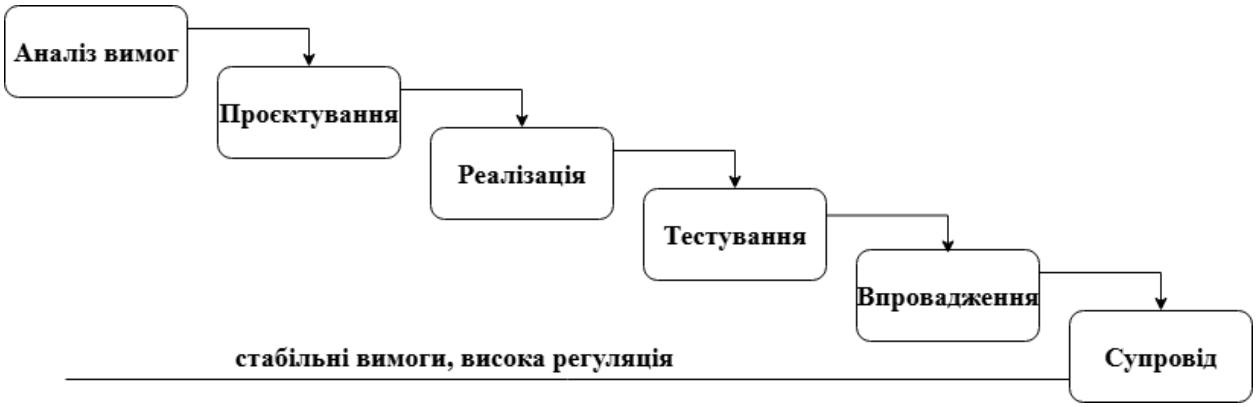


Рис. 1.2 – **Життєвий цикл каскадної моделі**

Джерело: побудовано автором на основі [1, 16]

Основною перевагою каскадної моделі є її передбачуваність, чіткість у плануванні та простота контролю. Завдяки ретельній документації на кожному етапі, проєкт стає легким для управління, а результати та терміни є чітко визначеними з самого початку [1]. Такий підхід є доцільним для проєктів, де вимоги є стабільними, добре зрозумілими та не зазнають змін протягом усього життєвого циклу. Це можуть бути державні замовлення, проєкти з високим рівнем регуляції або розробка систем, що інтегруються у вже існуючу, жорстко визначену інфраструктуру.

Однак у контексті розробки сучасних веборієнтованих продуктів, зокрема платформ дистанційного навчання, недоліки каскадної моделі стають критичними. Головним з них є відсутність гнучкості та неспроможність адаптуватися до змін вимог [7, 33]. Кінцевий користувач бачить продукт лише на завершальних стадіях, що створює високий ризик того, що розроблена система не відповідатиме реальним потребам. Будь-які зміни на пізніх етапах є надзвичайно дорогими та складними в реалізації. Таким чином, для інноваційних проєктів у конкурентних сферах, де швидкість виходу на ринок та здатність до адаптації є ключовими, каскадна модель демонструє свою неефективність.

На противагу жорсткості класичних моделей на початку 2000-х років сформувався рух Agile, заснований на принципах, викладених в «Маніфесті гнучкої розробки програмного забезпечення». Ця філософія ставить у пріоритет людей та взаємодію над процесами та інструментами, працюючий продукт над вичерпною документацією, співпрацю з замовником над обговоренням умов контракту та готовність до змін над слідуванням початковому плану [32]. Agile-підходи базуються на ітеративній та інкрементальній розробці, що дозволяє постачати цінність замовнику невеликими частинами, але регулярно, отримуючи швидкий зворотний зв'язок.

Найбільш поширеним фреймворком у рамках Agile є Scrum. Його процес базується на коротких, фіксованих за часом ітераціях, що називаються спринтами [2]. Робота організована навколо невеликої, самоорганізованої команди, яка включає Власника Продукту, Скрам-майстра та розробників. Ключовим елементом є емпіричний контроль процесу, що досягається через прозорість, інспекцію та адаптацію на регулярних подіях, таких як щоденні наради, огляд спринту та ретроспектива. Scrum є надзвичайно ефективним для проєктів зі складним продуктом та мінливими вимогами, оскільки дозволяє гнучко коригувати пріоритети та напрямок розробки після кожного спринту.

Іншою популярною методологією є Kanban, що походить з принципів ощадливого виробництва (Lean) [6]. На відміну від Scrum, Kanban не використовує часові ітерації. Його основна ідея полягає у візуалізації робочого процесу на дошці, обмеженні кількості одночасно виконуваної роботи (Work In Progress) та оптимізації потоку завдань. Це дозволяє команді зосередитись на швидкому завершенні поточних завдань та забезпечує безперервну поставку цінності. Kanban є ідеальним для проєктів з непередбачуваним потоком завдань, таких як технічна підтримка або безперервне вдосконалення існуючого продукту. У сучасній веброзробці Agile-методології стали стандартом де-факто, оскільки вони дозволяють створювати продукти в

умовах високої невизначеності, швидко реагувати на зміни ринку та забезпечувати високий рівень залученості замовника [3, 32].

Усвідомлення того, що ні суто класичний, ні суто гнучкий підходи не є універсальним рішенням для всіх типів проєктів, призвело до появи гібридних моделей. Їхня мета – поєднати найкращі риси обох парадигм: стратегічне планування та передбачуваність від каскадної моделі з тактичною гнучкістю та адаптивністю Agile [31]. Такий синергетичний ефект дозволяє великим організаціям зберігати довгострокове бачення та бюджетний контроль, водночас надаючи командам розробки автономію для ефективної роботи.

Одним із поширених прикладів є Water-Scrum-Fall, де початкові фази аналізу вимог та архітектурного проєктування виконуються в каскадному стилі. Потім етап розробки розбивається на серію Scrum-спринтів, що дозволяє ітеративно створювати та тестувати функціонал. Завершальні етапи, такі як фінальна інтеграція, розгортання та приймальне тестування, знову можуть відбуватися за лінійною схемою. Такий підхід є корисним у проєктах, де загальна концепція продукту визначена, але деталі реалізації потребують гнучкої розробки та валідації.

Іншою популярною гібридною моделлю є Scrumban, яка об'єднує елементи Scrum (ролі, регулярні зустрічі) з візуальним управлінням потоком Kanban. Команди можуть використовувати спринти для планування, але не обмежувати себе жорсткими рамками, дозволяючи додавати нові пріоритетні завдання в процес, якщо це необхідно. Це забезпечує більшу гнучкість, ніж у класичному Scrum, і є ефективним для команд, що працюють як над розробкою нового функціоналу, так і над підтримкою існуючого. Аналіз цих підходів свідчить, що для розробки складного продукту, як-от платформа дистанційного навчання, оптимальним може бути саме гібридний підхід, який дозволить поєднати довгострокове стратегічне бачення продукту з гнучкою тактикою його реалізації.

Таблиця 1.1

Порівняльна матриця методологій

Методологія	Waterfall (Каскадна)	Scrum	Kanban	Water-Scrum-Fall (гібрид)	Scrumban (гібрид)
Гнучкість (1–5)	1	5	4	3	4
Передбачуваність (1–5)	5	3	3	4	3
Швидкість інкрементів (1–5)	2	5	4	4	4
Рівень документування (1–5)	5	3	2	4	3
Залучення замовника (1–5)	2	5	4	4	4
Придатність до змінних вимог (1–5)	1	5	4	4	4
Ітераційність / тип потоку	Послідовні фази	Ітерації (спринти)	Безперервний потік, WIP-ліміти	Каскад + ітерації	Ітерації + потік
Рекомендовані сценарії	Стабільні вимоги, висока регуляція, інтеграція у жорстко визначене середовище	Складні продукти, мінливі вимоги, швидкий фідбек і пріоритезація	Підтримка, DevOps, потік інцидентів/заявок, безперервне вдосконалення	Великі організації: формальне планування + гнучка розробка та поетапний реліз	Команди зі змішаним потоком задач: новий функціонал і підтримка/експлуатація

Джерело: побудовано автором на основі [1, 3, 16, 32]

Матриця узагальнює сильні та слабкі сторони підходів за ключовими критеріями. Шкала 1–5: 1 – низький рівень ознаки, 5 – високий. Читати слід у два кроки: спершу обрати підхід за контекстом (тип потоку, сценарій застосування), потім звірити числові оцінки з обмеженнями проєкту (регуляція, потреба у швидких інкрементах, частота змін вимог). Гібриди поєднують передбачуваність планування з гнучкістю поставки.

1.2. Концепція двонаправленої структури управління проєктом

Аналіз обмежень класичних та суто гнучких методологій диктує необхідність формування більш інтегрованого підходу, здатного забезпечити одночасно стратегічну стійкість та тактичну адаптивність. Таким підходом є двонаправлена структура управління проєктом, яка являє собою організаційну

модель, що синтезує директивне планування «зверху-вниз» (Top-Down) з емпіричним зворотним зв'язком «знизу-вгору» (Bottom-Up). Ця модель створює замкнений контур безперервного вдосконалення, де стратегічні цілі бізнесу постійно перевіряються та коригуються на основі реальних даних, отриманих у процесі розробки та взаємодії з продуктом.

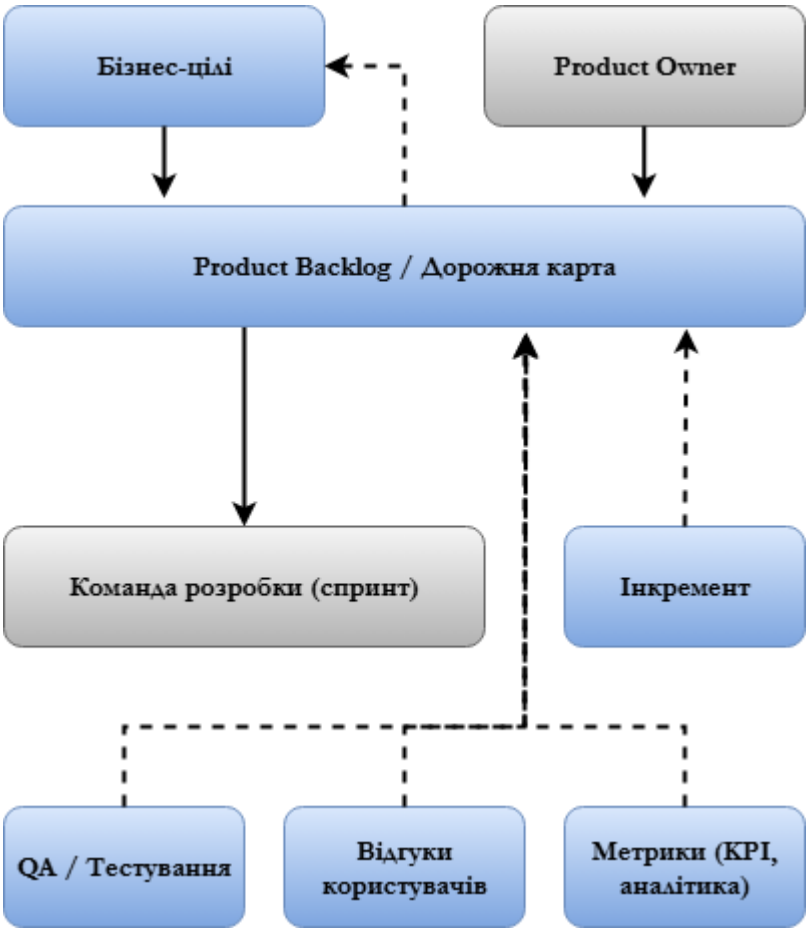


Рис. 1.3 – **Двонаправлена модель управління: контури Top-Down та Bottom-Up**

Джерело: побудовано автором

Двонаправлена структура управління проєктом – це гібридна управлінська модель, що поєднує довгострокове стратегічне планування, характерне для предиктивних підходів, з ітеративною розробкою та механізмами зворотного зв'язку, притаманними гнучким методологіям. Суть моделі полягає у створенні двох взаємопов'язаних та взаємозалежних

інформаційних потоків, що забезпечують постійну синхронізацію між баченням продукту на рівні стейкхолдерів та реаліями його технічної реалізації й ринкового сприйняття.

В основі цієї моделі лежать декілька ключових принципів. По-перше, принцип стратегічної спрямованості, згідно з яким будь-яка задача з розробки повинна мати чіткий зв'язок із вищою бізнес-ціллю, забезпечуючи осмисленість та пріоритетність роботи. По-друге, принцип емпіричного контролю, запозичений з філософії Agile [2], що вимагає приймати рішення на основі фактичних результатів, а не припущень. По-третє, принцип прозорості та спільної відповідальності, який передбачає, що як керівництво, так і команда розробки мають доступ до релевантної інформації та спільно несуть відповідальність за кінцевий результат. Разом ці принципи створюють середовище, де стратегія не є статичним документом, а живим організмом, що еволюціонує під впливом практичного досвіду.

Функціонування моделі забезпечується двома основними інформаційними потоками. Потік «зверху-вниз» (Top-Down) відповідає за каскадування стратегії та декомпозицію цілей. Він починається з визначення бізнес-цілей на рівні керівництва та стейкхолдерів (наприклад, підвищення рівня утримання студентів на 20%). Ці абстрактні цілі трансформуються у конкретні вимоги до продукту, які формулює власник продукту (Product Owner). На основі вимог створюється та пріоритезується бек лог продукту (Product Backlog) – впорядкований список функціоналу у вигляді історій користувачів (user stories). Нарешті, під час планування ітерацій команда розробки перетворює елементи беклогу на конкретні технічні завдання для виконання у спринті. Цей потік забезпечує ясність мети, контекст та напрямок руху для всієї команди.

У свою чергу, потік «знизу-вгору» (Bottom-Up) є механізмом зворотного зв'язку та валідації стратегії. Його відправною точкою є результати спринтів – працюючий інкремент продукту, який можна продемонструвати та протестувати. У процесі розробки команда ідентифікує технічні обмеження та

пропозиції щодо оптимізації, які можуть суттєво вплинути на вартість чи терміни реалізації певного функціоналу. Паралельно, через процеси забезпечення якості (QA) та демонстрації продукту зацікавленим сторонам, збираються відгуки користувачів та дані про реальну взаємодію з системою. Вся ця емпірична інформація – про технічну реалістичність, користувацький досвід та досягнутий прогрес – аналізується власником продукту та стейкхолдерами і використовується для коригування беклогу та стратегії. Таким чином, цей потік збагачує процес прийняття рішень реалістичними даними та інноваційними ідеями від команди, що безпосередньо створює продукт.

Рис. 1.4 – **Замкнений контур цінності**

Джерело: побудовано автором

При порівнянні двонаправленої моделі з класичними та гнучкими підходами стають очевидними її ключові відмінності. На відміну від каскадної моделі, яка є виключно «top-down» і не передбачає механізмів для адаптації після початкового планування [7], двонаправлена структура є динамічною та

ітеративною. Вона дозволяє вносити зміни на будь-якому етапі життєвого циклу, мінімізуючи ризик створення продукту, що не відповідає потребам ринку.

Водночас, у порівнянні з деякими реалізаціями Agile, які можуть страждати від надмірної тактичної зосередженості («фабрика фіч») та втрати стратегічного вектору, двонаправлена модель зберігає сильний стратегічний компонент. Потік «зверху-вниз» гарантує, що кожна ітерація та кожне завдання є кроком до досягнення чітко визначеної бізнес-цілі [31].

Саме тому головні переваги двонаправленої структури проявляються в довгострокових, складних проєктах зі змінними вимогами, якими і є розробка платформ дистанційного навчання. По-перше, вона забезпечує стійкість до невизначеності, дозволяючи проєкту еволюціонувати разом із ринком та технологіями. По-друге, вона підвищує залученість та мотивацію команди, оскільки їхній технічний досвід та пропозиції безпосередньо впливають на розвиток продукту. По-третє, вона оптимізує розподіл ресурсів, фокусуючи зусилля на тому функціоналі, який приносить найбільшу цінність згідно з отриманим зворотним зв'язком, що є ключовим принципом ощадливого підходу [3]. Таким чином, ця модель є ефективним інструментом для досягнення балансу між інноваційністю, якістю та відповідністю стратегічним цілям організації.

1.3. Специфіка розробки платформ дистанційного навчання

Розробка платформ дистанційного навчання є комплексним завданням, що виходить за межі суто технічної реалізації та охоплює педагогічні, організаційні та соціальні аспекти [21, 45]. На відміну від багатьох інших веборієнтованих систем, успіх EdTech-продукту визначається не лише його функціональними можливостями, а й здатністю створювати ефективне та мотивуюче навчальне середовище [22]. Це вимагає глибокого розуміння предметної області, потреб ключових користувачів та специфічних вимог до

якості, що накладає значний відбиток на весь процес проєктування та розробки.

Сучасний ринок платформ дистанційного навчання, або систем управління навчанням (LMS), є надзвичайно різноманітним та включає рішення, що відрізняються за архітектурою, моделлю розповсюдження та цільовою аудиторією [26]. Аналіз провідних представників ринку дозволяє ідентифікувати фундаментальні функціональні блоки, що є основою для будь-якої такої системи. Так, Moodle, як представник світу open-source, є потужною та гнучкою платформою, що надає широкі можливості для кастомізації та використовується переважно в академічних установах [35]. Її архітектура демонструє важливість модульності та розширюваності. На протилежному полюсі знаходиться Coursera – комерційна MOOC-платформа, що робить акцент на високоякісному користувацькому досвіді (UX), масштабованості для мільйонів користувачів та інтеграції відеоконтенту й автоматизованих систем оцінювання. Третім показовим прикладом є Google Classroom, який ілюструє силу екосистемної інтеграції та простоти; його успіх значною мірою зумовлений безшовною взаємодією з іншими сервісами Google та низьким порогом входження для викладачів та учнів. Попри відмінності, аналіз цих систем дозволяє виділити універсальний набір ключових функціональних блоків: модуль управління користувачами та ролями; модуль створення та адміністрування курсів; підсистема доставки навчального контенту (тексти, відео, файли); інструментарій для оцінювання (тести, завдання); комунікаційні засоби (форуми, чати) та система аналітики й звітності для відстеження навчального прогресу [40].

Ефективність платформи безпосередньо залежить від того, наскільки добре вона задовольняє потреби трьох основних груп користувачів, кожна з яких має унікальні цілі та сценарії взаємодії.

Перша і найчисельніша група – Студент. Для нього пріоритетом є інтуїтивно зрозумілий інтерфейс, легкий доступ до навчальних матеріалів, чітке розуміння свого прогресу та термінів, а також зручні інструменти для

здачі робіт та комунікації з викладачем та іншими студентами [37]. Негативний користувацький досвід на цьому рівні може призвести до демотивації та зниження ефективності навчання.

Друга ключова роль – Викладач (інструктор). Його потреби зосереджені на інструментах для створення контенту та управління курсом: зручний редактор для уроків, можливість створювати різноманітні типи завдань та тестів, ефективний інтерфейс для перевірки робіт та надання зворотного зв'язку, а також доступ до аналітики для моніторингу успішності студентів [24].

Третя, не менш важлива роль – Адміністратор. Він відповідає за технічне функціонування системи в цілому: управління обліковими записами та ролями, створення та архівацію курсів, налаштування глобальних параметрів платформи, забезпечення безпеки та цілісності даних, а також інтеграцію з іншими інформаційними системами закладу. Успішна архітектура системи повинна враховувати та гармонійно поєднувати потреби всіх трьох ролей.

На основі аналізу предметної області та потреб користувачів формулюється комплекс вимог до системи. Функціональні вимоги описують, *що* система повинна робити, і безпосередньо випливають з ідентифікованих функціональних блоків: реалізація реєстрації та автентифікації, створення курсів з модульною структурою, завантаження контенту, розробка системи тестування, механізм здачі та оцінювання завдань тощо. Проте для довгострокової життєздатності та конкурентоспроможності платформи не менш важливими є нефункціональні вимоги, які визначають, *як* система повинна виконувати свої функції. Ключовими серед них є масштабованість – здатність системи ефективно працювати при значному зростанні кількості користувачів та обсягів даних без деградації продуктивності, що є критичним для онлайн-освіти [14]. Надзвичайно важливою є безпека, оскільки платформа оперує чутливими персональними даними та результатами інтелектуальної діяльності, що вимагає надійних механізмів захисту від несанкціонованого доступу [39].

Юзабіліті та доступність є визначальними для сприйняття продукту кінцевими користувачами; система має бути зрозумілою для людей з різним рівнем технічної грамотності та доступною для використання людьми з обмеженими можливостями [38]. Ігнорування цих атрибутів якості на ранніх етапах проєктування неминуче призводить до створення технічного боргу та репутаційних ризиків у майбутньому. Таким чином, комплексне врахування як функціональних, так і нефункціональних аспектів є фундаментальною передумовою для створення успішної платформи дистанційного навчання.

Висновки до Розділу 1

У першому розділі здійснено системний аналіз сучасних методологій управління ІТ-проєктами розробки програмного забезпечення, зокрема каскадних, гнучких (Agile) та гібридних моделей. Виявлено їхні переваги й недоліки в контексті створення складних веборієнтованих платформ дистанційного навчання. Доведено, що класичні моделі забезпечують високу передбачуваність і контроль, але не відповідають умовам високої невизначеності та динаміки вимог. Agile-підходи, навпаки, дають необхідну гнучкість і швидке отримання зворотного зв’язку, однак іноді призводять до втрати стратегічного фокусу. Порівняльний аналіз гібридних рішень (Water-Scrum-Fall, Scrumban) засвідчив їхню спроможність поєднувати стратегічне планування з тактичною адаптивністю. На цій основі обґрунтовано доцільність використання двонаправленої структури управління як перспективного підходу для розробки EdTech-продуктів. Розкрито її ключові принципи (стратегічна спрямованість, емпіричний контроль, прозорість і спільна відповідальність) та описано інформаційні потоки «зверху-вниз» і «знизу-вгору». Також окреслено специфіку розробки дистанційних платформ з урахуванням функціональних і нефункціональних вимог та потреб різних груп користувачів. Таким чином, перший розділ сформував науково-методологічне підґрунтя для подальшого проєктування системи й реалізації двонаправленої моделі в наступних розділах.

РОЗДІЛ 2.
ПРОЄКТУВАННЯ ТА ОБҐРУНТУВАННЯ ТЕХНОЛОГІЧНОГО СТЕКУ

2.1. Двофакторна структуризація проєкту веборієнтованого програмного продукту дистанційної платформи навчання

Ефективне управління проєктом веборієнтованого програмного продукту потребує одночасного урахування двох взаємопов'язаних вимірів: структури робіт (що необхідно зробити) та структури команди (хто це робить). Такий підхід утворює двофакторну (двовекторну) структуризацію, яка забезпечує прозорість планування, чіткий розподіл відповідальності та можливість гнучкого коригування процесу у відповідь на зміни. У межах даного підходу використовуються три взаємодоповнювальні інструменти:

- дерево робіт,
- структура команди,
- матриця відповідальності.

Одним із ключових інструментів планування та організації проєктної діяльності у сучасному управлінні програмними продуктами є дерево робіт (Work Breakdown Structure, WBS). WBS забезпечує формалізоване уявлення проєкту шляхом ієрархічної декомпозиції його цілей та завдань на структуровані складові елементи, що дає можливість комплексно керувати обсягом робіт, ресурсами та відповідальністю. Згідно зі стандартами PMI та ISO 21511, дерево робіт розглядається як базовий артефакт управління проєктами, який дозволяє відслідковувати взаємозв'язок між стратегічними цілями й операційними діями команди розробки.

