НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри

_____комп'ютерних наук_____
(назва кафедри)

_____ /_____Голуб Б.Л._____/
(підпис) (ПІБ)

"02" червня 2025 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА
на тему

Інформаційно-управляюча система діяльності викладача навчального закладу

Спеціальність 122 – «Комп'ютерні науки»

Гарант освітньої програми

_____д.ек.н, професор_____ _____ __Руденський Р.А.__
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Керівник бакалаврської кваліфікаційної роботи

_____ _____ ___Степанов О.В.___
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав _____ _____Семчук Богдан Ігорович_____
(підпис) (ПІБ студента)

КИЇВ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БІОРЕСУРСІВ І
ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____комп'ютерних наук_____
(назва кафедри)

___к.т.н., доцент_____Голуб Б.Л.
(науковий ступінь, вчене звання) (підпис) (ПІБ)
"16" грудня 2024 р.

З А В Д А Н Н Я
на виконання бакалаврської кваліфікаційної роботи студенту
Семчук Богдан Ігорович

Спеціальність 122 – «Комп'ютерні науки»

ОП «Комп'ютерні науки»

1. Тема бакалаврської кваліфікаційної роботи «Інформаційно-управляюча система діяльності викладача навчального закладу» затверджена наказом ректора НУБіП України від 25.04.2025 № _699 «С»

2. Термін подання завершеної роботи на кафедру __2025. 05. _25__
(рік, місяць, число)

3. Вихідні дані до роботи: опис програмного забезпечення

4. Перелік питань, що розглядаються:
- Аналіз проблемної області
- Моделювання предметної області
- Проєктування програмної системи
- Впровадження та експлуатація системи

Дата видачі завдання "16" грудня 2024 р.

Керівник бакалаврської кваліфікаційної роботи _____/О.В. Степанов /
(підпис) (прізвище та ініціали)

Завдання прийняв до виконання:_____/ Б.І. Семчук /
(підпис) (прізвище та ініціали)

ЗМІСТ

ВСТУП

Стрімкий розвиток цифрових технологій суттєво трансформував навчальний процес, зробивши його більш інтерактивним та доступним. Проте багато навчальних закладів все ще стикаються з проблемами в організації та управлінні навчанням студентів, особливо у відстеженні завдань, оцінюванні роботи та підтримці ефективного спілкування між викладачами та студентами. Для вирішення цих питань пропонується розробка Інформаційно-управляючої системи викладача навчального закладу. Отже, у зв'язку зі змінами у форматі навчального процесу, та збільшення попиту на веб-системи для комфортного віддаленого навчання, розробка веб системи, яка орієнтована на підвищення рівня сучасної освіти, зменшення робочого навантаження, як на студентів, так і викладачів – є **актуальною**.

Головна **мета** цієї дипломної роботи – це створення веб-орієнтованої інформаційно-управляючої системи для сфери освіти.

Для виконання мети проєкту поставлені наступні **завдання**: аналіз існуючих рішень та дослідження проблематики, будування логічної та фізичної моделей даних, на основі яких створити програмний продукт з акцентом на зручний інтуїтивно зрозумілий інтерфейс, підтримкою клієнт-серверної системи та можливістю подальшого вдосконалення та масштабування.

Апробація. Дана робота була представлена тезами: «Інформаційно-управляюча система діяльності викладача навчального закладу» на VII Всеукраїнській науково-практичній конференції студентів і аспірантів «Теоретичні та прикладні аспекти розробки комп'ютерних систем '2025'», що відбулася 24 квітня 2025 року. Завдяки науковій конференції були чітко сформовані вектори руху проєкту, а з почутих доповідей сформоване більш актуальне уявлення про проєктування та розробку сучасних інформаційних систем будь-яких різновидів.

У розробці нашої системи використовуються сучасні веб-технології та методології програмної інженерії. Він створений як веб-платформа з використанням PHP і Symfony для розробки бекенду, MySQL для керування базами даних і зручного інтерфейсу для безперебійної взаємодії. Архітектура системи забезпечує масштабованість, надійність і простоту обслуговування.