Для веборієнтованої платформи дистанційного навчання дерево робіт має три рівні декомпозиції, що відображають дві площини управління – продуктову та процесну.

Рівень 1 – Фази проєкту. На цьому рівні формується високорівнева структура життєвого циклу проєкту, яка охоплює п'ять основних фаз: аналіз

вимог, проєктування архітектури, розробка, тестування та впровадження. Таке розбиття відповідає класичним принципам управління проєктами і водночас сумісне з ітеративними підходами (Agile/Scrum), коли всередині кожної фази можуть реалізовуватися спринти.

Рівень 2 – Основні продукти/модулі. Для кожної фази визначаються ключові функціональні підсистеми та модулі майбутньої платформи, які є носіями бізнес-цінності: «Аутентифікація користувачів», «Управління курсами», «Система завдань і тестів», «Аналітика та звітність», «Інтеграція із зовнішніми інформаційними системами (SIS)». Це дозволяє розподілити роботи за функціональними напрямами й забезпечити цільову орієнтацію кожного модуля на кінцевого користувача.

Рівень 3 – Конкретні роботи. На найнижчому рівні ієрархії здійснюється деталізація до конкретних робочих пакетів і завдань, які можуть бути безпосередньо призначені членам команди та оцінені у трудомісткості. Прикладами таких робіт є: написання REST API для модулю «Курси», створення компонентів React для відображення завдань, налаштування Docker Compose для локальної розробки, конфігурування GitHub Actions для CI/CD тощо.

Рис. 2.1 – **Дерево робіт проєкту платформи дистанційного навчання**

Джерело: побудовано автором

Верхній блок рисунка «Проєкт платформи дистанційного навчання» відображає загальну мету й контекст проєкту – створення веборієнтованої системи дистанційного навчання, яка забезпечує керування навчальними курсами, завданнями, аналітикою та інтеграцією із зовнішніми системами. Він є верхівкою дерева робіт і концентрує в собі всі наступні напрями діяльності, надаючи їм стратегічну спрямованість та визначаючи кінцевий результат, до якого прагне команда.

Блок «Аналіз вимог» описує процес формування й деталізації вимог до системи. У цьому контексті особлива увага приділяється аутентифікації користувачів, тобто визначенню механізмів реєстрації та входу, налаштуванню ролей і прав доступу. Паралельно розробляється та узгоджується специфікація RESTful API як основи для взаємодії клієнтської та серверної частин системи. Це дозволяє чітко зафіксувати очікування від програмного продукту й підготувати технічний фундамент для подальшого проєктування.

Блок «Проєктування архітектури» відображає етап створення структури та логіки майбутньої платформи. Тут формується модуль управління курсами, визначається його функціональна наповненість і взаємозв’язки з іншими підсистемами. Паралельно у бібліотеці React прототипуються компонентні елементи інтерфейсу користувача (наприклад, картки курсів, елементи уроків, блоки завдань), що забезпечує модульність та перевикористання UI. Проєктування архітектури закладає основу для високої масштабованості та підтримуваності коду.

Блок «Розробка» фокусується на практичному втіленні спроєктованих рішень. У межах цього етапу реалізується система завдань і тестів, яка дозволяє викладачам створювати й адмініструвати завдання, а студентам – здавати роботи й отримувати оцінки. Паралельно створюється конфігурація Docker Compose для Node.js, React і PostgreSQL, що забезпечує уніфіковане середовище розробки, а також налаштовується автоматизований процес CI/CD

у GitHub Actions, який гарантує стабільність і якість програмного продукту. Це забезпечує контрольоване й прозоре створення інкрементів платформи.

Блок «Тестування» відображає перевірку модулю аналітики та звітності, тобто перевірку коректності збору та відображення даних про активність користувачів та успішність навчання. На цьому етапі верифікується відповідність розробленого функціоналу сформульованим критеріям приймання, а також можуть проводитися автоматизовані тести інтерфейсу та функціональних модулів. Це дозволяє своєчасно виявляти помилки та забезпечувати якість кінцевого продукту.

Фінальний блок «Впровадження» присвячений інтеграції платформи дистанційного навчання із зовнішніми інформаційними системами закладу, такими як SIS. Тут налаштовується обмін даними, забезпечується безпека й узгодженість інформації, що дозволяє платформі функціонувати як частині єдиного інформаційного середовища. Завдяки цьому продукт отримує повноцінну практичну цінність і може бути безпосередньо використаний у навчальному процесі.

Запропонована структура WBS забезпечує прозорість контролю прогресу по кожній гілці проєкту, формування обсягу робіт для спринтів та оптимізацію розподілу ресурсів. Вона є не статичним документом, а динамічним артефактом, що може адаптуватися під впливом результатів і зворотного зв’язку у межах двонаправленої моделі управління.

Другий вектор структуризації ґрунтується на організаційній побудові команди й розподілі ролей, повноважень та зон відповідальності, що забезпечують керованість життєвого циклу веборієнтованого програмного продукту. У межах двонаправленої моделі управління (поєднання потоків «зверху-вниз» та «знизу-вгору») команда виступає не лише виконавцем робіт, а й активним генератором емпіричного зворотного зв’язку, який безперервно коригує продуктові цілі, беклог і план релізів. Структура команди визначає, хто приймає стратегічні рішення, хто забезпечує процесну дисципліну, хто

реалізує функціонал і гарантує якість, а також як саме циркулює інформація між цими ролями.

Керівник проєкту / Product Owner формує продуктове бачення, декомпонує бізнес-цілі у вимоги та епіки, підтримує й пріоритезує беклог, визначає критерії приймання, стежить за дотриманням обмежень термінів і бюджету, а також веде реєстр ризиків та план їхнього реагування. Його зона відповідальності – стратегічна когерентність розробки з очікуваною цінністю для користувачів і замовника, узгодження дорожньої карти з ресурсними можливостями команди, забезпечення трасованості «ціль → вимога → інкремент → метрика впливу». Саме ця роль матеріалізує потік «зверху-вниз», транслюючи пріоритети та критерії якості в операційні завдання.

Скрам-майстер (або координатор процесів) відповідає за методичну цілісність та ритм роботи команди. Він організовує ітераційні події (планування, щоденні синки, огляд спринту, ретроспективу), усуває перешкоди, захищає команду від неузгоджених змін контексту та сприяє безперервному вдосконаленню процесів. На рівні метрик Скрам-майстер відстежує швидкість, час циклу, стабільність спринтів, дефектну щільність, забезпечує прозорість через дашборди та артефакти (Definition of Ready/Done). Завдяки систематизації процесу і культивуванню відкритого обміну даними ця роль підсилює «знизу-вгору» компонент двонаправленої моделі – перетворює емпіричні спостереження команди на керовані зміни в практиках.

Команда розробки (Front-end і Back-end) реалізує інкременти функціональності відповідно до прийнятих архітектурних рішень і стандартів коду. Вона забезпечує контрактну сумісність API та UI, покриття модульними й інтеграційними тестами, дотримання принципів читабельності, інкапсуляції та повторного використання. Важливо, що розробники не лише «перекладають» вимоги у код, а й формують технічний зворотний зв’язок: пропонують оптимізації, виявляють приховані залежності й технічний борг, уточнюють оцінки трудомісткості, впливаючи на пріоритезацію беклогу та планування релізів. Через пул-реквести, код-рев’ю та обговорення

архітектурних компромісів вони утримують високий рівень внутрішньої якості продукту.

Дизайнер / аналітик виконує дві взаємопідсилювальні функції. Аналітична – це еліцитація та нормалізація вимог, побудова користувацьких сценаріїв, CJM і прототипування логіки взаємодії. Дизайнерська – створення інтерфейсних патернів і компонентів у Figma, контроль цілісності UX, врахування доступності (наприклад, відповідність WCAG 2.1) та дидактичних принципів для EdTech-контенту. Результати досліджень і юзабіліті-тестів, артефакти прототипування та експертні висновки інтегруються у беклог і критерії приймання, забезпечуючи зв’язок між педагогічною доцільністю та технічною реалізацією.

QA-інженер (забезпечення якості) розробляє стратегію тестування, формує тест-план і тест-процедури, підтримує матрицю трасованості «вимога → тест-випадок», організовує регресію та, за можливості, впроваджує автоматизовані тести на критичних шляхах користувача. Він відповідає за збір та інтерпретацію метрик якості (покриття тестами, дефектна щільність, середній час виправлення, стабільність релізів), проводить аналіз першопричин дефектів і спільно з розробниками знижує ризики повторних помилок. Разом із Product Owner QA визначає критерії приймання, тим самим замикаючи контур двонаправленої моделі: фактичні дані про якість повертаються у планування.

У контексті кваліфікаційної роботи допускається поєднання ролей однією особою; однак організаційно-процесне розмежування рішень і відповідальностей має бути збережене. Критично важливо забезпечити чіткі інтерфейси взаємодії між ролями (канали комунікації, частота синхронізацій, рішення-гейти), єдину систему артефактів (беклог, специфікації, прототипи, пул-реквести, тести) та прозорий інформаційний простір (дашборди, журнали змін, звіти). Саме така конфігурація робить потоки «зверху-вниз» і «знизу-вгору» взаємно доповнюваними: стратегія визначає напрям, а емпіричні дані

та експертиза команди – темп і спосіб руху, мінімізуючи ризики та підвищуючи предиктивність графіка.

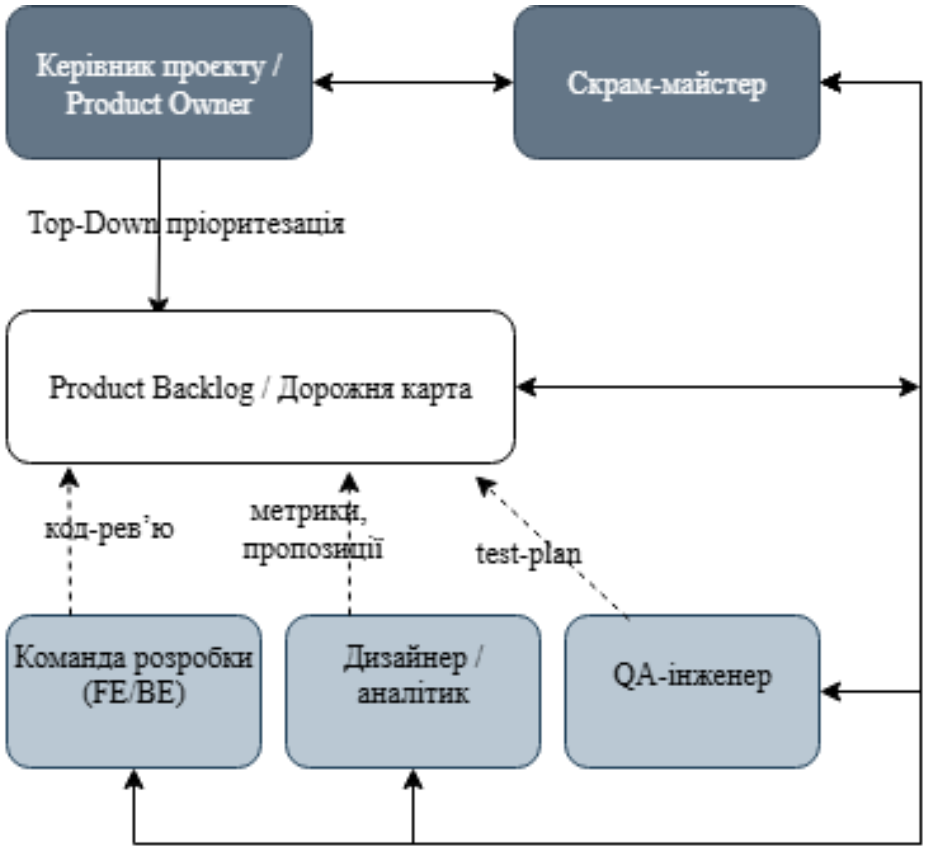


Рис. 2.2 – **Організаційна структура та двонаправлені контури взаємодії команди проєкту**

Джерело: побудовано автором

Необхідною складовою двофакторної структуризації є узгодження між декомпозицією робіт і організаційною структурою команди. Такий зв'язок досягається за допомогою матриці відповідальності (Responsibility Assignment Matrix), найпоширенішим варіантом якої є модель RACI. Ця матриця дозволяє уніфіковано й прозоро відобразити розподіл функцій, повноважень і зон відповідальності між усіма учасниками проєкту, тим самим мінімізуючи ризики дублювання, прогалин чи конфліктів ролей.

Матриця відповідальності у форматі RACI поєднує два виміри: перелік робочих пакетів із дерева робіт (Work Breakdown Structure) та перелік

визначених ролей у команді. Для кожної конкретної роботи у відповідному рядку зазначається, яка роль є Responsible (R) – безпосередньо виконує роботу; Accountable (A) – відповідає за кінцевий результат і ухвалює рішення; Consulted (C) – надає консультації й експертний зворотний зв'язок; Informed (I) – інформується про хід виконання. Завдяки такій структурі забезпечується трасованість «робочий пакет → відповідальні ролі», що є базою для управління завантаженістю, строками та якістю.

Наприклад, для задачі «Розробка модуля управління курсами» у колонці R будуть зазначені Front-end та Back-end розробники, оскільки вони створюють код і реалізують функціональність; у колонці A – Product Owner, який затверджує вимоги й приймає результат; у колонці C – дизайнер/аналітик, який консультує щодо UX і бізнес-логіки; у колонці I – QA-інженер, який має бути поінформований про зміни для побудови тестових сценаріїв. Такий підхід дозволяє швидко ідентифікувати потенційні «пляшкові горлечка», уникнути дублювання зусиль, а також забезпечує контроль виконання завдань у двох вимірах одночасно – як по роботах, так і по ролях.

	A	B	C	D	E	F	G
1		Product Owner	Скрам-майстер	Front-end розробник	Back-end розробник	Дизайнер/ аналітик	QA-інженер
2	Аналіз вимог	A	C	I	I	R	C
3	Розробка модулю аутентифікації	A	I	R	R	C	I
4	Розробка модуля управління курсами	A	I	R	R	C	I
5	Створення системи завдань і тестів	A	I	R	R	C	I
6	Налаштування CI/CD	I	A	C	R	I	I
7	Інтеграція із SIS	A	I	C	R	C	I

Рис. 2.3 – **Матриця відповідальності (RACI) для ключових робіт проєкту**

Джерело: побудовано автором

Наукові дослідження та стандарти PMI наголошують, що RACI-матриця є не просто таблицею, а інтеграційним артефактом процесів управління: вона дозволяє поєднати планування обсягу робіт, управління ресурсами, комунікаційне планування та моніторинг. В умовах двонаправленої моделі управління проєктом матриця виконує додаткову функцію – фіксує й

формалізує канали зворотного зв’язку від виконавців до керівництва, сприяючи адаптивності й гнучкому реагуванню на зміни.

2.2. Обґрунтування вибору технологічного стеку

Вибір технологічного стеку є критично важливим етапом проєктування, що безпосередньо впливає на продуктивність, надійність, вартість розробки та подальшої підтримки програмного продукту. Рішення щодо вибору конкретних технологій приймалося не довільно, а на основі комплексного аналізу функціональних та нефункціональних вимог до платформи дистанційного навчання, а також з урахуванням сучасних тенденцій та найкращих практик у веброзробці.

Серверна частина є ядром системи, що відповідає за реалізацію бізнес-логіки, обробку запитів, взаємодію з базою даних та надання API для клієнтської частини. Для вибору технології було проаналізовано три провідні платформи: Node.js, Python з фреймворком Django та PHP з фреймворком Laravel. Python/Django вирізняється підходом «batteries-included», що дозволяє швидко розробляти типовий функціонал, та має потужні бібліотеки для роботи з даними. PHP/Laravel є зрілою та надзвичайно поширеною екосистемою з низьким порогом входження.

Незважаючи на переваги конкурентів, для даного проєкту було обрано платформу Node.js. Ключовим аргументом на користь цього рішення є можливість використання мови JavaScript на всіх рівнях застосунку – як на клієнтській, так і на серверній стороні. Такий підхід, відомий як «JavaScript everywhere», суттєво спрощує процес розробки, зменшує когнітивне навантаження на розробника та сприяє повторному використанню коду. Фундаментальною перевагою Node.js є його архітектура, заснована на подієво-орієнтованій моделі та неблокуючому введенні-виведенні (I/O). Це робить платформу ідеальною для розробки застосунків з великою кількістю одночасних з'єднань, що є типовим сценарієм для освітньої платформи, де багато користувачів одночасно переглядають контент. В якості основного

фреймворку може бути використаний мінімалістичний Express.js для максимальної гнучкості або більш структурований NestJS, який побудований на TypeScript та впроваджує патерни, що сприяють кращій архітектурі та масштабованості коду.

Клієнтська частина відповідає за створення інтерактивного, динамічного та зручного користувацького інтерфейсу. Сучасна розробка інтерфейсів базується на використанні JavaScript-фреймворків для побудови односторінкових застосунків (SPA). Основними претендентами були React, Vue та Angular. Angular є комплексним та думковідмінним (opinionated) фреймворком, що добре підходить для великих корпоративних проєктів. Vue вирізняється простотою та низьким порогом входження.

Проте, вибір було зроблено на користь бібліотеки React. Це рішення обґрунтоване кількома факторами. По-перше, React має найбільшу та найактивнішу спільноту розробників, що гарантує величезну кількість готових рішень, бібліотек та якісної документації. По-друге, його компонентний підхід до побудови інтерфейсів ідеально відповідає структурі освітньої платформи: UI можна декомпозувати на незалежні компоненти, що перевикористовуються (наприклад, CourseCard, LessonItem, QuizQuestion). По-третє, екосистема React, що включає потужні інструменти для управління станом додатку (Redux, MobX) та маршрутизації (React Router), дозволяє створювати складні та масштабовані клієнтські застосунки. Використання віртуального DOM забезпечує високу продуктивність інтерфейсу навіть при частих оновленнях даних.

Вибір системи управління базами даних (СУБД) визначався структурою даних предметної області. Розглядалися два основних підходи: реляційний (на прикладі PostgreSQL) та нереляційний/документо-орієнтований (на прикладі MongoDB). MongoDB та інші NoSQL-рішення пропонують гнучку схему даних та відмінне горизонтальне масштабування, що є корисним для зберігання неструктурованих даних.

Однак, як було визначено в попередніх підрозділах, дані освітньої платформи є високоструктурованими та мають чіткі зв'язки: користувачі, курси, уроки, оцінки. У таких умовах забезпечення цілісності та узгодженості даних є першочерговим пріоритетом. Тому вибір було зроблено на користь реляційної СУБД PostgreSQL. Вона гарантує ACID-транзакції, що є критично важливим для операцій, пов'язаних з оцінками та прогресом навчання. PostgreSQL підтримує потужну мову запитів SQL, що дозволяє легко виконувати складні вибірки та агрегації даних для аналітики. Крім того, сучасні версії PostgreSQL мають розширену підтримку JSONB, що дозволяє зберігати частково структуровані дані всередині реляційної моделі, поєднуючи найкраще з обох світів.

Для забезпечення надійності, відтворюваності та автоматизації процесу розробки було обрано сучасні DevOps-інструменти. Для управління середовищем розробки та розгортання було прийнято рішення використовувати Docker. Технологія контейнеризації дозволяє «запакувати» застосунок та всі його залежності в ізольований контейнер. Це вирішує проблему «на моїй машині все працює», гарантуючи, що середовище розробки, тестування та продуктивне середовище є ідентичними.

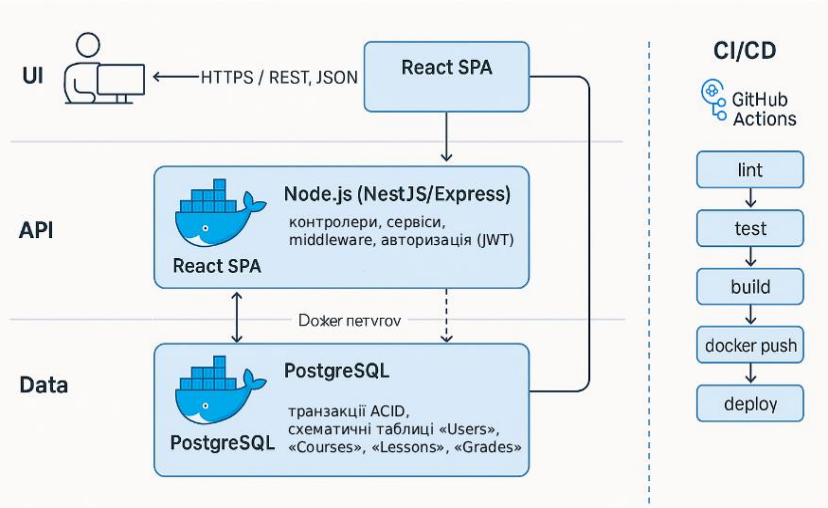


Рис. 2.4 – **Технологічний стек та шлях запиту**

Джерело: побудовано автором

Таблиця 2.1

Обґрунтування вибору технологічного стеку для платформи дистанційного навчання

Шар/компонент	Кандидати (розглядалися)	Критерії відбору (ключові)	Переваги кандидатів	Обмеження/ризики	Рішення (обрано)	Коротке обґрунтування вибору
Серверна платформа	**Node.js (Express/NestJS)**; Python **Django**; PHP **Laravel**	Продуктивність при високій паралельності; єдина мова FE/BE; зрілість екосистеми; швидкість розробки; тестованість; кадровий ринок	Node.js: неблокуюче I/O, «JavaScript everywhere», багата NPM-екосистема. Django: «batteries-included», швидкий CRUD, ORM. Laravel: зрілий фреймворк, простий старт	Django/Laravel: різні мови з фронтендом → більша когнітивна ціна; Waterfall-подібні шаблони; Node.js: потребує дисципліни архітектури	**Node.js (NestJS + Express)**	Єдина мова на всіх рівнях (JS/TS) знижує TCO; неблокуюче I/O для великої кількості одночасних з'єднань; NestJS дає структуровану архітектуру, DI, модульність
Клієнтська частина (SPA)	**React**; Angular; Vue	Компонентність; продуктивність UI; масштабованість; екосистема; документація; поріг входу	React: найбільша спільнота, VDOM, гнучкість; Angular: все з коробки; Vue: простота	Angular: більш «думковідмінний», важчий стек; Vue: менша корпоративна підтримка	**React (+ React Router, Redux/RTK)**	Компонентний підхід і велика екосистема бібліотек; стабільний hiring-pool; зручний для побудови модульного EdTech-UI (CourseCard, LessonItem, QuizQuestion)
СУБД	**PostgreSQL**; MongoDB	Модель даних (зв'язні сутності); транзакційність (ACID); складні запити/аналітика; масштабування; гнучкість схеми	PostgreSQL: ACID, складні JOIN/CTE, індекси; JSONB для напівструктурованих даних. MongoDB: гнучка схема, горизонтальне шардіння	MongoDB: відсутність класичних транзакцій у старих версіях, ускладнення консистентності; PostgreSQL: потребує чіткої схеми	**PostgreSQL**	Дані LMS високоструктуровані (користувачі, курси, уроки, оцінки); потрібна цілісність і складні запити; JSONB покриває гнучкі поля
Контейнеризація	**Docker**; Podman; BM	Відтворюваність середовищ; портативність; швидкість деплою; підтримка в CI/CD	Docker: стандарт де-факто, Compose/BuildKit, екосистема registry	Вимагає базових DevOps-навичок; оркестрація поза межами дипломного MVP	**Docker (+ Docker Compose)**	Ізолює залежності, «однакове» середовище Dev/Test/Prod; швидкий локальний старт команди
CI/CD	**GitHub Actions**; GitLab CI; Jenkins	Інтеграція з репозиторієм; простота конфігурації; секрети; паралельні job'и	GitHub Actions: нативна інтеграція з GitHub, маркетплейс actions, YAML-workflow	Jenkins: потребує хостингу/адміністрування; GitLab CI – інша платформа	**GitHub Actions**	Автоматизація lint/unit-tests/build/push-image на push/PR; мінімальний ops-овергенд; швидке підключення до Docker-реєстру

Джерело: побудовано автором

Для автоматизації процесів збірки, тестування та інтеграції коду було обрано GitHub Actions. Цей інструмент безперервної інтеграції та доставки (CI/CD) тісно інтегрований з репозиторієм коду на GitHub. Він дозволяє налаштувати автоматизовані робочі процеси (workflows), які запускаються при кожній зміні в коді (push). Наприклад, можна автоматично запускати статичний аналіз коду (linting), unit-тести та збірку Docker-образів. Впровадження CI/CD на ранніх етапах проєкту підвищує якість коду, прискорює виявлення помилок та спрощує процес доставки оновлень.

2.3. Організація процесу розробки згідно з двонаправленою моделлю

Теоретична концепція двонаправленої структури управління набуває практичної цінності лише за умови її інструментального та процесного забезпечення. Ефективна організація розробки вимагає створення інтегрованого цифрового середовища, де кожен інструмент та процес слугує для посилення та формалізації інформаційних потоків «зверху-вниз» та «знизу-вгору». Даний підрозділ описує методологічний каркас та інструментарій, обраний для реалізації проєкту розробки платформи дистанційного навчання відповідно до принципів двонаправленого управління.

Для забезпечення прозорості та ефективної комунікації було обрано синергетичний набір інструментів, кожен з яких відіграє унікальну роль у підтримці двонаправленої моделі.

- Jira (або Trello як спрощений аналог) обрано в якості центральної системи для управління завданнями та беклогом. Цей інструмент є ключовим для реалізації потоку «зверху-вниз»: власник продукту (Product Owner) створює та пріоритезує беклог у вигляді епіків та історій користувачів (user stories), що є прямою трансляцією бізнес-вимог у конкретні робочі елементи. Водночас Jira слугує потужним каналом для потоку «знизу-вгору»: розробники оновлюють статуси завдань, додають коментарі з технічними

деталями, а тестувальники створюють звіти про дефекти. Таким чином, Jira стає єдиним джерелом правди про стан проєкту, доступним для всіх учасників.

- Git та GitHub є фундаментальними для управління вихідним кодом та спільної розробки. Система контролю версій Git забезпечує відстеження всіх змін, тоді як платформа GitHub надає інструменти для колаборації. Ця зв'язка є критично важливою для потоку «знизу-вгору». Механізм запитів на злиття (Pull Requests) є не просто технічною процедурою, а формалізованим процесом для код-рев'ю, де команда надає технічний зворотний зв'язок, обговорює архітектурні рішення та забезпечує якість коду ще до його інтеграції в основну гілку.
- Figma використовується для проєктування інтерфейсів та прототипування. Цей інструмент відіграє важливу роль на ранніх етапах життєвого циклу задачі, дозволяючи візуалізувати вимоги (потік «зверху-вниз»). Разом з тим, завдяки функціям коментування, Figma стає платформою для раннього зворотного зв'язку «знизу-вгору»: розробники можуть оцінити технічну складність реалізації дизайну, а стейкхолдери – надати відгук щодо зручності інтерфейсу ще до початку розробки, що значно знижує вартість внесення змін.

Життєвий цикл задачі, від моменту її виникнення до релізу, є мікрокосмом, що наочно демонструє роботу двонаправленої моделі. Цей процес можна візуалізувати у вигляді циклічної діаграми, що складається з наступних етапів:

1. Ідея та Формулювання (Top-Down): На основі бізнес-цілі або відгуку користувача власник продукту формулює історію користувача в Jira, описуючи, *хто*, *що* і *навіщо* хоче отримати.

2. Проєктування та Декомпозиція (Top-Down → Bottom-Up): Дизайнер створює макет у Figma. На етапі обговорення (grooming) команда розробки аналізує історію, ставить уточнюючі питання та проводить технічну декомпозицію на підзадачі. Це перший етап зворотного зв'язку «знизу-вгору», де технічна експертиза команди впливає на фіналізацію вимог.

3. Розробка та Код-рев'ю (Bottom-Up): Розробник реалізує функціонал в окремій Git-гілці та створює Pull Request. Процес код-рев'ю колегами є чистою формою потоку «знизу-вгору», спрямованого на підвищення якості.

4. Тестування (Bottom-Up): QA-інженер перевіряє реалізований функціонал на відповідність критеріям приймання. Знайдені дефекти реєструються в Jira, що є прямим сигналом для команди та власника продукту про якість інкременту.

5. Демонстрація та Валідація (Bottom-Up → Top-Down): Готовий функціонал демонструється зацікавленим сторонам. Отриманий зворотний зв'язок є найважливішим каналом «знизу-вгору», який може призвести до коригування беклогу та навіть зміни пріоритетів на стратегічному рівні.

6. Реліз та Моніторинг: Після успішної демонстрації функціонал розгортається на продуктивному середовищі. Збір аналітики та відгуків реальних користувачів замикає цикл, надаючи дані для формування нових ідей та вимог.

Процес розробки організовано у вигляді коротких ітерацій (спринтів) тривалістю два тижні, що дозволяє регулярно постачати цінність та отримувати зворотний зв'язок. Ключові події Scrum-фреймворку використовуються як формалізовані механізми для реалізації двонаправленої комунікації.

Планування спринту є точкою, де потік «зверху-вниз» (пріоритети від власника продукту) зустрічається з потоком «знизу-вгору» (оцінка складності та можливостей від команди). Результатом є реалістичний план роботи на найближчу ітерацію.

Огляд спринту (Демо) є головною подією для отримання зворотного зв'язку від стейкхолдерів. Команда демонструє працюючий інкремент продукту, що дозволяє перевірити гіпотези та адаптувати подальші плани на основі реальних результатів, а не припущень.

Ретроспектива спринту – це внутрішній механізм зворотного зв'язку «знизу-вгору», сфокусований на процесі. Команда аналізує, що було добре, що

можна покращити, і формулює конкретні дії для вдосконалення своєї роботи в наступному спринті.

Аналіз метрик (швидкість команди, час циклу задачі, кількість дефектів) слугує об'єктивним, кількісним каналом «знизу-вгору», що дозволяє керівництву проєкту оцінювати стан справ та приймати більш обґрунтовані рішення щодо планування.

Висновки до Розділу 2

У другому розділі розроблено архітектурно-технологічну основу платформи дистанційного навчання, що ґрунтується на принципах модульного моноліту з можливістю подальшої еволюції до мікросервісів.

Обґрунтовано вибір архітектурного стилю RESTful API та реляційної моделі даних із використанням PostgreSQL для забезпечення цілісності й надійності освітньої інформації. Сформовано логічну структуру бази даних із ключовими сутностями (користувачі, курси, уроки, завдання, оцінки) і зв’язками між ними.

Детально розглянуто вибір технологічного стеку: Node.js як серверна платформа завдяки «JavaScript everywhere» та неблокуючій архітектурі; React як клієнтська бібліотека з компонентним підходом і широкою екосистемою; PostgreSQL як СУБД, оптимальну для високоструктурованих освітніх даних; Docker та GitHub Actions як інструменти контейнеризації й CI/CD. Водночас описано організацію процесу розробки відповідно до двонаправленої моделі – від формування та пріоритезації беклогу в Jira до код-рев’ю в GitHub і прототипування у Figma. Показано, як на кожному етапі відбувається перетин потоків «Top-Down» і «Bottom-Up», що підвищує прозорість, керованість і швидкість зворотного зв’язку. Таким чином, другий розділ забезпечив практичну інженерну базу для реалізації концепції, розробленої у першому розділі, та заклав фундамент для побудови й тестування прототипу платформи.

РОЗДІЛ 3.
ПРАКТИЧНА РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ

3.1. Налаштування середовища розробки та інфраструктури

Перехід від теоретичного проєктування до практичної реалізації вимагає створення надійної, відтворюваної та ефективної інфраструктури розробки. Правильно налаштоване середовище не лише прискорює процес написання коду, але й закладає основи для стабільного функціонування системи на всіх етапах її життєвого циклу, від локальної машини розробника до продуктивного сервера. У даному підрозділі детально описано архітектурні та інфраструктурні рішення, прийняті для забезпечення якості та автоматизації процесу розробки платформи дистанційного навчання.

Основою для ефективної командної роботи є стандартизація середовища розробки та впровадження автоматизованих процесів, що мінімізують вплив людського фактору. Для досягнення цієї мети були застосовані сучасні практики управління кодом, контейнеризації та безперервної інтеграції.

На етапі організації структури проєкту було розглянуто два підходи: використання окремих репозиторіїв для клієнтської (front-end) та серверної (back-end) частин або застосування монорепозиторію. Хоча окремі репозиторії надають повну незалежність командам, для проєкту, що розробляється невеликою командою, такий підхід може призвести до ускладнення синхронізації змін між клієнтом та сервером, особливо при модифікації API.

З огляду на це, було прийнято рішення на користь монорепозиторію. Вся кодова база проєкту – серверний застосунок на Node.js, клієнтський на React, а також конфігураційні файли – зберігається в єдиному Git-репозиторії. Така структура має низку переваг. По-перше, вона спрощує управління залежностями та виконання наскрізних змін, оскільки будь-яка модифікація API серверної частини та її використання на клієнті відбуваються в рамках одного коміту та одного pull request. По-друге, це забезпечує єдине джерело правди та спрощує налаштування процесів CI/CD. Всередині

монорепозиторію проєкт має чітку директоріальну структуру, наприклад: /server, /client, /docker, що забезпечує логічне розмежування коду різних частин системи.

Для вирішення проблеми консистентності середовищ розробки та уникнення конфліктів версій програмного забезпечення було застосовано технологію контейнеризації Docker. Замість того, щоб кожен розробник встановлював на свою локальну машину Node.js, PostgreSQL та інші залежності потрібних версій, було створено набір Docker-образів, що інкапсулюють кожну частину системи.

Для оркестрації цих контейнерів на локальній машині використовується інструмент Docker Compose. Було створено конфігураційний файл docker-compose.yml, який описує всі сервіси, необхідні для роботи застосунку:

- api-server: сервіс для серверної частини, що будується з Dockerfile у директорії /server та запускає Node.js застосунок.
- client-app: сервіс для клієнтської частини, що запускає dev-сервер React.
- db: сервіс, що використовує офіційний образ PostgreSQL та монтує локальний том для збереження даних між перезапусками.
- nginx (опційно): сервіс, що виступає в ролі зворотного проксі для розподілу запитів між клієнтською та серверною частинами.

Такий підхід дозволяє будь-якому члену команди розгорнути повне робоче середовище проєкту однією командою docker-compose up, гарантуючи, що всі працюють в ідентичних умовах, що повністю відповідають конфігурації продуктивного середовища.

З метою автоматизації перевірки якості коду та раннього виявлення помилок було налаштовано пайплайн безперервної інтеграції (CI) за допомогою GitHub Actions. У корені репозиторію було створено директорію .github/workflows з YAML-файлом, що описує автоматизований робочий процес.

Даний пайплайн налаштовано на автоматичний запуск при кожному створенні або оновленні Pull Request до основної гілки розробки (main або develop). Він включає наступні кроки (jobs):

1. Checkout Code: Завантаження коду з репозиторію.
2. Setup Environment: Встановлення необхідної версії Node.js.
3. Install Dependencies: Виконання команди npm install для серверної та клієнтської частин для встановлення всіх залежностей.
4. Linting: Запуск статичного аналізатора коду (наприклад, ESLint) для перевірки відповідності коду встановленим стандартам форматування та виявлення потенційних помилок.
5. Run Tests: Запуск автоматизованих тестів (наприклад, unit-тестів за допомогою Jest) для перевірки коректності бізнес-логіки.

Якщо будь-який з цих кроків завершується з помилкою, GitHub Actions автоматично блокує можливість злиття Pull Request, інформуючи розробника про необхідність виправлення. Такий підхід гарантує, що в основну гілку потрапляє лише код, який пройшов усі автоматичні перевірки, що є ключовим елементом забезпечення високої якості та стабільності програмного продукту.

3.2. Розробка ключових функціональних модулів

Представимо карту продукту платформи дистанційного навчання, яка слугує інтегральним артефактом узгодження стратегії та реалізації у межах двонаправленої моделі управління. На одній площині вона візуалізує логіку ціннісної пропозиції (користувацькі проблеми/потреби, унікальна цінність, неринкова перевага), цільові сегменти користувачів і канали просування, а також ключові метрики успіху, структуру витрат і потенційні джерела доходів (MVP/підписка/послуги). На іншій – відображає зв’язок цих елементів із продуктово-технічною реалізацією: від формування беклогу та пріоритизації інкрементів до планування ресурсів і перевірки гіпотез на даних. Таким чином, карта продукту виконує роль «референтної моделі» для прийняття рішень: вона забезпечує каскадування цілей Top-Down (від місії та бізнес-цілей до

вимог) і водночас збирає емпіричний зворотний зв’язок Bottom-Up (метрики використання, відгуки, результати експериментів), що дає змогу системно керувати ризиками та концентрувати зусилля на найцінніших інкрементах.

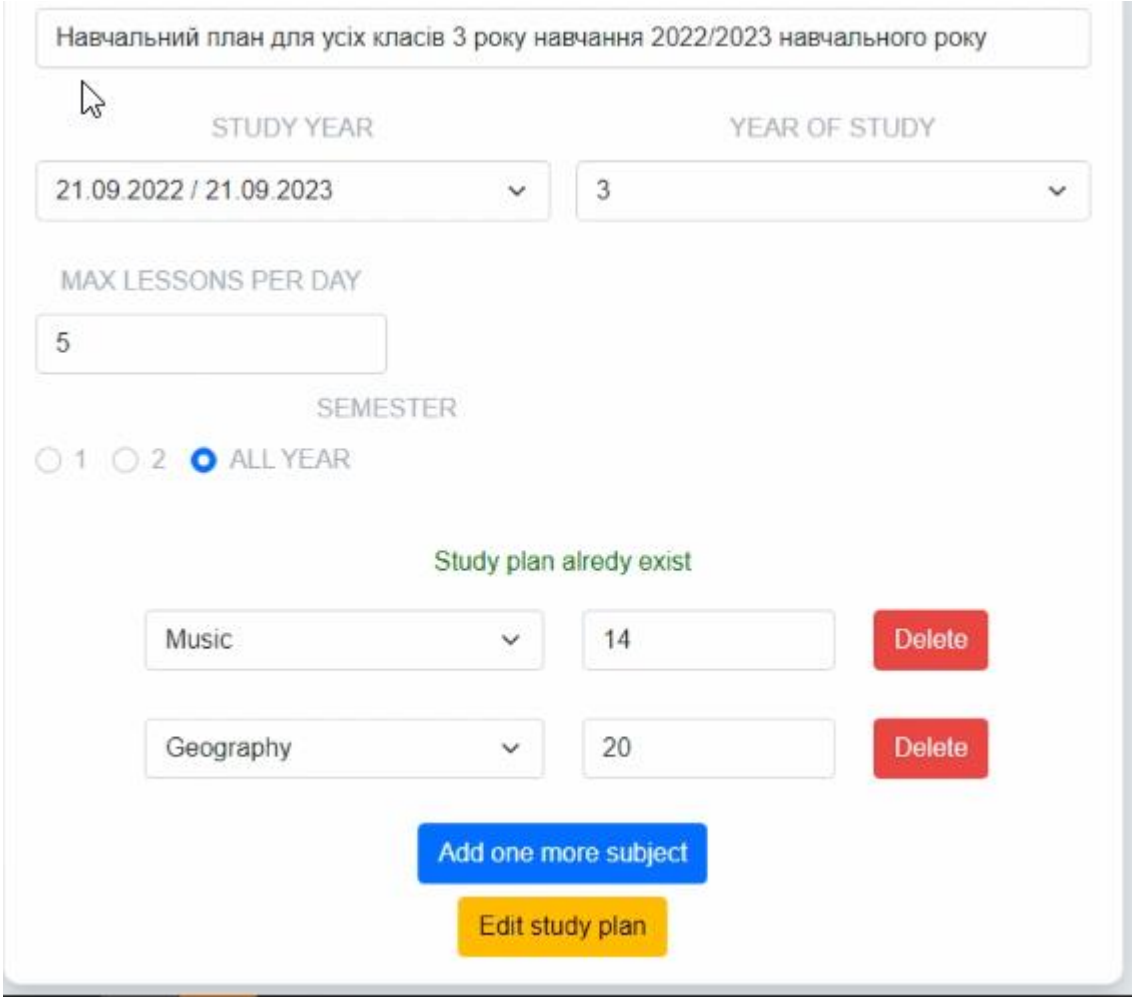


Рис 3.5 – **Вікно зміни навчальних планів у адміністратора**

Джерело: побудовано автором

Управління платформою дистанційного навчання (ПДН) передбачає виконання комплексу адміністративних функцій, спрямованих на забезпечення операційної стабільності та дидактичної когерентності освітнього процесу. Центральною фігурою у цьому процесі є адміністратор, до сфери компетенції якого належить адміністрування навчальних планів та розкладів.

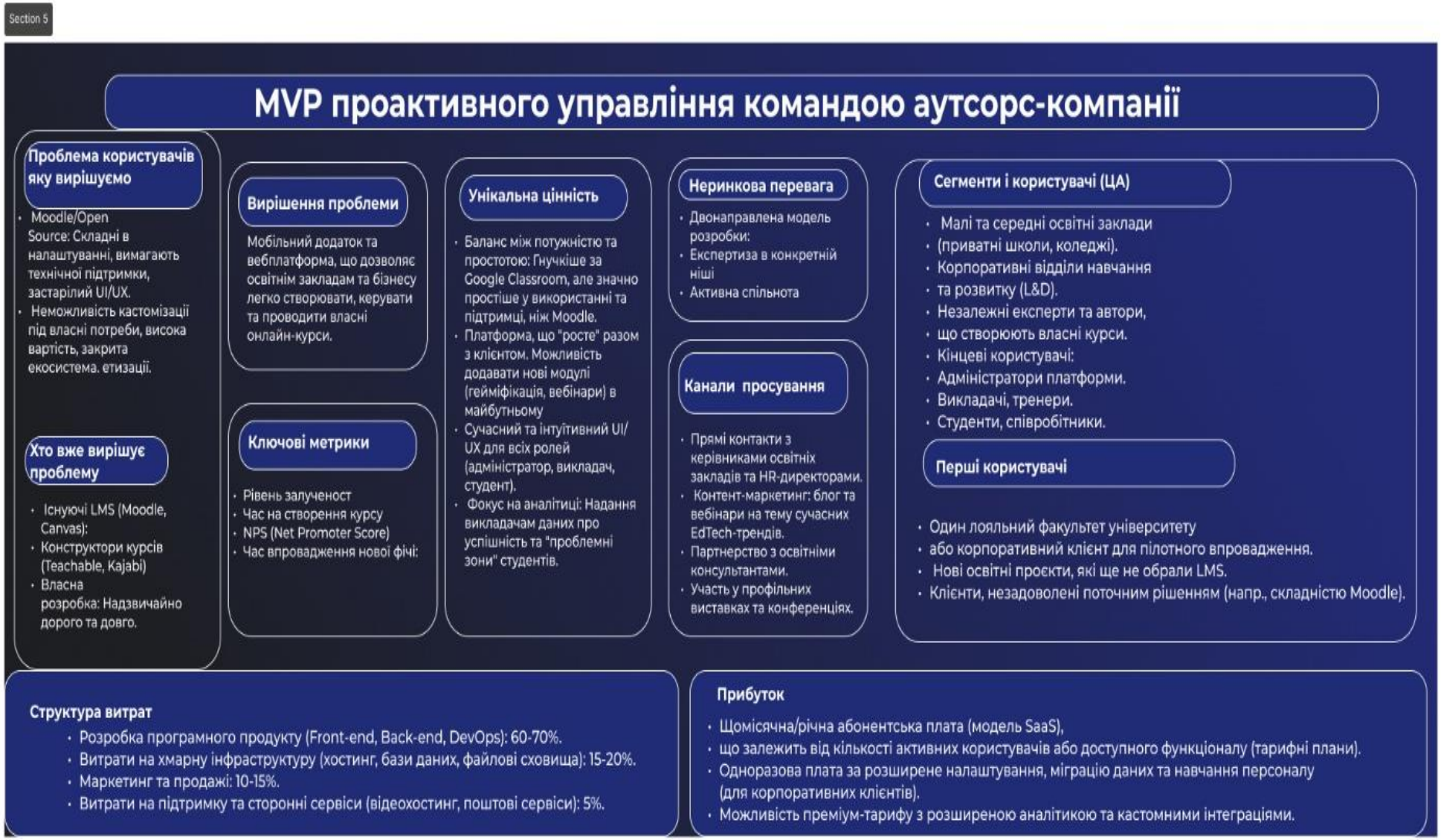


Рис 3.1 – **Карта продукту**

Джерело: побудовано автором

Ключовим завданням є формування та конфігурація розкладу занять, що включає визначення часових параметрів, закріплення викладачів за дисциплінами та розподіл цифрових ресурсів, як-от віртуальні аудиторії. Принципове значення має синхронізація розкладу із затвердженими навчальними планами для гарантування цілісності освітньої програми.

Також до обов'язків адміністратора належить управління навчальними планами: їх створення, модифікація та контроль відповідності державним освітнім стандартам і внутрішнім нормативам закладу (рис. 3.5). Він відповідає за коректність даних про дисципліни, їхню послідовність та обсяг годин.

Адміністратор здійснює оперативне внесення змін до розкладу та планів, що може бути зумовлено заміною викладачів, перерозподілом академічних годин або додаванням нових освітніх компонентів. Важливою частиною цього процесу є своєчасне інформування всіх учасників освітнього процесу про відповідні коригування.

Крім того, адміністратор проводить моніторинг ефективності розкладу та навчальних планів, аналізуючи дані про їх використання та збираючи зворотний зв'язок від викладачів і здобувачів освіти для подальшої оптимізації. Важливою технічною функцією є забезпечення інтеграції ПДН з іншими інформаційними системами закладу (наприклад, SIS), що дозволяє автоматизувати обмін даними та підвищити загальну ефективність управління (рис. 3.6).