Ця дипломна робота складається з чотирьох розділів: аналіз предметної області та існуючих рішень, де проводиться дослідження в потребах сучасних викладачів та огляд готових навчальних платформ; проектування та моделювання системи, тобто ключові етапи створення інформаційно-управляючої системи; процес розробки, що здійснюється з допомогою фреймворку Symphony та СУБД MySQL, а також впровадження та оцінка кінцевого продукту після етапу тестування. Робота містить 66 сторінок, у додатках наведені допоміжні матеріали.

1 АНАЛІЗ ПРОБЛЕМНОЇ ОБЛАСТІ

1.1 Опис предметної області

Зростання ролі цифрових технологій в освіті призвело до значних трансформацій в академічному менеджменті. Однак, незважаючи на технологічний прогрес, багато навчальних закладів досі борються з неефективними системами організації дисциплін, розподілу завдань, відстеження прогресу студентів та оцінювання успішності. Відсутність централізованої платформи для управління цими процесами створює численні проблеми як для викладачів, так і для студентів, що призводить до непотрібного адміністративного тягаря, затримок у зворотному зв’язку та відсутності прозорості в оцінюванні.

Основною проблемою, з якою стикаються викладачі, є відсутність цілісної системи структурування дисциплін і завдань. У традиційних підходах такі завдання, як лабораторні роботи, практичні завдання, самостійні проекти, модульні тести та іспити, часто виконуються за допомогою комбінації паперових записів, електронних листів або платформ загального призначення для обміну файлами. Ця фрагментація ускладнює підтримку узгодженості, відстеження подання та забезпечення справедливого оцінювання. Викладачі повинні вручну збирати та організовувати роботу студентів, що збільшує ризик втрати файлів, невідстежених поданих робіт і непослідовних критеріїв оцінювання.

З боку студентів відсутність уніфікованої системи створює плутанину в відстеженні призначених завдань, термінів виконання та відгуків. Студентам часто доводиться покладатися на кілька каналів зв’язку, що може призвести до непорозумінь або пропуску термінів. Крім того, отримання та перегляд минулих оцінок може бути складним, якщо немає спеціального сховища для зберігання оцінок і коментарів викладачів.

Існуючі рішення, такі як комерційні системи управління навчанням (LMS), намагаються вирішити ці проблеми, але часто є надто складними,

дорогими або не пристосованими до конкретних потреб управління на рівні відділу. Багато платформ LMS розроблено для широкомасштабного інституційного використання, інтегруючи функції, які непотрібні для невеликих академічних відділів, у той час як їм бракує специфічних функцій, необхідних для індивідуального управління курсами. У результаті викладачі та студенти або стикаються з надто складними інтерфейсами, або вдаються до використання кількох відключених інструментів, що знижує загальну ефективність [22].

Іншою критичною проблемою є відсутність прозорості у відстеженні успішності студентів. Багато навчальних закладів покладаються на системи оцінювання та зворотного зв’язку вручну, де студенти мають обмежений доступ до свого прогресу до остаточного оцінювання. Це може вплинути на мотивацію та перешкодити можливостям своєчасного вдосконалення. Вчителі також стикаються з труднощами у підтримці структурованої історії оцінювання, що ускладнює надання персоналізованого зворотного зв’язку або аналіз тенденцій ефективності з часом.

Щоб вирішити ці проблеми, Інформаційна система для викладачів факультету має на меті забезпечити комплексну та зручну платформу, яка спрощує керування дисциплінами, завданнями та оцінюванням студентів. Запропонована система дозволить викладачам створювати та керувати курсами, призначати класифіковані завдання, збирати подання студентів, оцінювати роботу з оцінками та коментарями та зберігати всі дані в структурованому вигляді. Студенти, у свою чергу, матимуть легкий доступ до своїх дисциплін, ефективно подаватимуть завдання та відстежуватимуть свій прогрес через спеціальний інтерфейс.

Автоматизуючи рутинні адміністративні завдання та покращуючи спілкування між викладачами та студентами, система підвищить ефективність навчальних процесів. Інтеграція централізованої системи оцінювання забезпечить прозорість, дозволяючи студентам контролювати свою академічну успішність у режимі реального часу. Крім того, система підтримуватиме

зберігання та пошук даних, полегшуючи вчителям перегляд робіт студентів, коригування стратегій навчання та ведення повної академічної документації [1].