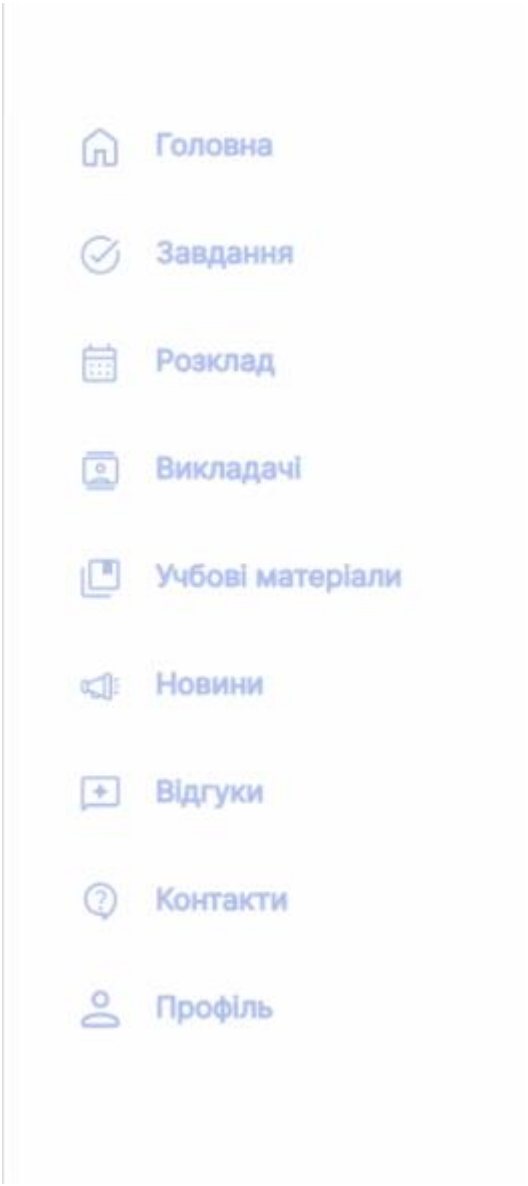


Рис 3.6 – **Головне меню**

Джерело: побудовано автором

Нарешті, адміністратор відповідає за розмежування прав доступу користувачів до модулів розкладу та навчальних планів, забезпечуючи безпеку та конфіденційність даних, а також керує створенням та переглядом навчальних підгруп (рис. 3.7, рис. 3.8).

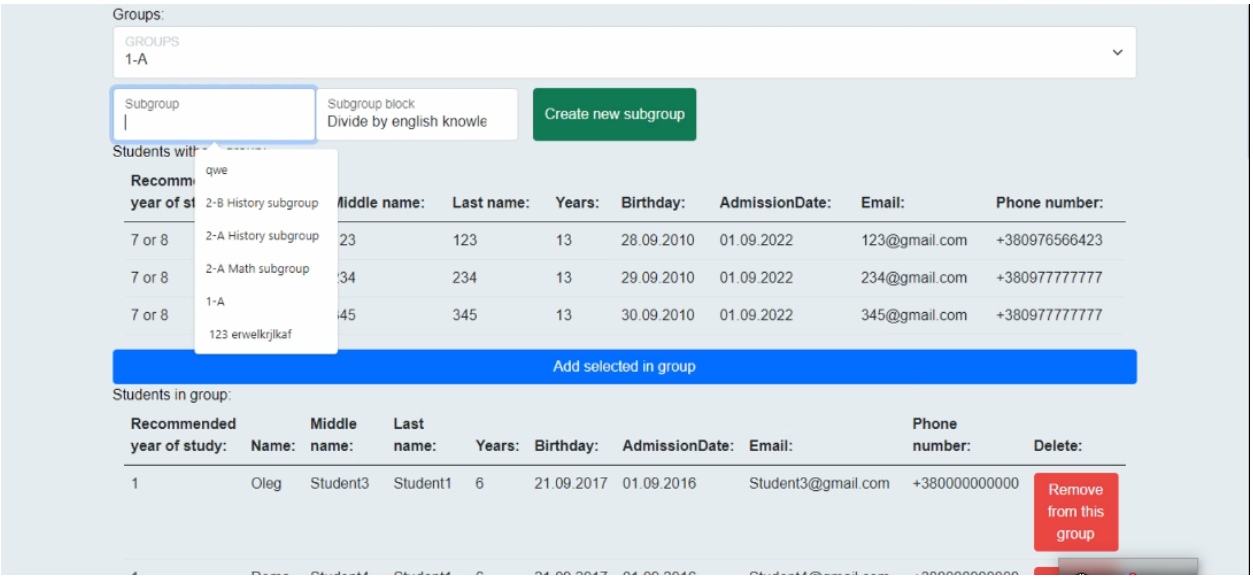


Рис 3.7 – **Створення підгруп**

Джерело: побудовано автором

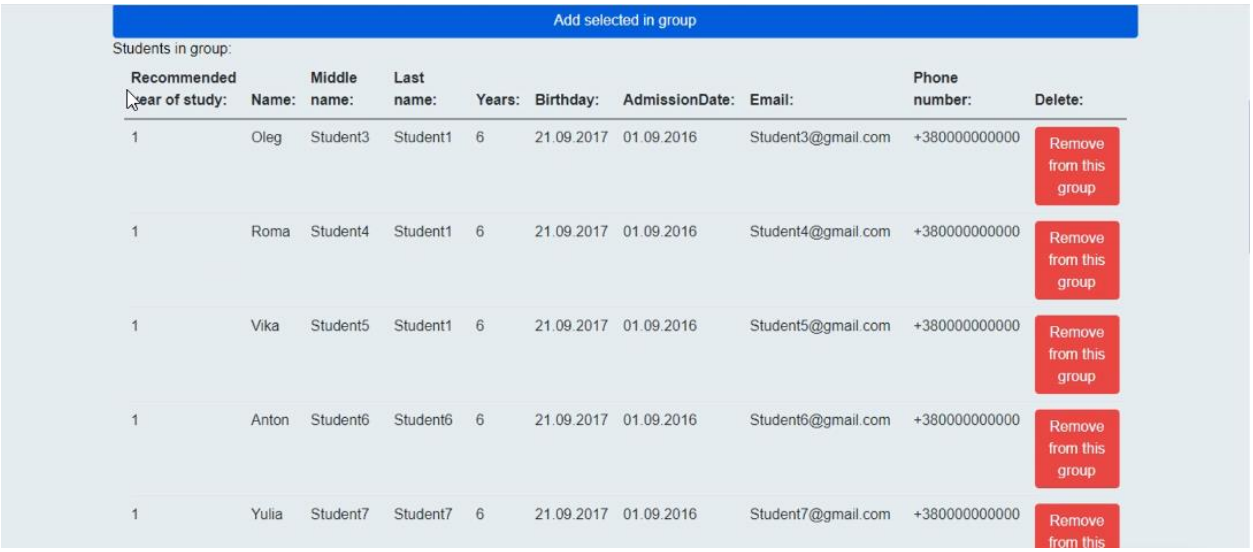


Рис 3.8 – **Перегляд підгруп**

Джерело: побудовано автором

У цифровому середовищі ПДН відбувається суттєве розширення традиційної ролі викладача, яка набуває ознак освітнього дизайнера, модератора комунікації та аналітика (рис. 3.9).

Рис 3.9 – **Головне вікно викладача**

Джерело: побудовано автором

Основною функцією стає дидактичне проєктування навчального курсу: від формулювання освітніх цілей та змісту до вибору релевантних навчальних матеріалів (текстових, мультимедійних, інтерактивних) та методів оцінювання. Викладач структурує курс за модульним принципом та адмініструє доступ до ресурсів.

Центральне місце займає організація навчальної взаємодії. Викладач фасилітує комунікацію зі здобувачами освіти через асинхронні (форуми, системи повідомлень) та синхронні (вебінари, відеоконференції) інструменти. Він надає консультації, роз'яснює складні питання та забезпечує формувальний зворотний зв'язок щодо виконаних завдань (рис. 3.10, рис. 3.11, рис. 3.12).

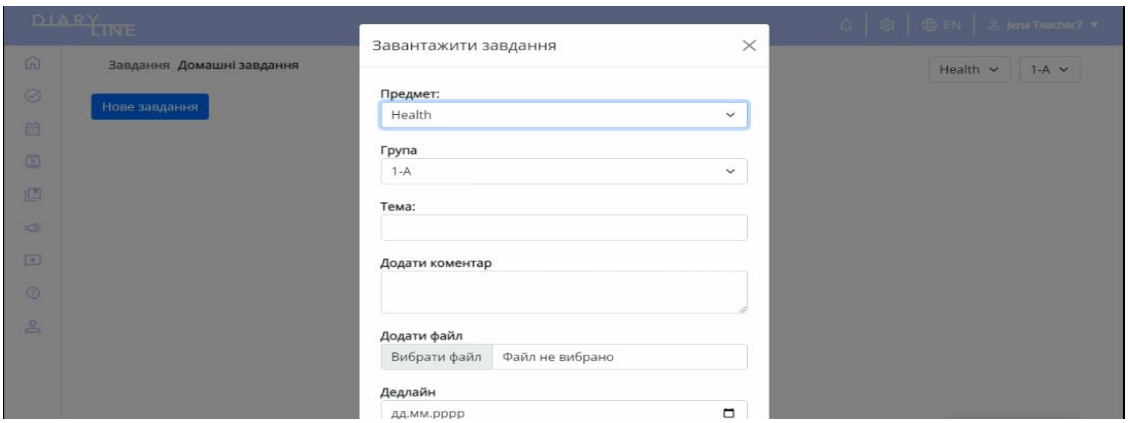


Рис 3.10 – **Створення завдання**

Джерело: побудовано автором

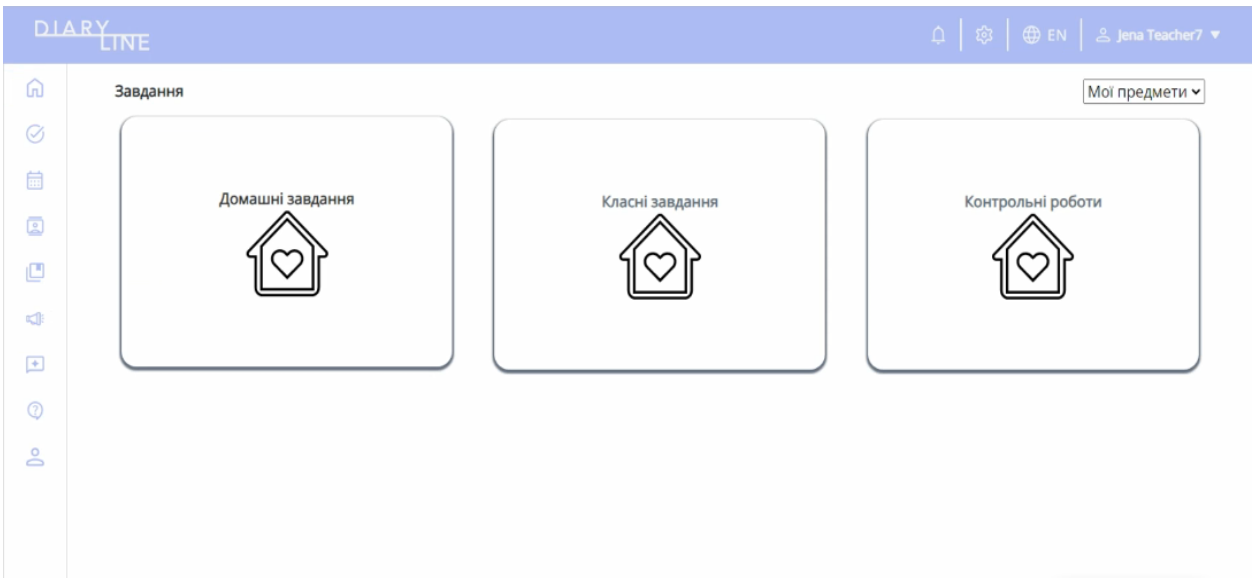


Рис 3.11 – **Обрання типу завдання**

Джерело: побудовано автором

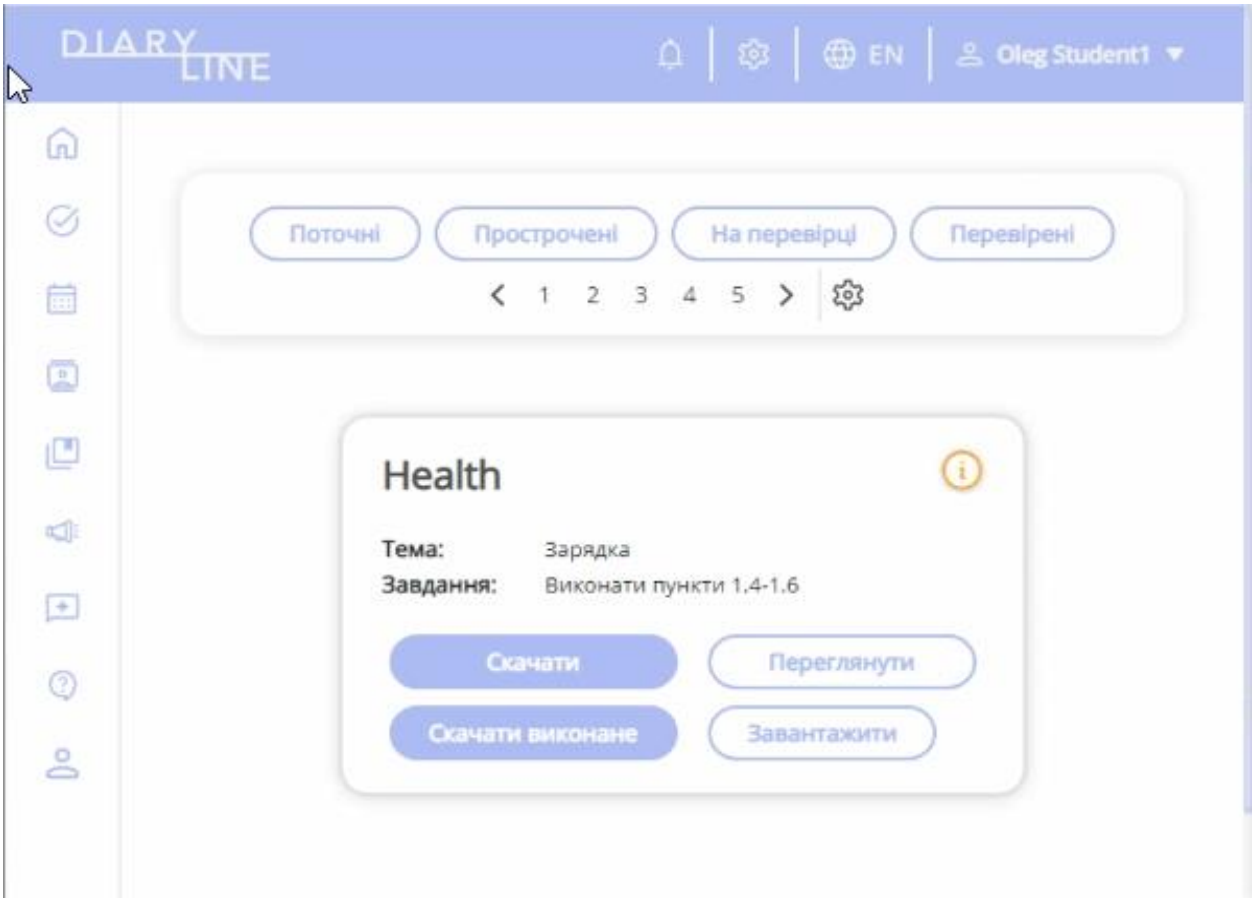


Рис 3.12 – **Виконане завдання**

Джерело: побудовано автором

Викладач здійснює моніторинг освітнього прогресу, використовуючи вбудовані аналітичні інструменти для відстеження активності студентів та результатів оцінювання. Ці дані дозволяють оцінювати ефективність курсу та своєчасно ідентифікувати проблемні зони в навчанні. Налаштування параметрів завдань, таких як вибір підгруп, є частиною цього процесу (рис. 3.13, рис. 3.14).

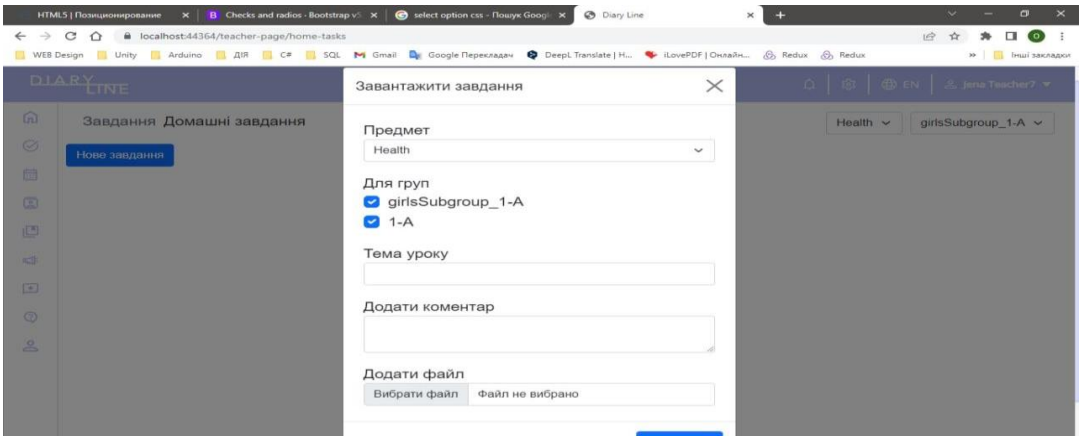


Рис 3.13 – **Налаштування завдання**

Джерело: побудовано автором

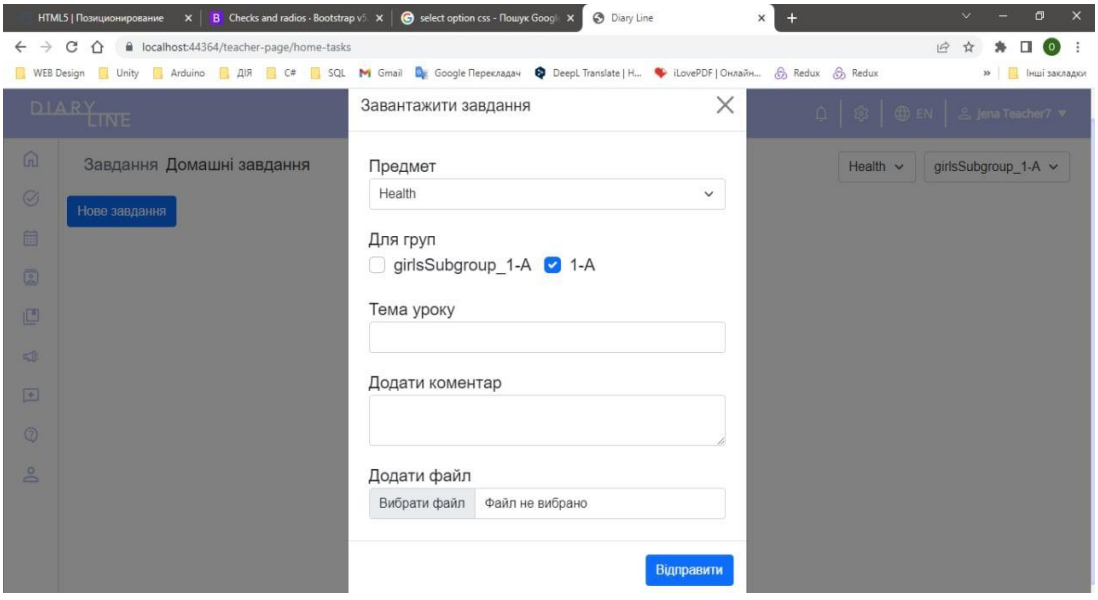


Рис 3.14 – **Вибір підгрупи завдання**

Джерело: побудовано автором

Також роль викладача передбачає участь у розробці нового цифрового контенту та надання експертних відгуків щодо функціональних можливостей самої платформи, що сприяє її вдосконаленню.

В рамках ПДН здобувач освіти (учень) трансформується з пасивного об'єкта в активного суб'єкта освітнього процесу, який наділений інструментами для самостійного управління власною освітньою траєкторією.

Система надає гнучкий доступ до дидактичного контенту, дозволяючи вивчати матеріали у власному темпі, з будь-якого пристрою та в будь-який час. Це сприяє розвитку навичок самоорганізації та відповідальності за результати навчання.

Здобувач освіти є активним учасником комунікаційних процесів, взаємодіючи з викладачами та іншими студентами через форуми та чати для обговорення матеріалів і вирішення спільних завдань.

Цикл зворотної взаємодії реалізується через систему завдань (рис. 3.15, рис. 3.16). Здобувач освіти завантажує результати своєї роботи на платформу, після чого отримує оцінку та детальний фідбек від викладача (рис. 3.19, рис. 3.20). Платформа надає візуалізацію навчального прогресу через інформаційні панелі та структурований вигляд завдань за дисциплінами (рис. 3.17, рис. 3.21, рис. 3.22).

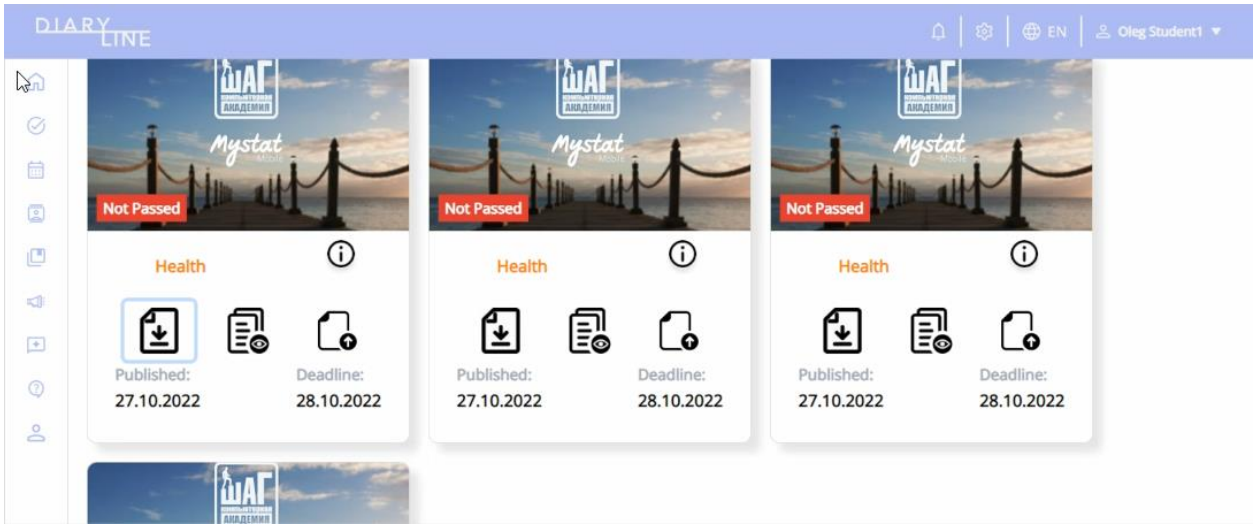


Рис 3.15 – **Вигляд завдань у вікні учня**

Джерело: побудовано автором

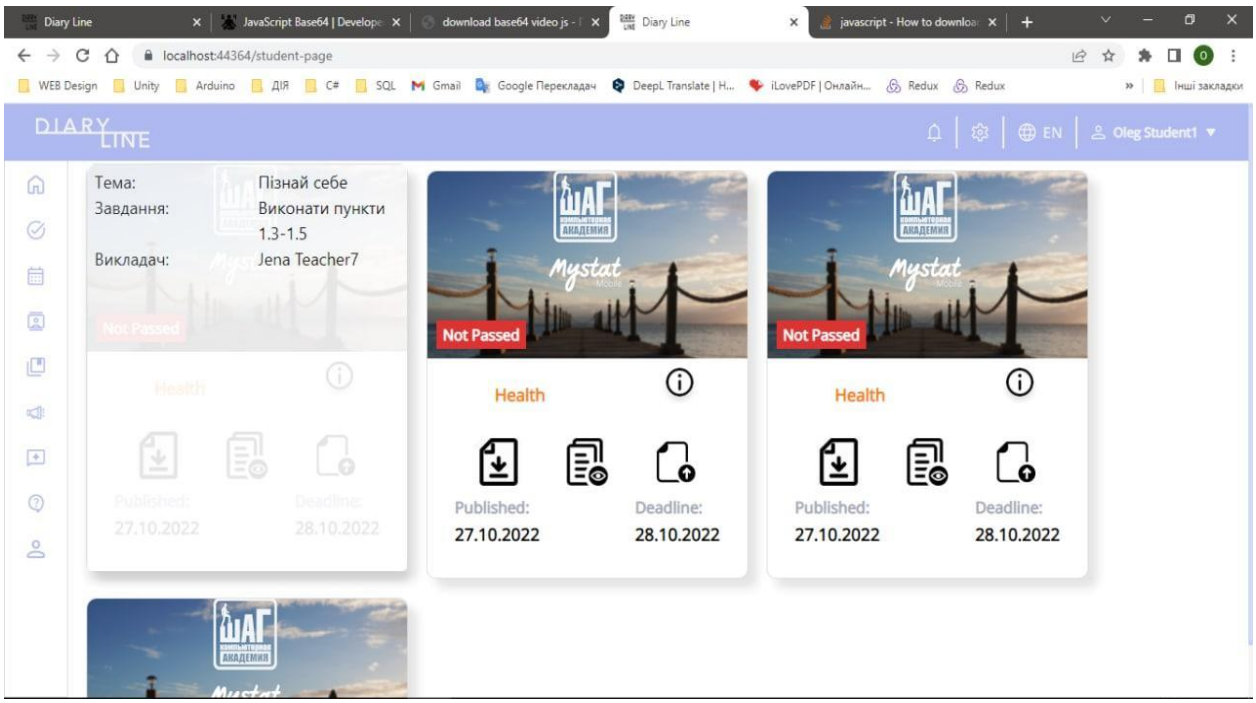


Рис 3.16 – **Картка завдання**

Джерело: побудовано автором

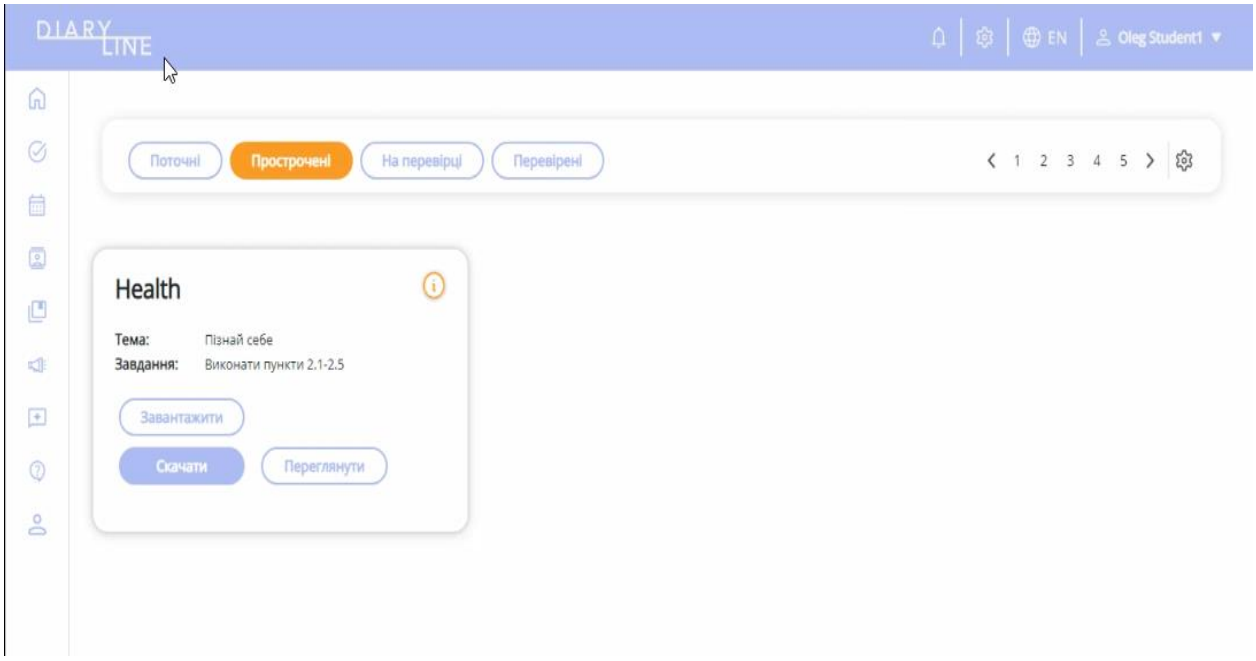


Рис 3.17 – **Завдання у дисципліни за категоріями**

Джерело: побудовано автором

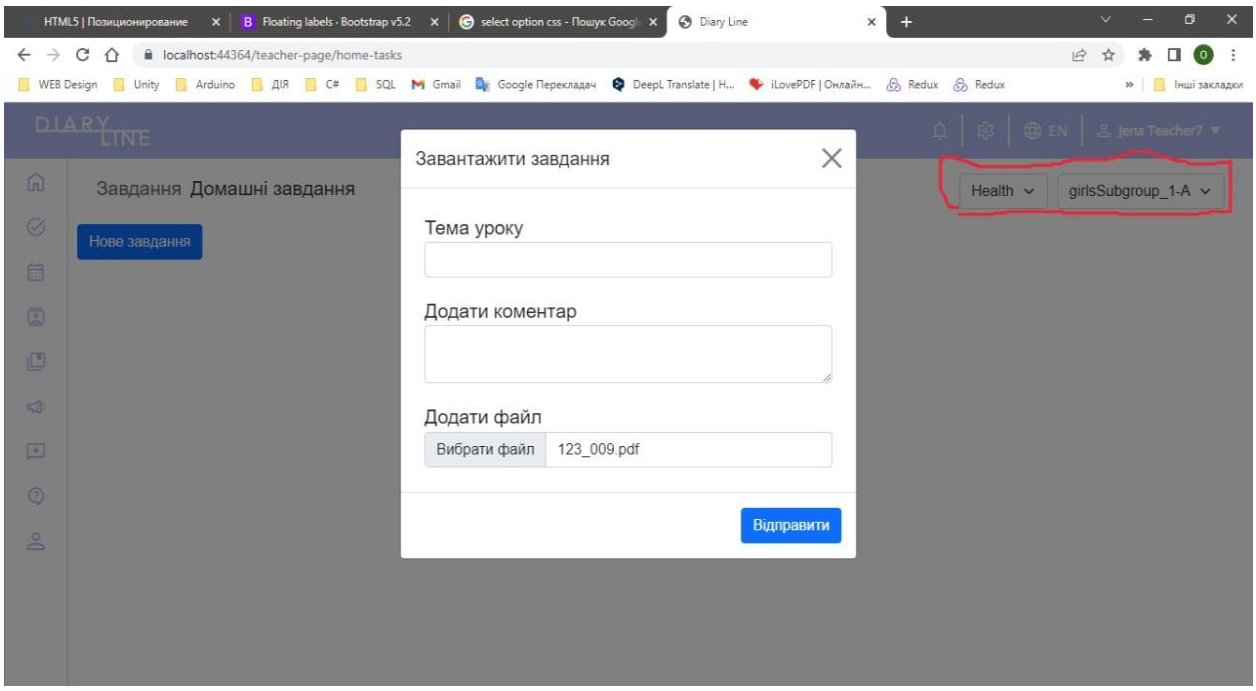


Рис 3.18 – **Додавання завдання для підгрупи**

Джерело: побудовано автором

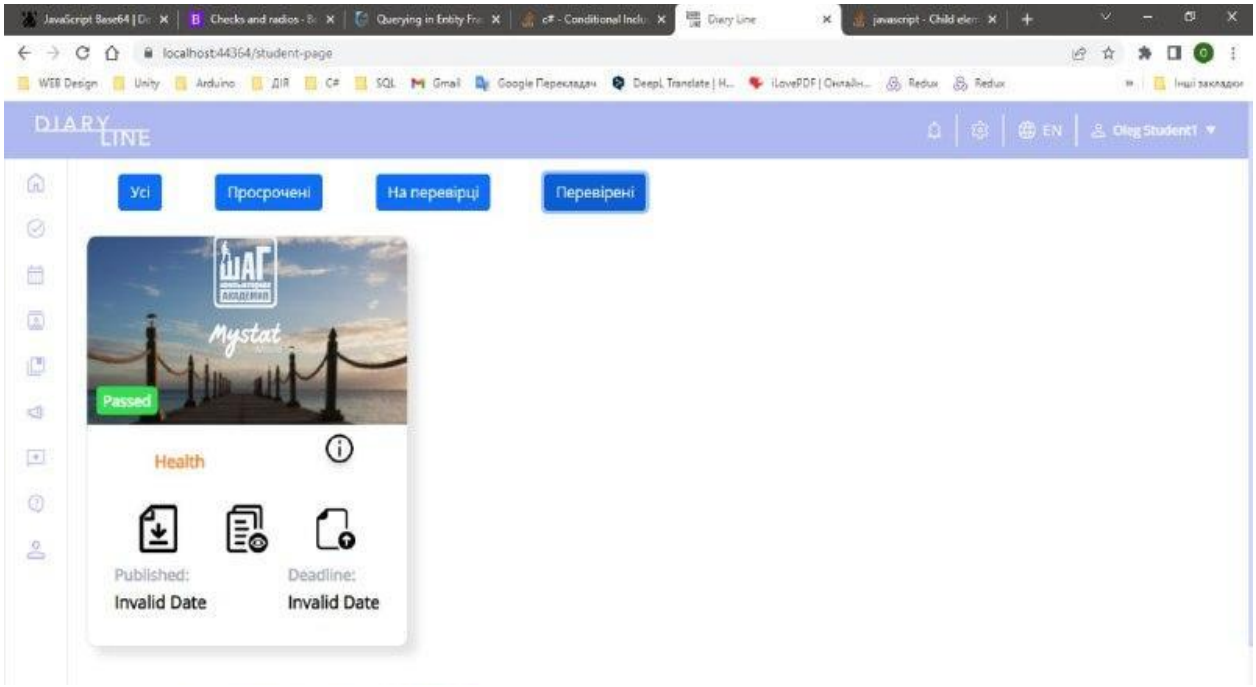


Рис 3.19 – **Вигляд зданого завдання**

Джерело: побудовано автором

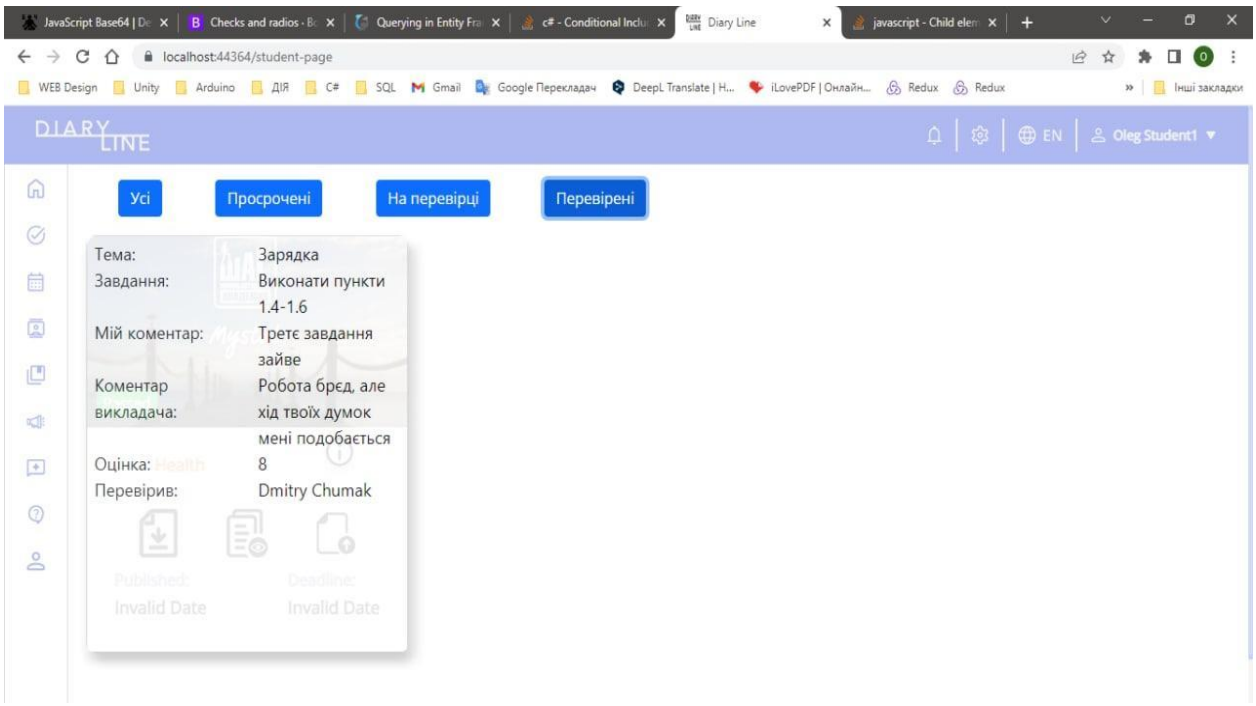


Рис 3.20 – **Відгук викладача на завдання**

Джерело: побудовано автором

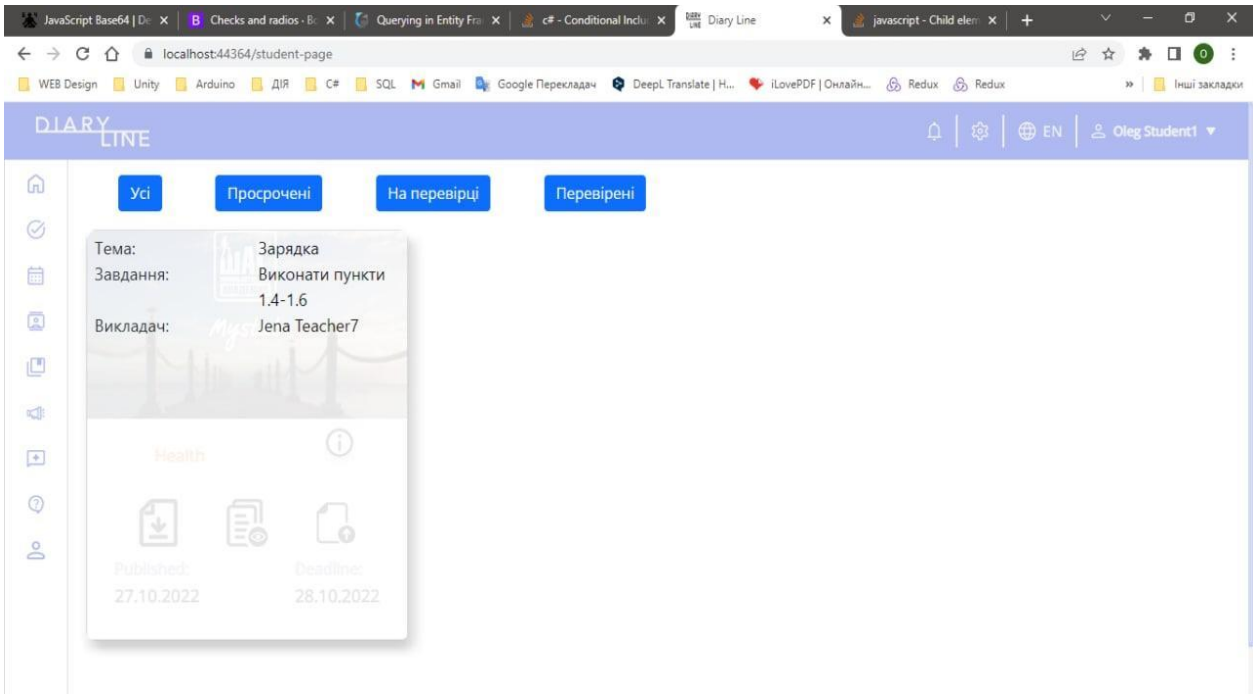


Рис 3.21 – **Картка завдання**

Джерело: побудовано автором

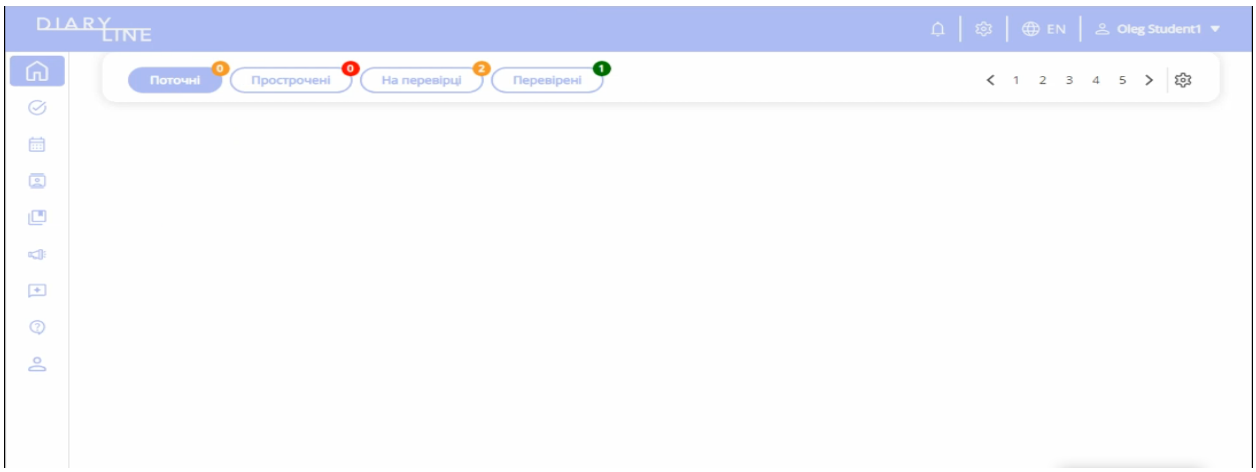


Рис 3.22 – **Інформаційна панель завдань**

Джерело: побудовано автором

Інтерактивні елементи платформи, такі як симуляції та вікторини, підвищують залученість та роблять процес засвоєння знань більш ефективним та цікавим. Таким чином, учень не просто споживає інформацію, а активно конструює власні знання.

Навчальна група в структурі ПДН функціонує як соціально-комунікативне ядро, що перетворює індивідуальний процес навчання на колективну взаємодію та підтримує соціалізацію в цифровому просторі.

Функціонал груп спрямований на організацію колаборативної діяльності, як-от спільне виконання проєктів, що сприяє розвитку навичок командної роботи та критичного мислення. Платформа надає інструменти для спільного доступу до документів та обговорення завдань.

Не менш важливою є функція взаємної підтримки (peer support), де студенти можуть допомагати один одному, ділитися досвідом та спільно вирішувати проблеми, що створює сприятливу навчальну атмосферу та підвищує впевненість (рис. 3.24).

Групи є основою для організації інтерактивних заходів, таких як командні вікторини чи конкурси, що підвищує мотивацію та залученість. Інструменти платформи дозволяють гнучко керувати складом груп, редагувати їх та переглядати розклад у контексті конкретної групи (рис. 3.25, рис. 3.26).