Навчальний процес в університетах і коледжах включає численні адміністративні та академічні завдання, які вимагають ефективного управління. Викладачі несуть відповідальність за організацію дисциплін, призначення та оцінювання робіт студентів, ведення академічної документації. Студенти, у свою чергу, потребують доступу до матеріалів курсу, завдань і своєчасного відгуку про їхні успіхи. Однак відсутність добре структурованої цифрової системи часто призводить до неефективності, що ускладнює оптимізацію цих процесів.

Інформаційна система для викладачів кафедри розроблена для вирішення цих проблем, надаючи спеціалізовану веб-платформу, яка спрощує керування дисциплінами, завданнями та оцінюванням студентів. Предметна область цієї системи охоплює організацію курсів, обробку завдань, відстеження успішності студентів, а також автоматизацію процесів оцінювання та зворотного зв’язку.

Дисципліна в системі являє собою навчальний курс, який читає викладач. Кожна дисципліна містить кілька завдань, які поділяються на лабораторні, практичні, самостійні роботи, модульні контрольні роботи та іспити. Вчителі мають можливість створювати ці завдання, встановлювати терміни виконання та контролювати прогрес учнів.

Студенти, які вивчають дисципліну, можуть отримати доступ до призначених їм завдань, подати виконану роботу та отримати оцінки разом із коментарями викладача. Система гарантує, що всі подання належним чином зберігаються та можуть бути переглянуті в будь-який час. Це усуває потребу в розпорошених каналах зв’язку, таких як електронна пошта або хмарне сховище, що робить процес більш структурованим і ефективним.

Однією з основних функцій системи є модуль оцінювання, який дозволяє викладачам оцінювати роботу студентів і виставляти оцінки з коментарями. Ці оцінки реєструються в системі та надаються студентам, забезпечуючи повну прозорість у процесі оцінювання. Крім того, система забезпечує загальну

функцію відстеження оцінок, де студенти можуть переглядати свою успішність з усіх дисциплін.

Система є ключовим інструментом для модернізації освітніх робочих процесів і забезпечення більш структурованого та прозорого процесу навчання.

Детальний опис класів та атрибутів предметної області наведено у таблиці 1.1.

Таблиця 1.1

Опис атрибутів класів предметної області

Клас предметної області	**Атрибут**	**Опис**
Користувач	Ім'я	Повне ім'я користувача (студента або викладача).
	Роль	Визначає, чи є користувач студентом або викладачем.
	Електронна пошта	Використовується для входу в систему та отримання сповіщень.
Дисципліна	Назва дисципліни	Найменування навчальної дисципліни.
	Викладач	Особа, яка веде дисципліну та керує нею.
	Студенти	Список студентів, які вивчають дисципліну.
Завдання	Назва роботи	Тема або заголовок конкретного завдання.
	Категорія	Тип роботи (лабораторна, практична, модульна тощо).
	Дисципліна	До якої навчальної дисципліни належить це завдання.
	Дедлайн	Кінцевий термін здачі завдання.

1.2 Огляд існуючих рішень

Розглянемо існуючі рішення предметної області.

Moodle, що розшифровується як модульне об’єктно-орієнтоване динамічне навчальне середовище, є популярною системою керування навчанням (LMS) із відкритим вихідним кодом, розробленою для полегшення онлайн-навчання та змішаного навчання. Широко застосовуваний університетами, школами та корпоративними навчальними програмами, Moodle надає комплексну платформу для керування курсами, завданнями, оцінками та спілкуванням між викладачами та студентами.