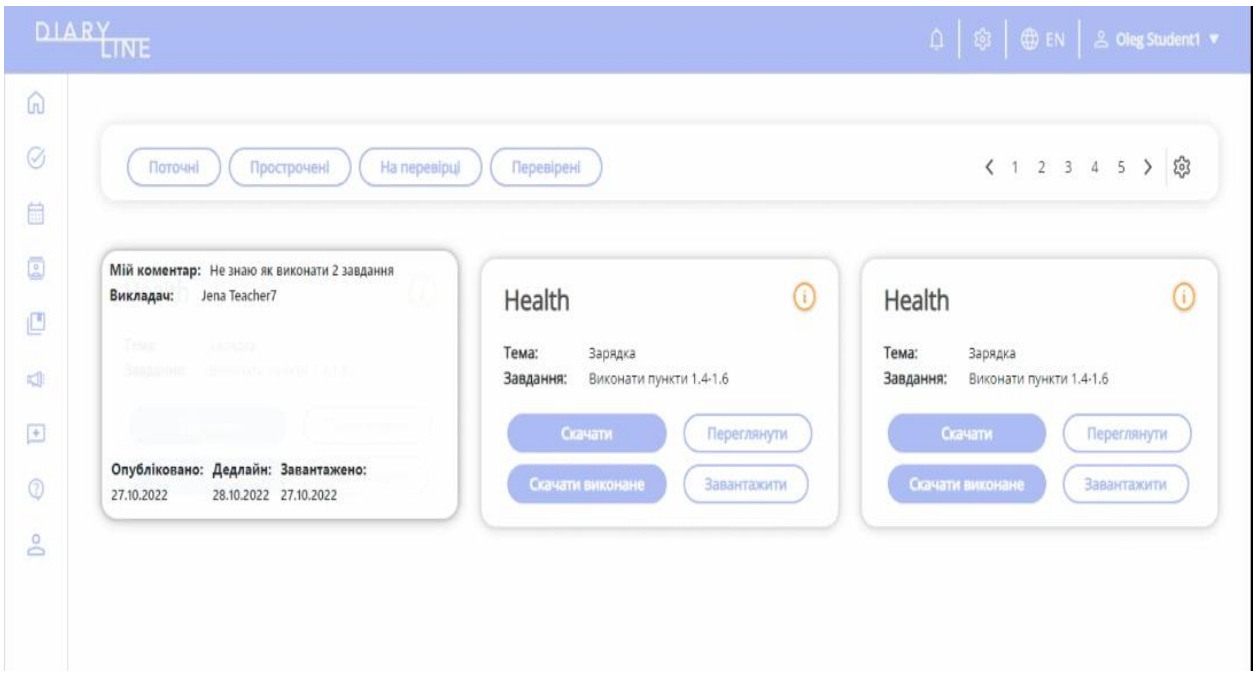


Рис 3.23 – **Вигляд завершених завдань у викладача**

Джерело: побудовано автором

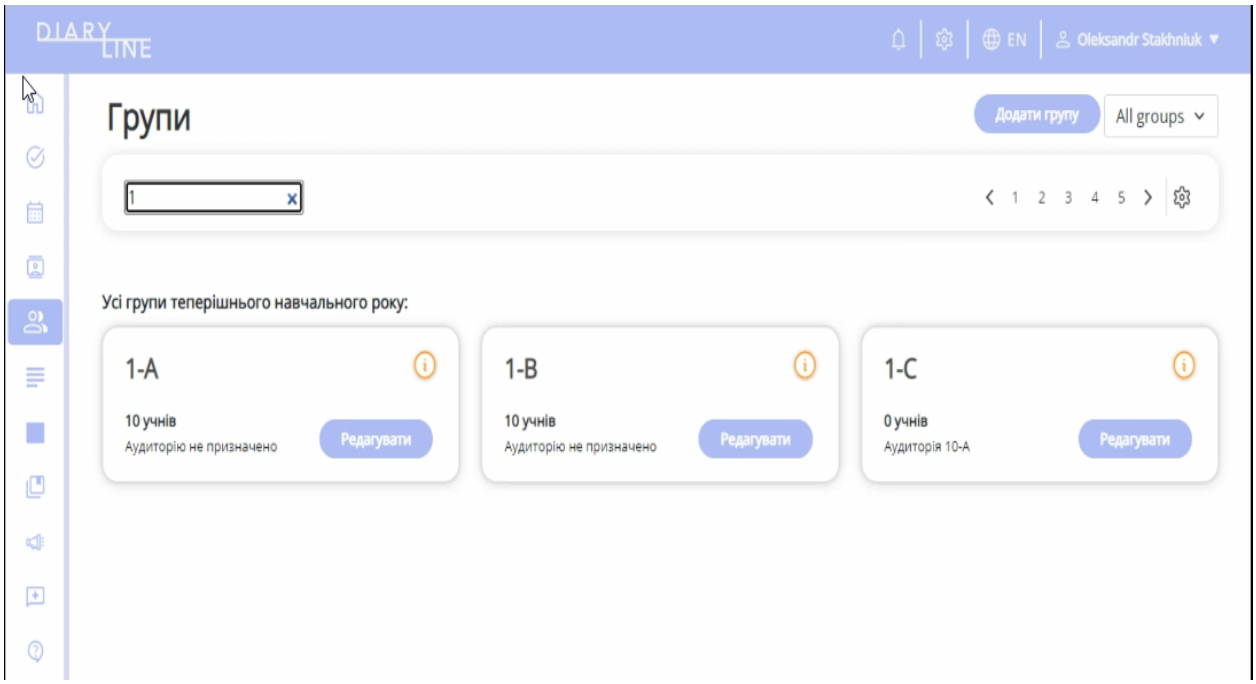


Рис 3.24 – **Групи учнів**

Джерело: побудовано автором

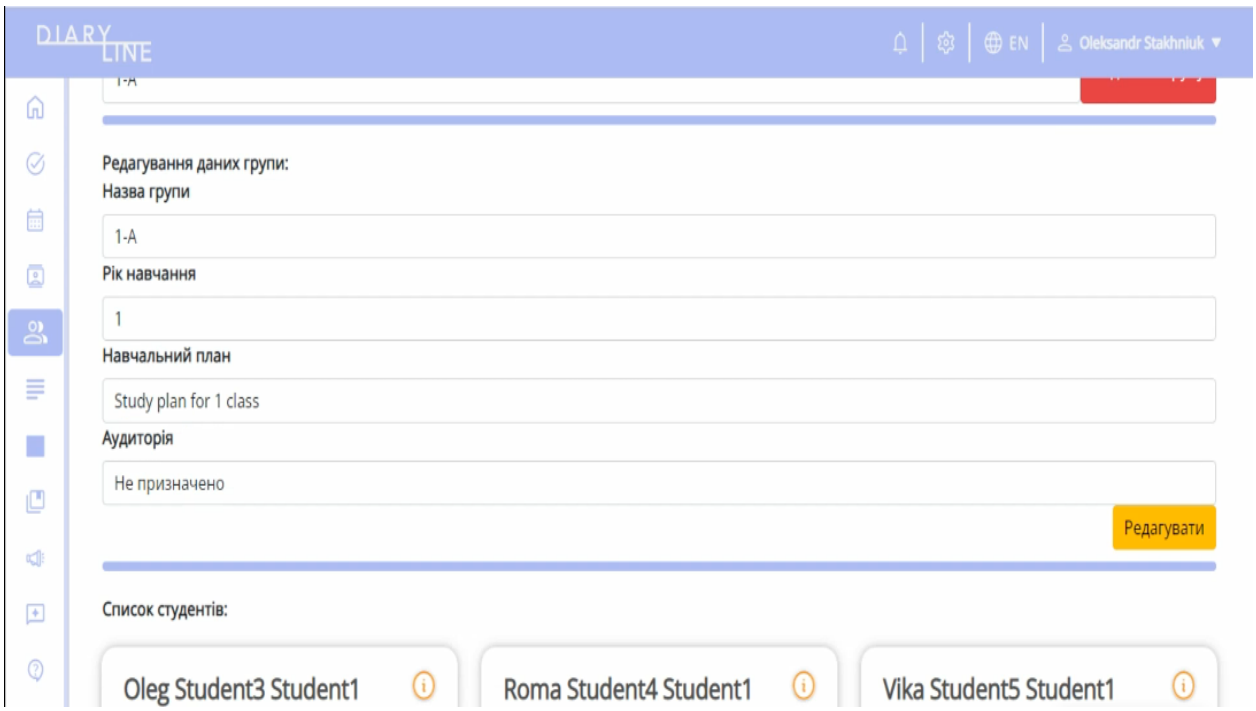


Рис 3.25 – **Редагування групи**

Джерело: побудовано автором

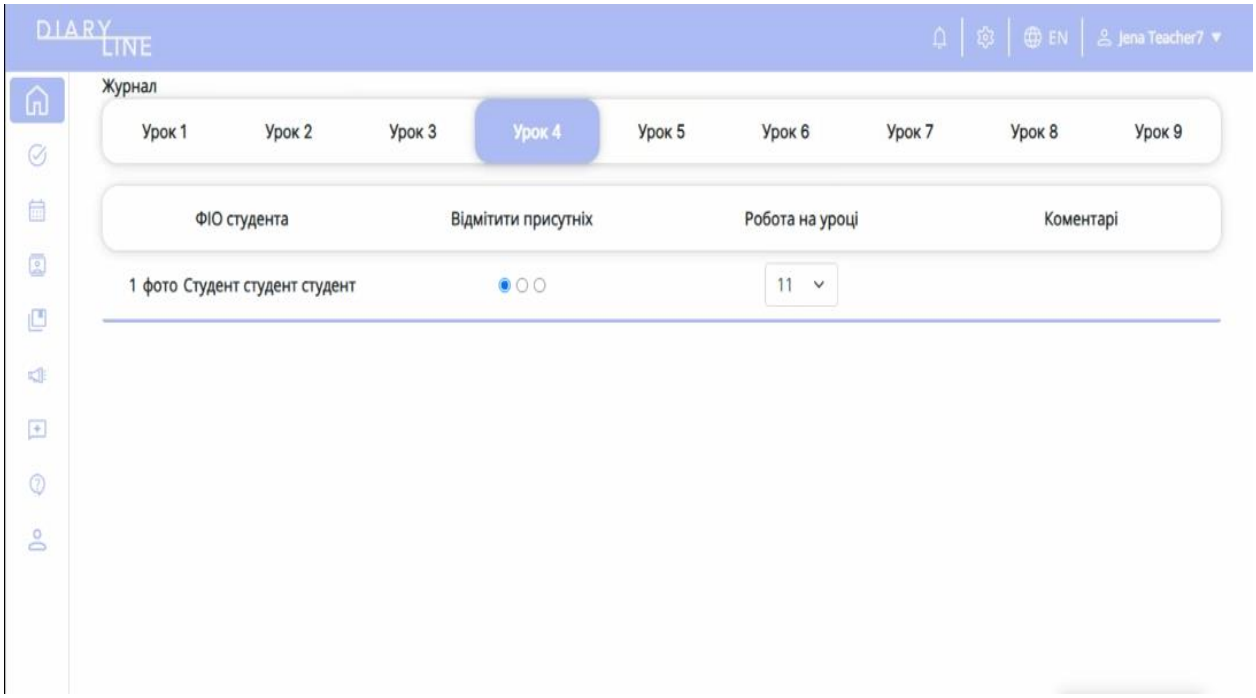


Рис 3.26 – **Перегляд розкладу та редагування групи**

Джерело: побудовано автором

Отже, навчальна група є важливим елементом, що збагачує освітній досвід та сприяє досягненню як індивідуальних, так і колективних навчальних цілей.

3.3. Демонстрація двонаправленого управління на прикладі розробки нової функціональності

Практична ефективність будь-якої управлінської моделі найкраще ілюструється на конкретному прикладі її застосування в реальному циклі розробки. Для демонстрації роботи двонаправленої структури управління було обрано кейс імплементації значущої нової функціональності – інтеграції системи тестування (квізів) до уроків платформи. Цей процес дозволяє наочно простежити рух інформаційних потоків як «зверху-вниз», так і «знизу-вгору», та їхній взаємовплив на кінцевий результат.

Постановка задачі виникла з аналізу ринкових вимог та потреби у підвищенні інтерактивності навчального процесу. Було сформульовано бізнес-мету: надати викладачам інструмент для автоматизованої перевірки знань студентів безпосередньо після проходження уроку, що дозволить підвищити рівень засвоєння матеріалу та зменшити навантаження на викладачів.

Процес розпочався з директивного потоку «зверху-вниз», який трансформував абстрактну бізнес-ідею в конкретні артефакти для команди розробки.

1. Власник продукту (Product Owner) зафіксував загальну вимогу: «Система повинна дозволяти викладачам створювати тести з різними типами питань (одна правильна відповідь, декілька правильних) та прикріплювати їх до уроків. Студенти повинні мати можливість проходити ці тести, а система – автоматично оцінювати результати».

Вимога була декомпозована на епік під назвою «Інтеграція системи квізів». В рамках цього епіку було створено декілька ключових історій користувачів, що описували функціонал з точки зору кінцевих ролей:

- *Як Викладач, я хочу створювати новий квіз, додаючи питання з варіантами відповідей, щоб перевіряти знання студентів.*
- *Як Викладач, я хочу прикріплювати створений квіз до конкретного уроку, щоб він був доступний після вивчення матеріалу.*
- *Як Студент, я хочу проходити квіз, обираючи відповіді на питання, щоб перевірити свої знання.*
- *Як Студент, я хочу бачити свій результат одразу після завершення тесту, щоб отримати миттєвий зворотний зв'язок.*

Паралельно UI/UX-дизайнер розробив інтерактивні прототипи інтерфейсів для створення та проходження квізів. Візуалізація дозволила всім зацікавленим сторонам отримати єдине бачення майбутнього функціоналу та обговорити його до початку написання коду.

На етапі планування спринту команда розробки взяла в роботу перші історії користувачів з епіку. Відбулась подальша технічна декомпозиція:

Створення нових таблиць у базі даних (Quizzes, Questions, Answers, UserQuizAttempts).

Проєктування нових ендпоінтів в API для управління квізами (POST /api/quizzes, GET /api/lessons/{id}/quiz тощо).

Розробка React-компонентів для інтерфейсів створення та проходження тесту.

Кожен розробник брав на себе конкретні технічні завдання, що відображалося в Jira, забезпечуючи повну прозорість процесу виконання.

Найбільш яскраво переваги двонаправленої моделі проявилися на цьому етапі, коли від команди розробки та процесу тестування почала надходити цінна інформація, що вплинула на подальший хід проєкту.

Технічний фідбек: Під час реалізації інтерфейсу для створення квізів, front-end розробник виявив, що обрана на етапі планування бібліотека для створення динамічних форм є надто складною для кастомізації та має проблеми з продуктивністю. Замість того, щоб сліпо слідувати початковому плану, він ініціював обговорення з командою. В результаті спільного

обговорення було прийнято рішення змінити технічний підхід та використати іншу, більш гнучку бібліотеку. Цей зворотний зв'язок «знизу-вгору» дозволив уникнути накопичення технічного боргу та спростив подальшу підтримку коду.

Після завершення розробки першої версії функціоналу QA-інженер розпочав тестування. Було виявлено кілька дефектів, зокрема некоректний підрахунок балів при виборі кількох правильних відповідей. Усі знайдені баги були зареєстровані в Jira з детальним описом кроків для відтворення. Цей потік інформації забезпечив швидке виправлення помилок та підвищив якість кінцевого продукту.

На огляді спринту (демо) команда продемонструвала працюючий функціонал власнику продукту та запрошеному викладачу. Під час демонстрації було отримано критично важливий фідбек щодо незручного інтерфейсу створення питань: виявилося, що процес додавання кожного нового варіанту відповіді вимагав забагато кліків. Ця інформація, отримана «знизу-вгору» від потенційного користувача, призвела до того, що власник продукту негайно створив нову історію користувача в беклозі на покращення UI/UX та надав їй високий пріоритет.

Таким чином, даний кейс демонструє, що двонаправлена модель управління створює адаптивний та самокоригувальний процес. Стратегічні цілі, що спускаються «зверху-вниз», постійно валідуються та уточнюються завдяки технічній експертизі, результатам тестування та відгукам користувачів, що надходять «знизу-вгору», забезпечуючи створення продукту, який є одночасно технічно якісним та справді цінним для кінцевого споживача.

3.4. Тестування та результати роботи

Процес входу в ПДН, що включає в себе автентифікацію та авторизацію, є ключовим елементом безпеки та надійності системи. Тому його ретельне

тестування та валідація є необхідними для забезпечення безпечного та стабільного функціонування платформи.

Тестування автентифікації передбачає перевірку коректності обробки вхідних даних, таких як ім'я користувача та пароль, включаючи тестування алгоритмів хешування та зберігання хешів в базі даних. Важливо перевірити реакцію системи на неправильне введення даних, наприклад, на некоректний формат імені користувача або пароля, чи видається повідомлення про помилку, чи блокується доступ після кількох невдалих спроб входу.

Валідація автентифікації передбачає перевірку збереження даних користувачів в базі даних, включаючи хеші паролів, ім'я користувача, електронну пошту та інші атрибути, а також перевірку безпеки зберігання даних від несанкціонованого доступу. Важливо перевірити коректність генерування та зберігання токенів автентифікації, а також їх дійсність в час звернення до ресурсів.

Тестування авторизації передбачає перевірку того, як система визначає роль користувача та надає йому відповідні права доступу, а також перевірку реакції системи на спроби доступу до ресурсів, що не доступні для даної ролі.

Валідація авторизації передбачає перевірку правильності налаштування прав доступу для кожної ролі користувача, збереження прав доступу в базі даних та безпеку прав доступу від несанкціонованого зміни або видалення.

Окрім тестування автентифікації та авторизації, важливо провести тестування продуктивності, безпеки та юзабіліті системи входу. Тестування продуктивності перевіряє стабільність роботи системи при великому навантаженні користувачів, тестування безпеки перевіряє систему на вразливості до атаки, а тестування юзабіліті перевіряє зручність використання системи входу для користувачів.

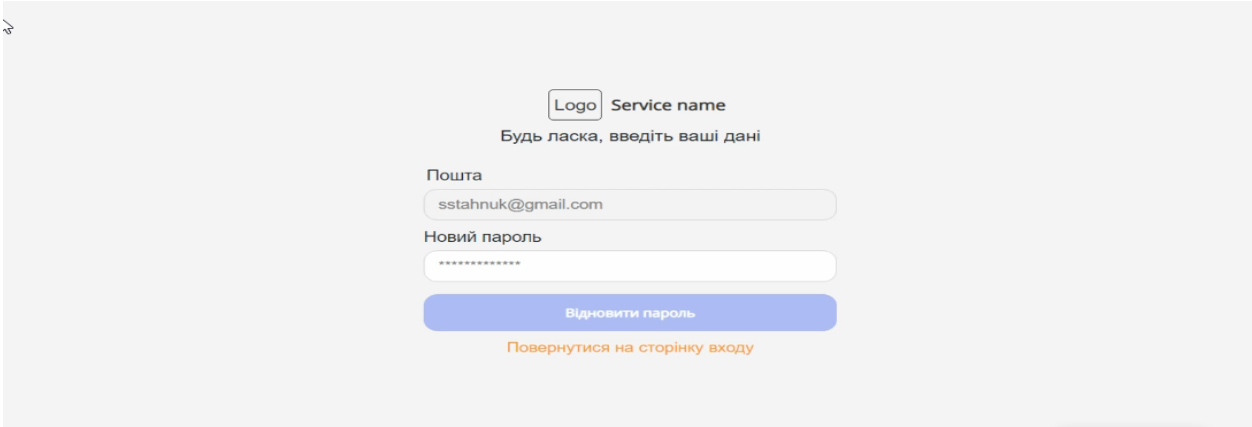


Рис 3.27 – **Вікно входу**

Джерело: побудовано автором

Ретельне тестування та валідація додавання предметів до розкладу в ПДН є критично важливими для забезпечення коректності та ефективності навчального процесу.

Тестування вводу даних передбачає перевірку коректності формату даних, наприклад, чи дата записана в потрібному форматі, чи час введений у 12-годинному або 24-годинному форматі, а також перевірку на наявність обов'язкових полів та неправильних значень.

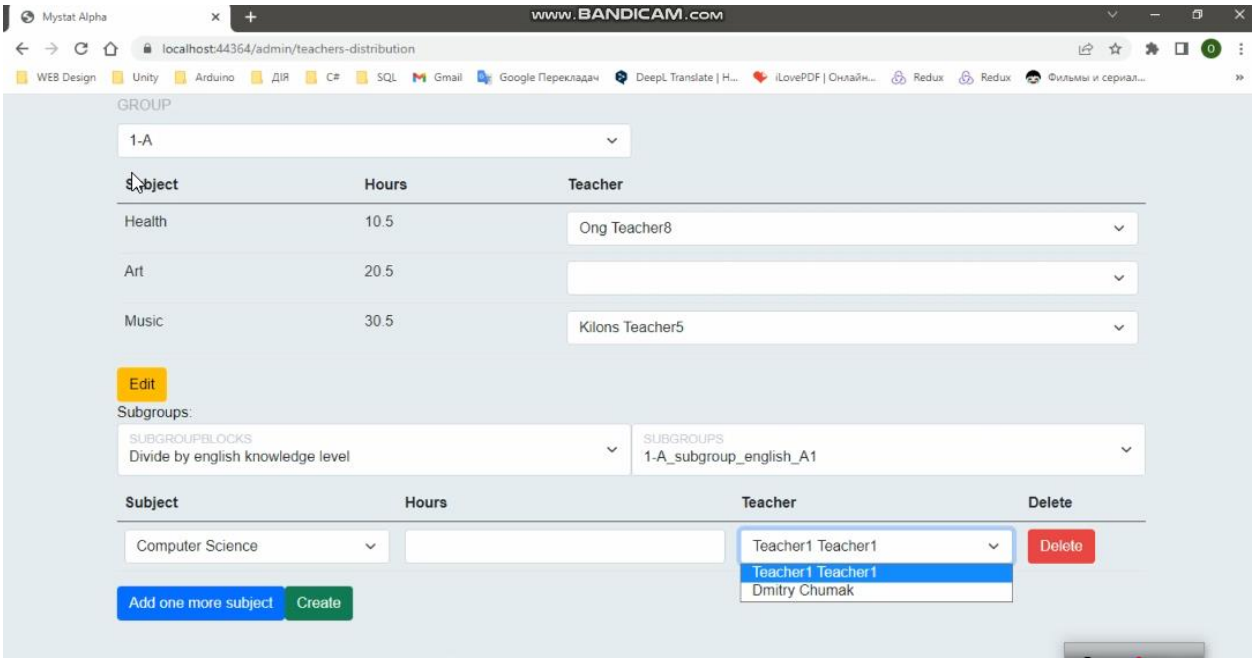


Рис 3.27 – **Вікно додавання предмету**

Джерело: побудовано автором

Тестування логіки додавання передбачає перевірку на наявність конфліктів в розкладі, правильність призначення викладача та групи студентів до предмета, враховуючи наявність прав доступу та відсутність конфліктів з іншими заняттями.

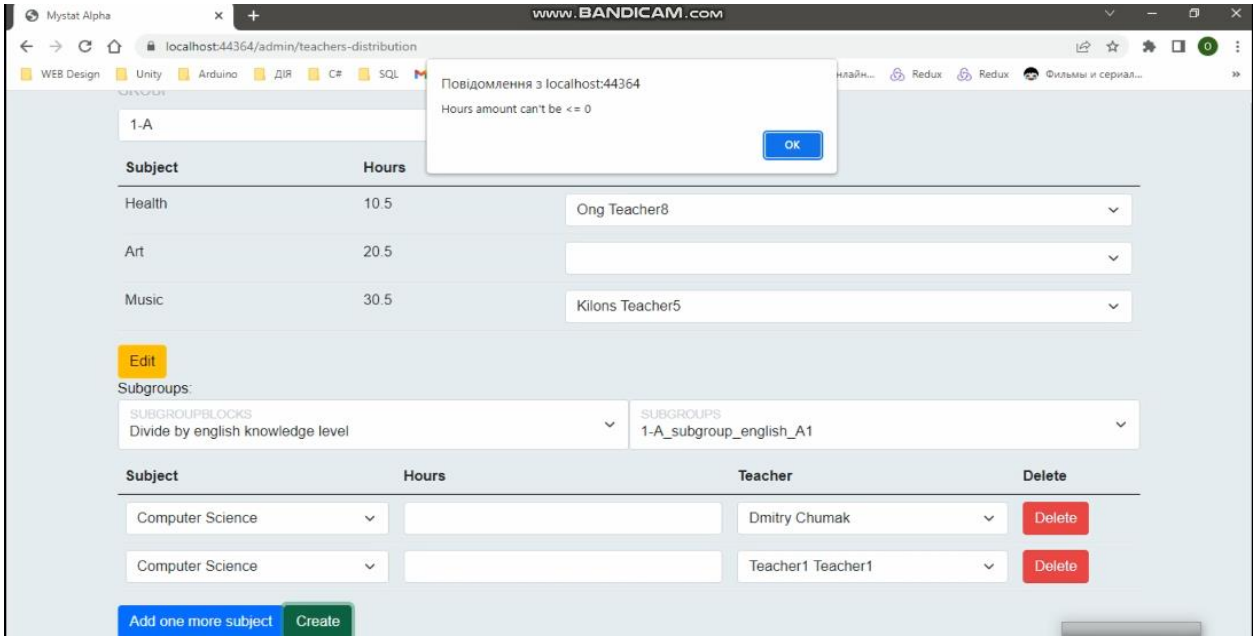


Рис 3.28 – **Перевірка коректності**

Джерело: побудовано автором

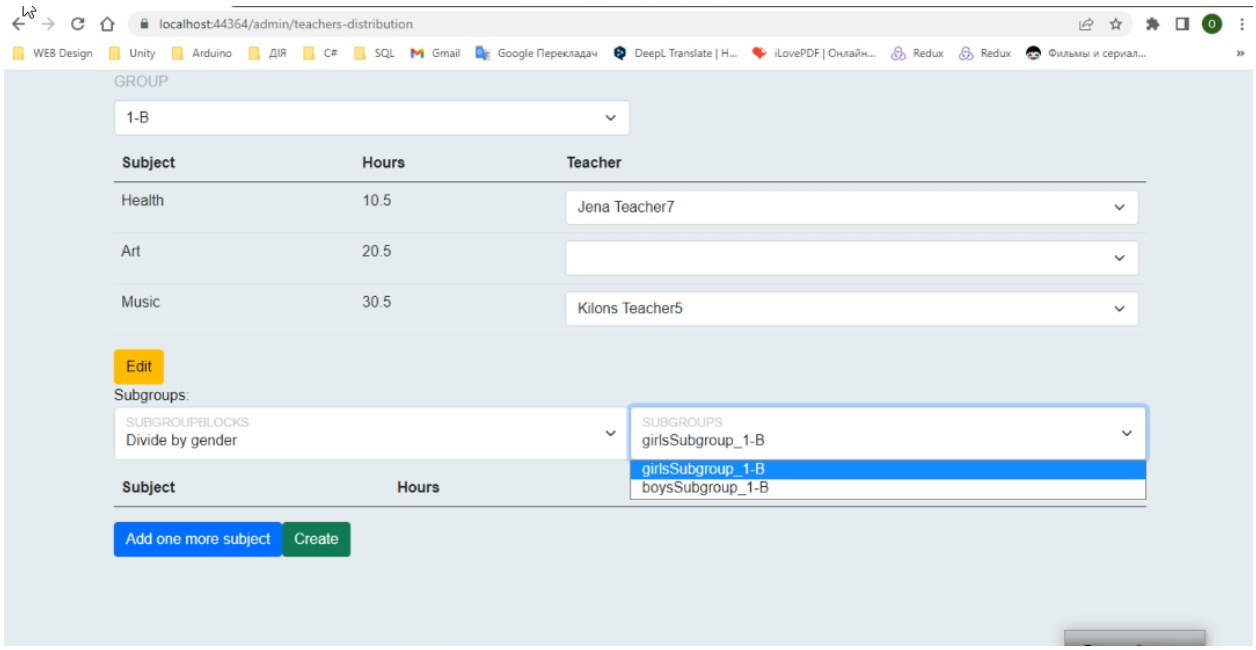


Рис 3.29 – **Вікно обрання групи**

Джерело: побудовано автором

Валідація додавання предметів передбачає перевірку збереження даних в базі даних, коректності відображення інформації в розкладі та інтерфейсі користувачів, а також оновлення розкладу після додавання нового предмета.

Важливо перевірити реакцію системи на неправильно введені дані, наприклад, чи видається повідомлення про помилку, чи заблоковано додавання предмета, а також перевірити обробку конфліктів в розкладі.

Ретельне тестування та валідація процесу додавання уроків до розкладу в ПДН є критично важливими для забезпечення коректності та ефективності навчального процесу. Цей процес вимагає перевірки правильності вводу даних, логіки додавання та збереження інформації в базі даних, а також враховує реакцію системи на помилки та конфлікти.

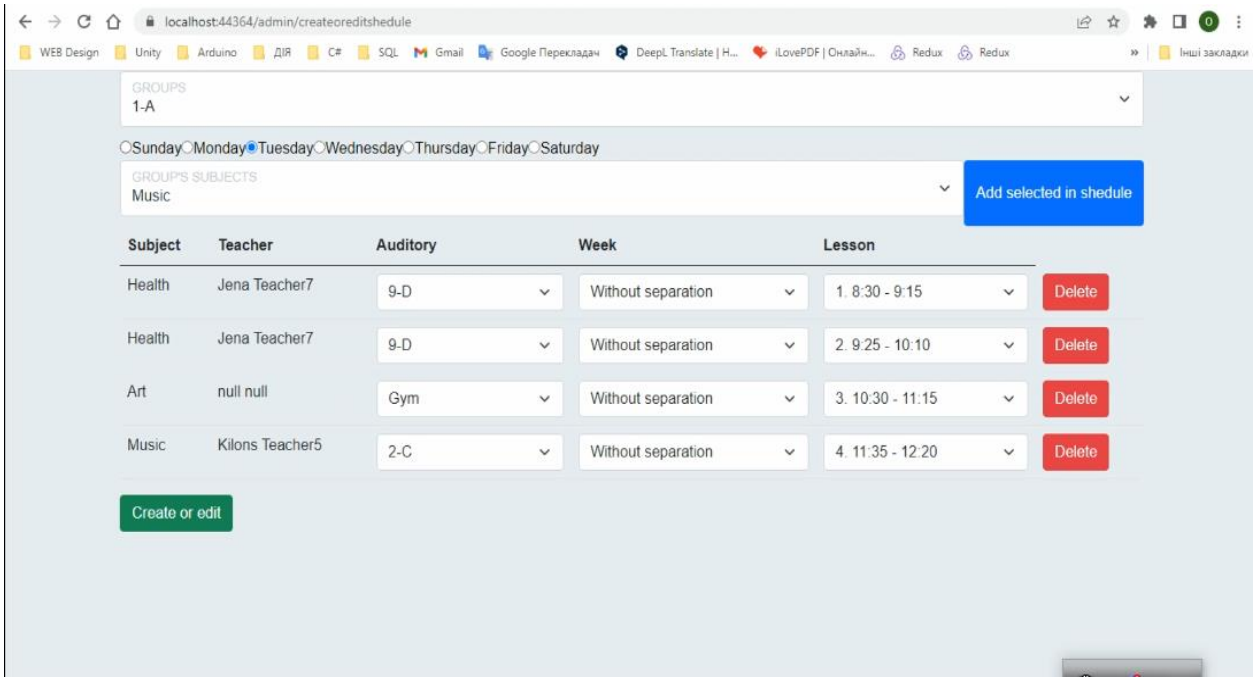


Рис 3.30 – **Вікно редагування розкладу**

Джерело: побудовано автором

Тестування вводу даних передбачає перевірку коректності формату даних, наприклад, чи дата записана в потрібному форматі, чи час введений у 12-годинному або 24-годинному форматі, чи тип уроку вибраний з дозволених варіантів (лекція, практичне заняття, семінар). Важливо перевірити наявність

обов'язкових полів та правильність введених значень, наприклад, чи існує викладач з введеним іменем, чи дата уроку не випадає на вихідний день.

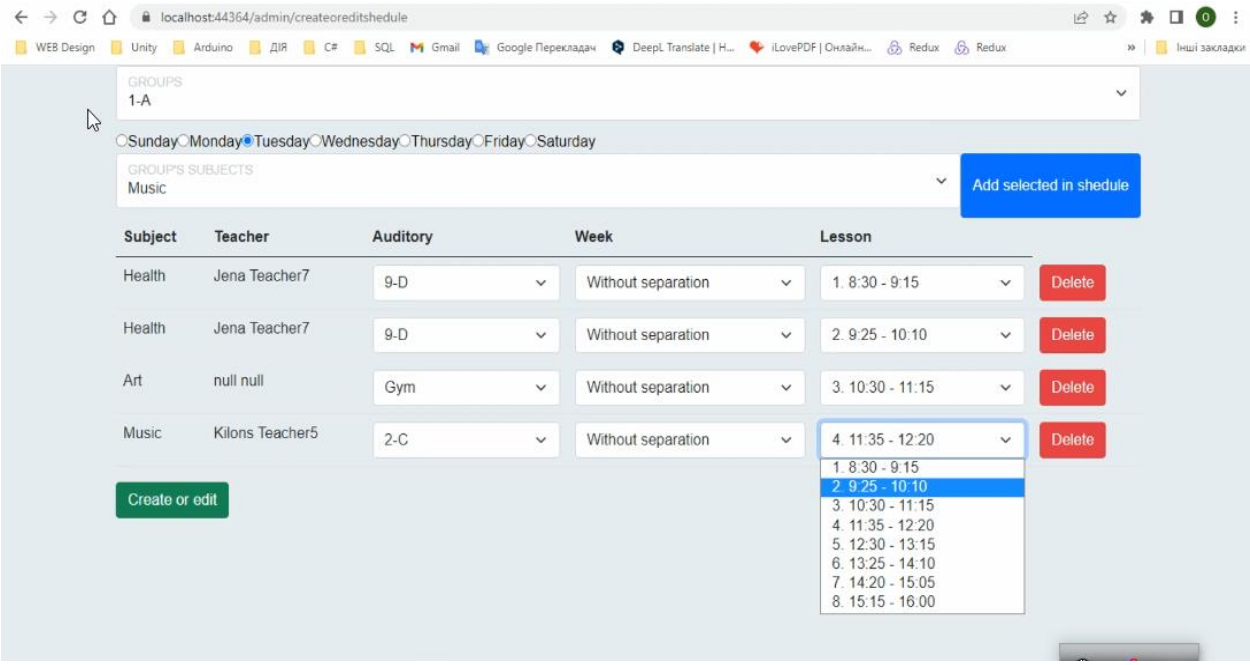


Рис 3.31 – **Зміна часу проведення**

Джерело: побудовано автором

Тестування логіки додавання перевіряє відсутність конфліктів між новим уроком та існуючими заняттями в розкладі викладача або в розкладі студентів. Також важливо перевірити правильність призначення викладача та групи студентів до уроку, враховуючи наявність прав доступу та відсутність конфліктів з іншими заняттями.

Валідація додавання уроків передбачає перевірку збереження даних про новий урок в базі даних, коректності відображення інформації про урок в розкладі та інтерфейсі користувачів, а також оновлення розкладу після додавання нового уроку.

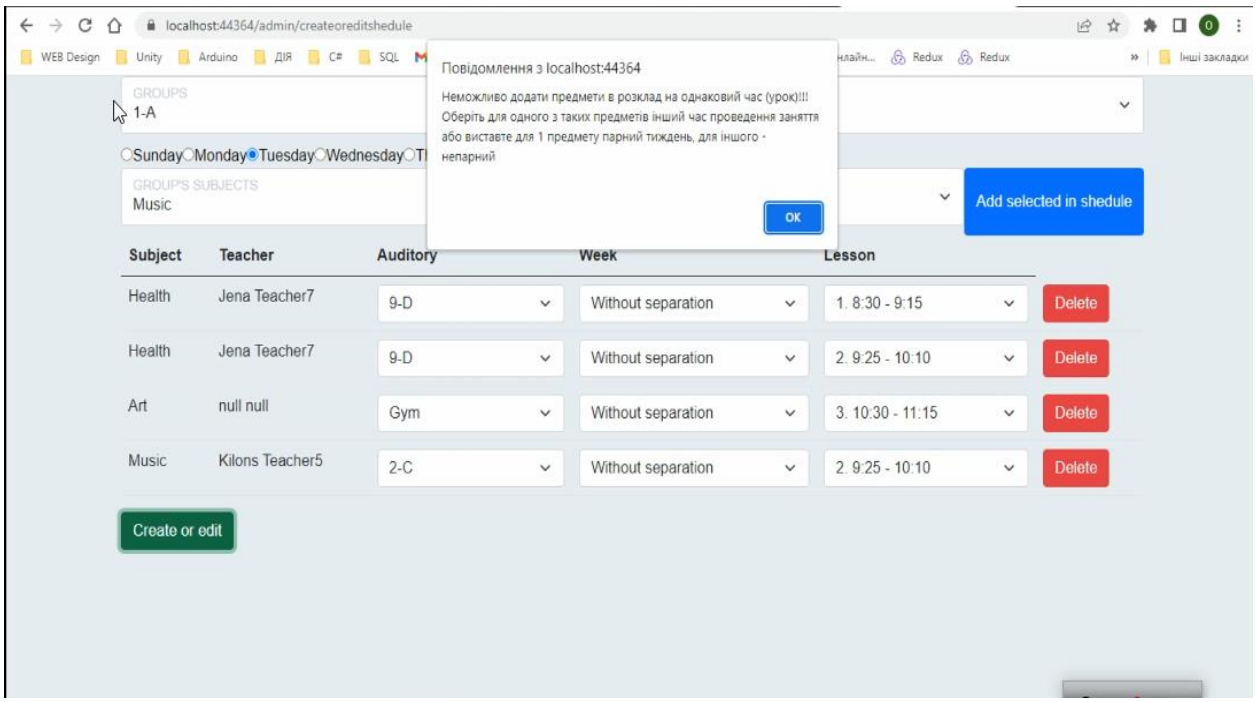


Рис 3.32 – **Конфлікт у розкладі**

Джерело: побудовано автором

Важливо перевірити реакцію системи на неправильно введені дані, наприклад, чи видається повідомлення про помилку, чи заблоковано додавання уроку. Також необхідно перевірити обробку конфліктів в розкладі, чи видається повідомлення про помилку, чи заблоковано додавання уроку.

Окрім основних видів тестування, важливо провести тестування продуктивності, безпеки та юзабіліті системи додавання уроків. Тестування продуктивності перевіряє стабільність роботи системи при великому навантаженні запитів на додавання уроків, тестування безпеки перевіряє систему на вразливості до атаки, а тестування юзабіліті перевіряє зручність використання системи додавання уроків для викладачів та адміністраторів.

Ретельне тестування та валідація додавання уроків до розкладу є ключовими для забезпечення стабільності та ефективності ПДН, а також для забезпечення точного та узгодженого розкладу навчального процесу.

Висновки до Розділу 3

У третьому розділі проведено практичну реалізацію прототипу веборієнтованої платформи дистанційного навчання відповідно до розробленої архітектури та принципів двонаправленої структури управління. Детально описано налаштування середовища розробки, впровадження монорепозиторію, контейнеризацію за допомогою Docker і автоматизацію процесів CI/CD через GitHub Actions. Реалізовано ключові функціональні модулі для адміністратора, викладача та здобувача освіти, зокрема управління навчальними планами та розкладом, створення й оцінювання завдань, аналітику прогресу, роботу з підгрупами та комунікаційні інструменти. Кожний модуль супроводжено пояснювальними рисунками й прикладами інтерфейсу, що відображають специфіку ролей користувачів. Окремо продемонстровано дію двонаправленої моделі на прикладі розробки нової функціональності (інтеграції квізів): простежено формування бізнес-мети, трансляцію її у backlog, технічну декомпозицію, код-рев'ю, тестування й отримання зворотного зв'язку, який вплинув на подальше планування. Це наочно підтвердило ефективність підходу для досягнення балансу між стратегічними цілями та тактичною гнучкістю команди. Третій розділ доводить практичну життєздатність запропонованої моделі управління та її здатність забезпечувати якість і адаптивність складних EdTech-проєктів.

ВИСНОВКИ

У ході виконання даної кваліфікаційної роботи було досягнуто поставленої мети, яка полягала у дослідженні, обґрунтуванні та практичній апробації двонаправленої структури управління проєктом на прикладі розробки веборієнтованого програмного продукту – платформи дистанційного навчання. В результаті проведеної роботи було вирішено всі поставлені завдання та отримано наступні наукові та практичні результати.

1. Проведено системний аналіз існуючих методологій управління проєктами. Було встановлено, що класичні каскадні моделі, хоча й забезпечують передбачуваність, є надто ригідними для сучасних EdTech-проєктів з високим рівнем невизначеності. Водночас суто гнучкі (Agile) підходи, забезпечуючи тактичну адаптивність, можуть призводити до втрати довгострокового стратегічного бачення. Це обґрунтувало необхідність розробки гібридної моделі, що поєднує переваги обох парадигм.

2. Сформульовано та теоретично обґрунтовано концепцію двонаправленої структури управління проєктом. Запропонована модель визначається як інтегрована система, що поєднує стратегічне планування «зверху-вниз» (від бізнес-цілей до технічних завдань) з оперативним емпіричним зворотним зв'язком «знизу-вгору» (від результатів розробки та тестування до коригування стратегії). Доведено, що такий підхід створює замкнений контур безперервного вдосконалення, забезпечуючи баланс між довгостроковими цілями та гнучкістю реалізації.

3. Здійснено глибокий аналіз предметної області платформ дистанційного навчання. На основі дослідження ринку та провідних рішень (Moodle, Coursera, Google Classroom) було ідентифіковано ключові функціональні блоки та визначено потреби основних ролей користувачів (Студент, Викладач, Адміністратор). Це дозволило сформулювати деталізовані функціональні та нефункціональні вимоги до програмного продукту, зокрема до його масштабованості, безпеки та юзабіліті.

4. Спроєктовано стійку та гнучку архітектуру програмного продукту. В якості архітектурного стилю було обґрунтовано обрано модульний моноліт, який поєднує простоту розгортання монолітної архітектури з логічним розмежуванням та слабкою зв'язаністю компонентів. Було розроблено нормалізовану реляційну модель бази даних та спроєктовано RESTful API для взаємодії між клієнтською та серверною частинами.

5. Обґрунтовано вибір сучасного та релевантного технологічного стеку. На основі порівняльного аналізу було обрано Node.js для серверної частини завдяки його асинхронній природі та можливості використання JavaScript; бібліотеку React для клієнтської частини через її компонентний підхід та розвинену екосистему; та СУБД PostgreSQL для забезпечення цілісності та надійності даних.

6. Розроблено детальний план організації робочих процесів. Було визначено та налаштовано інструментальний комплекс (Jira, Git/GitHub, Figma), що забезпечує практичну реалізацію двонаправленої моделі. Описано життєвий цикл задачі, який наочно ілюструє взаємодію інформаційних потоків, та формалізовано механізми збору зворотного зв'язку через практики Scrum (демо, ретроспективи).

7. Здійснено практичну реалізацію програмного прототипу платформи. Було розроблено ключові функціональні модулі, що підтверджують життєздатність обраних архітектурних та технологічних рішень. Реалізовано систему автентифікації та авторизації, функціонал управління курсами для викладачів та механізм проходження навчальних матеріалів для студентів.

8. Ефективність двонаправленої моделі було доведено на практичному кейсі. На прикладі розробки системи тестування (квізів) було продемонстровано, як зворотний зв'язок «знизу-вгору» (технічні пропозиції, результати тестування, відгуки з демонстрації) безпосередньо вплинув на коригування технічних рішень та пріоритетів у беклозі, що дозволило уникнути технічного боргу та покращити користувацький досвід.

9. Проведено комплексне тестування прототипу. Результати тестування підтвердили працездатність реалізованого функціоналу та його відповідність сформульованим на етапі аналізу вимогам, що свідчить про успішність практичної частини роботи.

Було розроблено не лише програмний продукт, а й запропоновано та валідовано цілісну методологію управління його створенням. Двонаправлена структура управління проєктом показала себе як ефективний підхід для розробки складних веборієнтованих систем в умовах змінних вимог, що забезпечує синергію між стратегічним баченням та тактичною гнучкістю команди. Отримані результати мають як теоретичну, так і практичну цінність та можуть бути використані як основа для подальшого розвитку платформи та впровадження запропонованої моделі управління в реальних IT-проєктах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Project Management Institute. A Guide to the Project Management Body of Knowledge (PMBOK® Guide). 7-th ed. – Newtown Square, PA: PMI, 2021. – 496 p.
2. Schwaber K., Sutherland J. The Scrum Guide: The definitive guide to Scrum: the rules of the game / K. Schwaber, J. Sutherland. – 2020. – 40 p.
3. Ries E. The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses. Revised ed. – New York: Crown Business, 2021. – 336 p.
4. Bass L., Weber I., Zhu L. DevOps: A Software Architect's Perspective. – Boston: Addison-Wesley, 2020. – 312 p.
5. Fowler M. Patterns of Enterprise Application Architecture. 2nd ed. – Boston: Addison-Wesley, 2021. – 448 p.
6. Kniberg H., Skarin M. Lean from the Trenches: Managing Agile Projects. – Stockholm: Pragmatic Bookshelf, 2020. – 256 p.
7. Sommerville I. Software Engineering. 10th ed. – Harlow: Pearson, 2021. – 832 p.
8. Humble J., Farley D. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. 2nd ed. – Boston: Addison-Wesley, 2020. – 432 p.
9. Richardson C. Microservices Patterns: With examples in Java / C. Richardson. – Shelter Island: Manning, 2020. – 352 p.
10. Newman S. Building Microservices: Designing Fine-Grained Systems. 3rd ed. – Sebastopol: O'Reilly, 2022. – 384 p.
11. Laudon K. C., Laudon J. P. Management Information Systems: Managing the Digital Firm. 16th ed. – Harlow: Pearson, 2021. – 720 p.
12. Chen H., Chiang R. H. L., Storey V. C. Business Intelligence and Analytics: from big data to big impact / H. Chen, R. H. L. Chiang, V. C. Storey // MIS Quarterly. – 2020. – Vol. 45, № 2. – P. 471–482.

13. O’Reilly M. Building Real-Time Data Pipelines for Web Applications / M. O’Reilly // Data Engineering Review. – 2022. – Vol. 4. – P. 77–89.

14. AWS Well-Architected Framework. – Amazon Web Services, 2021. – 84 p.

15. Bass L. Microservices and Service Mesh Patterns / L. Bass // IEEE Software. – 2021. – Vol. 38, № 1. – P. 54–63.

16. Humphrey W. S. Managing the Software Process: Classic insights for modern teams / W. S. Humphrey. – Boston: Addison-Wesley, 2020. – 256 p.

17. Beck K. Test-Driven Development: By Example. – Boston: Addison-Wesley, 2020. – 220 p.

18. Sculley D., Holt G. Hidden technical debt in machine learning systems / D. Sculley et al. // Communications of the ACM. – 2020. – Vol. 63, № 10. – P. 56–65.

19. Feurer M., Hutter F. Automated Machine Learning: state of the art and open challenges / M. Feurer, F. Hutter // Machine Learning. – 2020. – Vol. 110. – P. 1–33.

20. Kairouz P., et al. Advances and open problems in federated learning / P. Kairouz et al. // Foundations and Trends in Machine Learning. – 2021. – Vol. 14, № 1–2. – P. 1–210.

21. Bates A. W., Poole G. Effective Teaching with Technology in Higher Education: Foundations for success / A. W. Bates, G. Poole. – New York: Routledge, 2020. – 320 p.

22. Clark R. C., Mayer R. E. E-Learning and the Science of Instruction: proven guidelines for consumers and designers of multimedia learning. 4th ed. – Hoboken: Wiley, 2020. – 464 p.

23. Siemens G., Baker R. S. J. D. Learning Analytics: Definitions, frameworks and big questions / G. Siemens, R. S. J. D. Baker // Journal of Learning Analytics. – 2021. – Vol. 8, № 1. – P. 1–23.

24. Pappas C. Instructional Design for e-Learning: essential models and practices / C. Pappas. – New York: Routledge, 2022. – 240 p.

25. Anderson T., Dron J. Three generations of distance education pedagogy / T. Anderson, J. Dron // International Review of Research in Open and Distributed Learning. – 2020. – Vol. 21, № 3. – P. 1–20.

26. Bates T. Backbone of online learning: planning and strategy for LMS selection / T. Bates // Online Learning Journal. – 2021. – Vol. 25, № 2. – P. 47–62.

27. Conole G. Designing for Learning in an Open World. 2nd ed. – New York: Springer, 2021. – 300 p.

28. Selwyn N. Should robots replace teachers? critical perspectives on edtech / N. Selwyn. – Cambridge: Polity, 2022. – 240 p.

29. Wiley D., Hilton J. Open Educational Resources and LMS integration: practices and lessons / D. Wiley, J. Hilton // Educational Technology Research and Development. – 2020. – Vol. 68. – P. 1–15.

30. Siemens G., Long P. Educational data mining and learning analytics: from theory to practice / G. Siemens, P. Long // Journal of Learning Analytics. – 2021. – Vol. 8, № 1. – P. 4–17.

31. Коваленко О. М. Проєктний менеджмент в ІТ: методи та інструменти / О. М. Коваленко. – Київ: КНЕУ, 2021. – 312 с.

32. Єфремова Н. П. Agile-проєкти: організація та управління / Н. П. Єфремова. – Харків: ХНУ, 2022. – 280 с.

33. Литвиненко Т. Д. Управління розробкою веб-проєктів: методики і кейси / Т. Д. Литвиненко. – Київ: Вища школа, 2023. – 260 с.

34. Пархоменко О. І. Архітектура веб-орієнтованих інформаційних систем / О. І. Пархоменко // Журнал інформаційних систем. – 2021. – № 3. – С. 14–27.

35. Сидоренко Л. М. Досвід впровадження LMS у вищій освіті України / Л. М. Сидоренко // Освітні технології. – 2024. – № 1. – С. 88–99.

36. Кузьменко І. С. Інтеграція відеоконференцзв’язку і LMS: технічні та педагогічні аспекти / І. С. Кузьменко // Інформаційні технології в освіті. – 2022. – № 2. – С. 33–44.

37. Якимчук О. В. UX/UI для дистанційних платформ: дизайн для навчального досвіду / О. В. Якимчук. – Львів: ЛНУ, 2020. – 220 с.

38. Петренко І. В. Доступність та інклюзивний дизайн веб-освітніх ресурсів / І. В. Петренко // Педагогічна наука і практика. – 2023. – № 4. – С. 54–63.

39. Назаренко В. І. Безпека даних у дистанційному навчанні: ризики та засоби захисту / В. І. Назаренко. – Київ: Право й Технології, 2022. – 196 с.

40. Хоменко І. В. Аналітика участі студентів у дистанційних курсах: інструменти і методи / І. В. Хоменко // Науковий вісник університету. – 2024. – № 2. – С. 70–82.

41. Dabbagh N., Kitsantas A. The emergence of micro-learning and mobile learning for just-in-time education / N. Dabbagh, A. Kitsantas // Educational Technology Research and Development. – 2020. – Vol. 68, № 4. – P. 1–18.

42. Rasheed R. A., Kamsin A., Abdullah N. A. Challenges in the onlineization of higher education in COVID-19 era / R. A. Rasheed et al. // International Journal of Interactive Mobile Technologies. – 2021. – Vol. 15, № 5. – P. 4–17.

43. Hodges C., Moore S., Lockee B., Trust T., Bond A. The difference between emergency remote teaching and online learning / C. Hodges et al. // Educause Review. – 2020. – P. 1–12.

44. Graham C. R. Blended learning models and frameworks for higher education / C. R. Graham // Online Learning Journal. – 2021. – Vol. 25, № 2. – P. 1–19.

45. Spector J. M. Foundations of educational technology: theory and practice / J. M. Spector. – New York: Routledge, 2022. – 360 p.

46. Величко Н. І. Контент-менеджмент та авторські права в дистанційній освіті / Н. І. Величко // Публічне право. – 2021. – № 3. – С. 34–42.

47. Bower M. Design of technology-enhanced learning: models and strategies / M. Bower // Research in Learning Technology. – 2020. – Vol. 28. – P. 1–15.

48. Stojanovic N., Djuric D. Learning ecosystems: integration of services and platforms for lifelong learning / N. Stojanovic, D. Djuric // Computers & Education. – 2023. – Vol. 196. – P. 104663.

49. Лазаренко О. П. Методика експериментального оцінювання дистанційних курсів / О. П. Лазаренко. – Харків: ХНУ, 2024. – 198 с.

50. OECD. Education at a Glance 2023: OECD Indicators. – Paris: OECD Publishing, 2023. – 240 p.

ДОДАТКИ

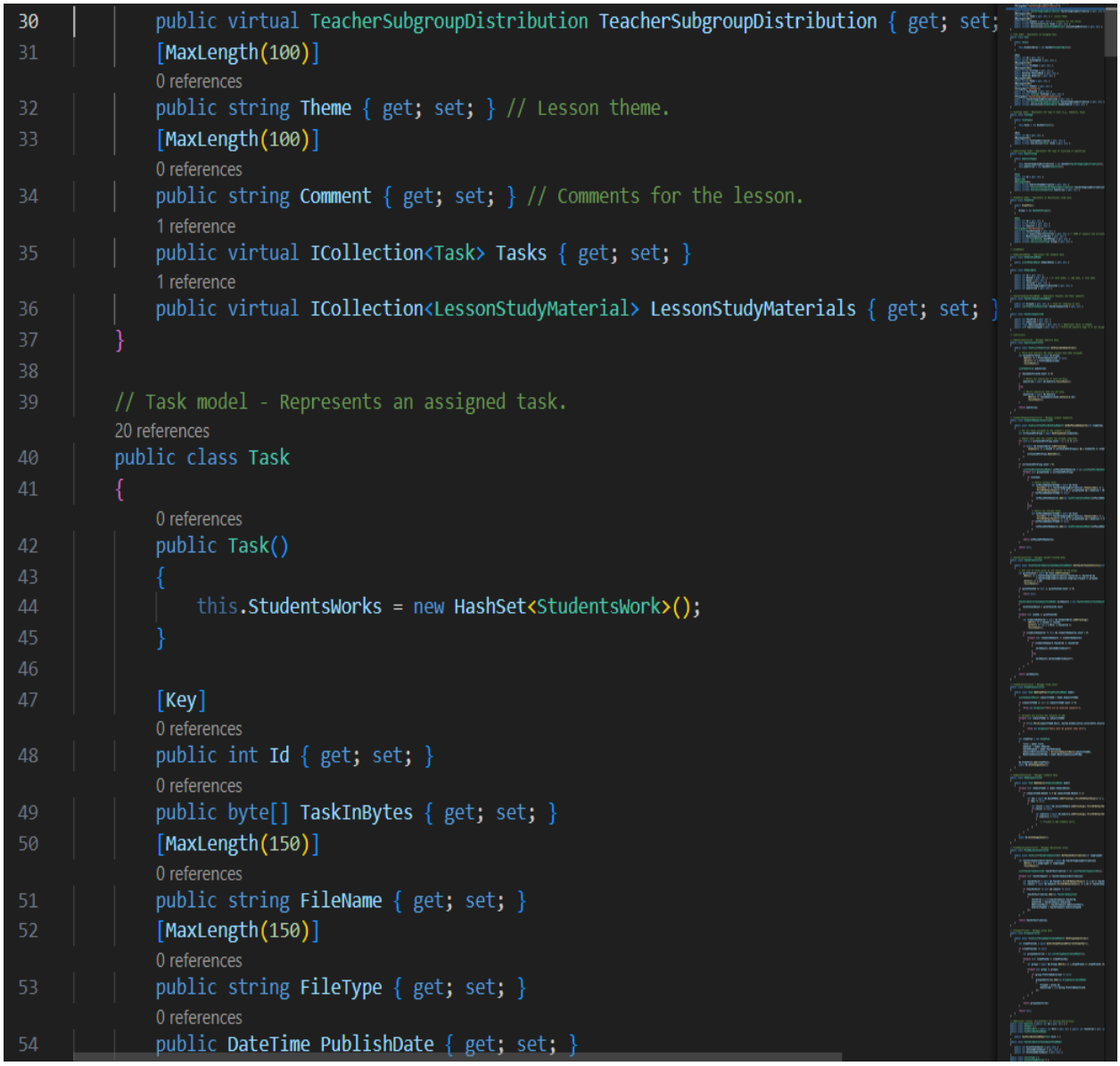
Додаток А

Прототип реалізації

main.cs

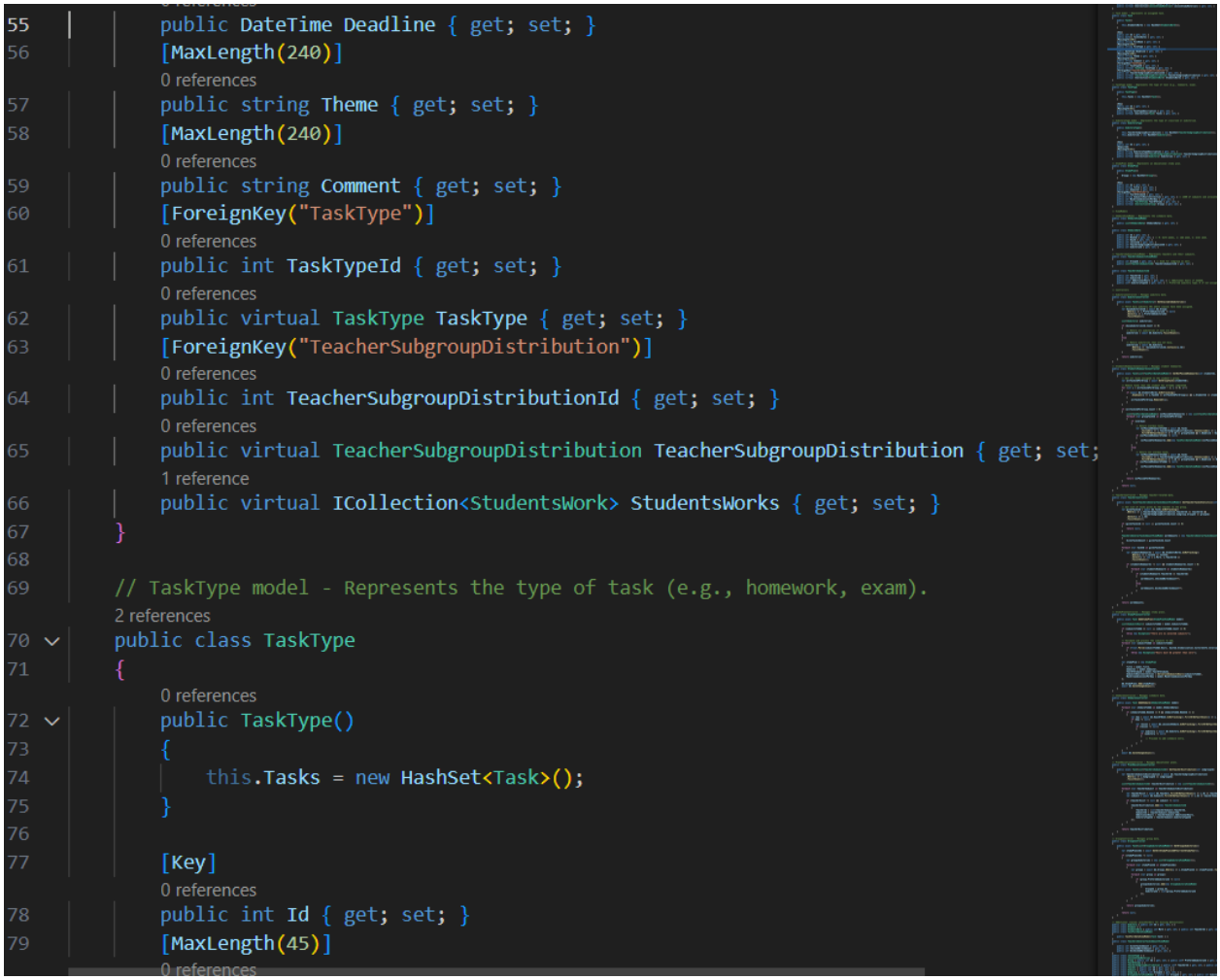
```cs
// Combined backend code with main components and brief comments.

using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using System.Threading.Tasks;

namespace DailyDiary
{
    // DbModels

    // WorkPlan model - Represents a teacher's work plan.
    public class WorkPlan
    {
        public WorkPlan()
        {
            this.Tasks = new HashSet<Task>();
            this.LessonStudyMaterials = new HashSet<LessonStudyMaterial>();
        }

        [Key]
        public int Id { get; set; }
        public DateTime? LessonDate { get; set; } // Date of the lesson.
        [ForeignKey("LessonType")]
        public int? LessonTypeId { get; set; } // Type of lesson.
        public virtual LessonType LessonType { get; set; }
        [ForeignKey("TeacherSubgroupDistribution")]
        public int TeacherSubgroupDistributionId { get; set; } // Teacher-subgroup-subject
        public virtual TeacherSubgroupDistribution TeacherSubgroupDistribution { get; set;
        [MaxLength(100)]
        public string Theme { get; set; } // Lesson theme.
        [MaxLength(100)]
        public string Comment { get; set; } // Comments for the lesson.
        public virtual ICollection<Task> Tasks { get; set; }
        public virtual ICollection<LessonStudyMaterial> LessonStudyMaterials { get; set; }
```

Прототип реалізації



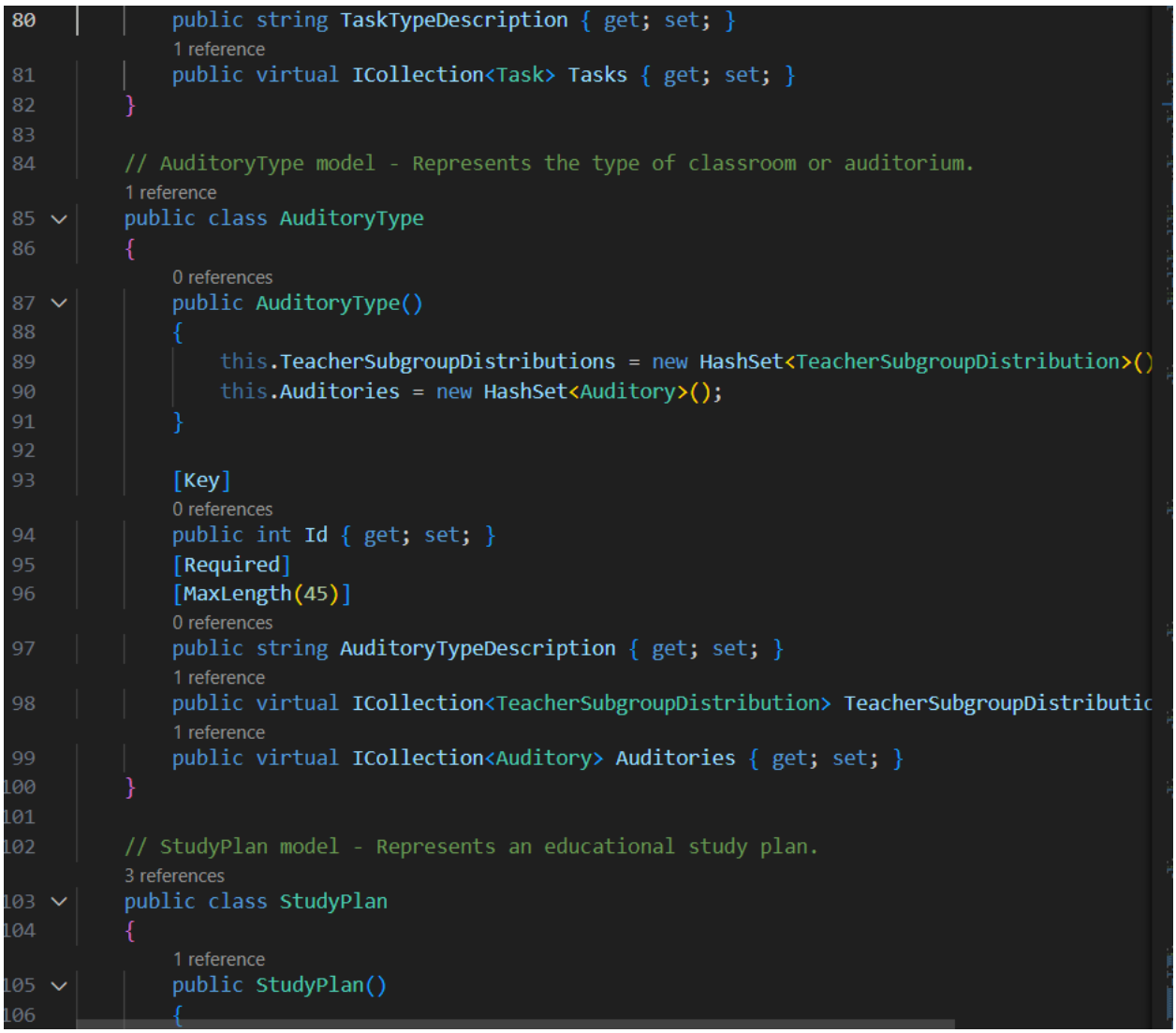
```
30          public virtual TeacherSubgroupDistribution TeacherSubgroupDistribution { get; set;
31          [MaxLength(100)]
            0 references
32          public string Theme { get; set; } // Lesson theme.
33          [MaxLength(100)]
            0 references
34          public string Comment { get; set; } // Comments for the lesson.
            1 reference
35          public virtual ICollection<Task> Tasks { get; set; }
            1 reference
36          public virtual ICollection<LessonStudyMaterial> LessonStudyMaterials { get; set; }
37      }
38
39      // Task model - Represents an assigned task.
        20 references
40      public class Task
41      {
            0 references
42          public Task()
43          {
44              this.StudentsWorks = new HashSet<StudentsWork>();
45          }
46
47          [Key]
            0 references
48          public int Id { get; set; }
            0 references
49          public byte[] TaskInBytes { get; set; }
50          [MaxLength(150)]
            0 references
51          public string FileName { get; set; }
52          [MaxLength(150)]
            0 references
53          public string FileType { get; set; }
            0 references
54          public DateTime PublishDate { get; set; }
```

Прототип реалізації



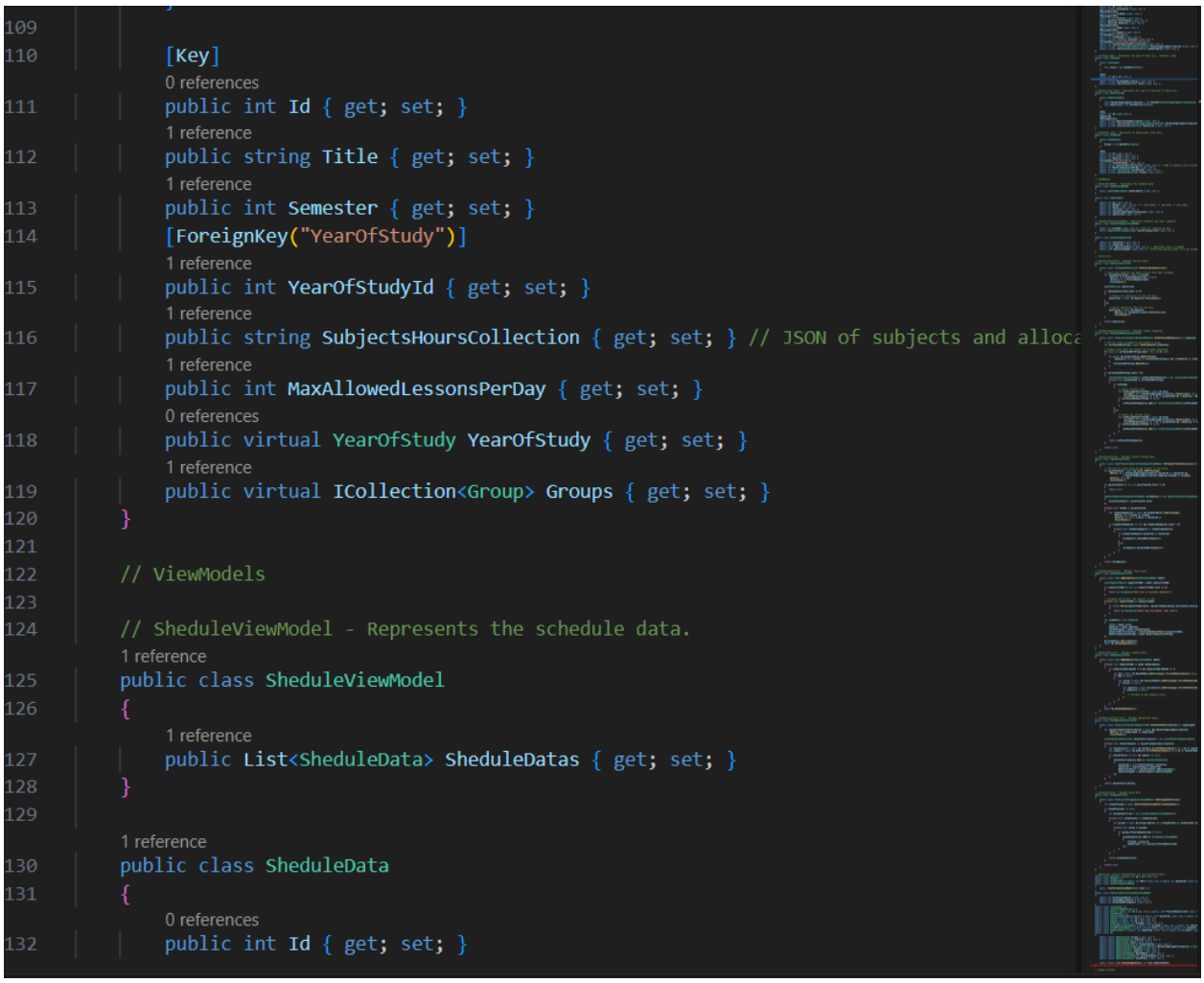
```
55          public DateTime Deadline { get; set; }
56          [MaxLength(240)]
            0 references
57          public string Theme { get; set; }
58          [MaxLength(240)]
            0 references
59          public string Comment { get; set; }
60          [ForeignKey("TaskType")]
            0 references
61          public int TaskTypeId { get; set; }
            0 references
62          public virtual TaskType TaskType { get; set; }
63          [ForeignKey("TeacherSubgroupDistribution")]
            0 references
64          public int TeacherSubgroupDistributionId { get; set; }
            0 references
65          public virtual TeacherSubgroupDistribution TeacherSubgroupDistribution { get; set;
            1 reference
66          public virtual ICollection<StudentsWork> StudentsWorks { get; set; }
67      }
68
69      // TaskType model - Represents the type of task (e.g., homework, exam).
        2 references
70      public class TaskType
71      {
            0 references
72          public TaskType()
73          {
74              this.Tasks = new HashSet<Task>();
75          }
76
77          [Key]
            0 references
78          public int Id { get; set; }
79          [MaxLength(45)]
            0 references
```

Прототип реалізації



```
80          public string TaskTypeDescription { get; set; }
            1 reference
81          public virtual ICollection<Task> Tasks { get; set; }
82      }
83
84      // AuditoryType model - Represents the type of classroom or auditorium.
        1 reference
85      public class AuditoryType
86      {
            0 references
87          public AuditoryType()
88          {
89              this.TeacherSubgroupDistributions = new HashSet<TeacherSubgroupDistribution>()
90              this.Auditories = new HashSet<Auditory>();
91          }
92
93          [Key]
            0 references
94          public int Id { get; set; }
95          [Required]
96          [MaxLength(45)]
            0 references
97          public string AuditoryTypeDescription { get; set; }
            1 reference
98          public virtual ICollection<TeacherSubgroupDistribution> TeacherSubgroupDistributic
            1 reference
99          public virtual ICollection<Auditory> Auditories { get; set; }
100     }
101
102     // StudyPlan model - Represents an educational study plan.
        3 references
103     public class StudyPlan
104     {
            1 reference
105         public StudyPlan()
106         {
```

Прототип реалізації



```
109
110         [Key]
            0 references
111         public int Id { get; set; }
            1 reference
112         public string Title { get; set; }
            1 reference
113         public int Semester { get; set; }
114         [ForeignKey("YearOfStudy")]
            1 reference
115         public int YearOfStudyId { get; set; }
            1 reference
116         public string SubjectsHoursCollection { get; set; } // JSON of subjects and alloca
            1 reference
117         public int MaxAllowedLessonsPerDay { get; set; }
            0 references
118         public virtual YearOfStudy YearOfStudy { get; set; }
            1 reference
119         public virtual ICollection<Group> Groups { get; set; }
120     }
121
122     // ViewModels
123
124     // SheduleViewModel - Represents the schedule data.
        1 reference
125     public class SheduleViewModel
126     {
            1 reference
127         public List<SheduleData> SheduleDatas { get; set; }
128     }
129
        1 reference
130     public class SheduleData
131     {
            0 references
132         public int Id { get; set; }
```

Прототип реалізації

```
140     // TeachersSubjectsViewModel - Represents teachers and their subjects.
        0 references
141     public class TeachersSubjectsViewModel
142     {
            0 references
143         public int GroupId { get; set; } // Used for subgroup as well.
            0 references
144         public List<TeachersSubjectsId> TeachersSubjectsId { get; set; }
145     }
146
        5 references
147     public class TeachersSubjectsId
148     {
            1 reference
149         public int TeacherId { get; set; }
            1 reference
150         public int SubjectId { get; set; }
            1 reference
151         public float AdditionalHours { get; set; } // Additional hours if needed.
            1 reference
152         public int? AuditoryTypeId { get; set; } // Preferred auditory type; 0 if not ass
153     }
154
155     // Controllers
156
157     // AuditoryController - Manages auditory data.
        0 references
158     public class AuditoryController
159     {
            0 references
160         public async Task<List<Auditory>> GetAvailableAuditories()
161         {
162             // Fetch busy auditory IDs where classes have been assigned.
163             var busyAuditoriesId = await db.Groups
164                 .Where(x => x.PreferedAuditoryId != null)
165                 .Select(x => x.PreferedAuditoryId)
166                 .ToListAsync();
```