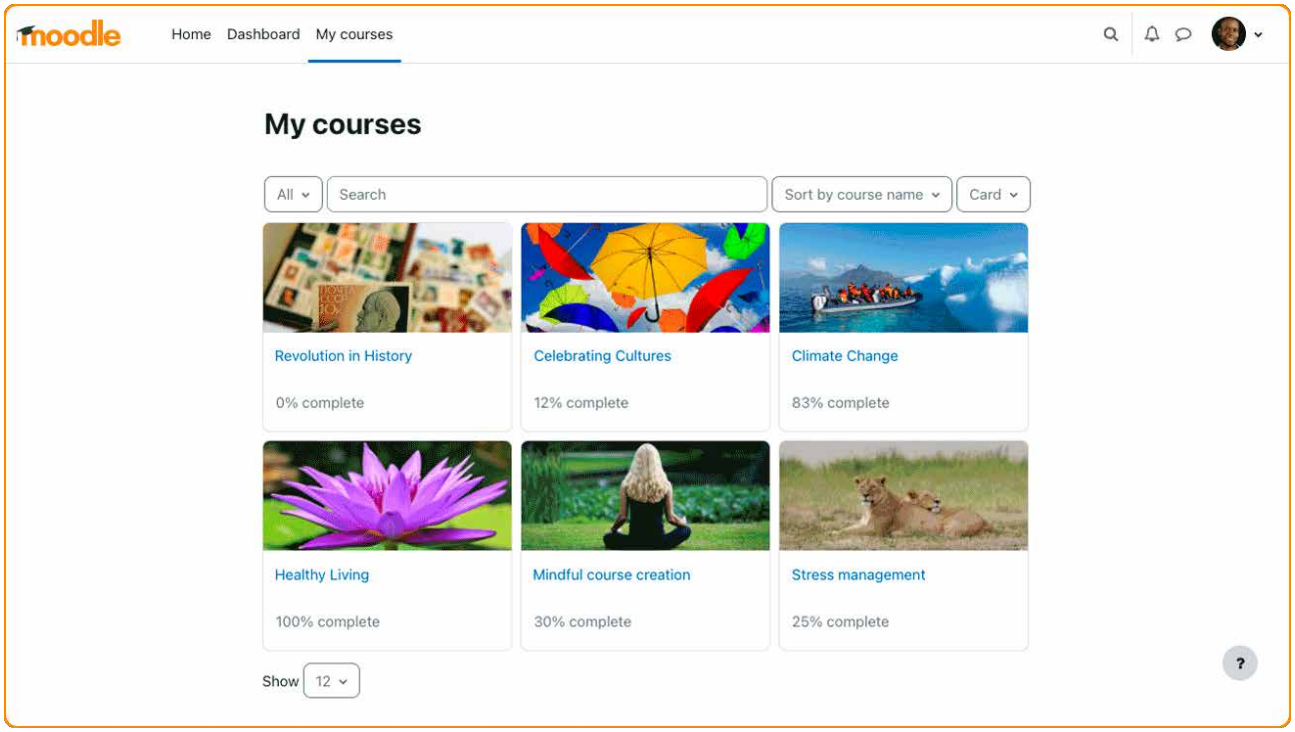

Рис. 1 Інформаційна система «Moodle»

Однією з ключових особливостей Moodle є його надійні можливості керування курсами. Педагоги можуть легко створювати структуровані курси, які включають різні модулі, теми та уроки, адаптовані до їхньої навчальної програми. Платформа підтримує низку інструментів оцінювання, що дозволяє вчителям призначати завдання, керувати надсиланням файлів, створювати тести та автоматизувати процеси оцінювання [7].

Співпраця є фундаментальним аспектом досвіду навчання, і Moodle пропонує кілька інструментів для покращення взаємодії студентів. Такі функції,

як дискусійні форуми, вікі та групові проекти, сприяють взаємодії та полегшують спілкування між учнями. Крім того, Moodle містить журнал оцінок, який відстежує успішність студентів і дозволяє викладачам надавати детальний відгук, сприяючи сприятливому навчальному середовищу [20].

Гнучкість платформи є однією з її суттєвих переваг. Будучи відкритим кодом, Moodle є безкоштовним для використання, що робить його доступним для широкого кола установ. Його можна налаштувати відповідно до конкретних освітніх потреб, і існує величезна бібліотека плагінів і тем, доступних для розширення його функціональності. Moodle також підтримує інтеграцію з різними інструментами сторонніх розробників, такими як Google Drive, Zoom і Microsoft Teams, що покращує його зручність використання.

Незважаючи на численні сильні сторони, Moodle має деякі проблеми. Налаштування та підтримка платформи може потребувати технічних знань, особливо для установ, які бажають розмістити її на своїх серверах. Деякі користувачі вважають інтерфейс менш інтуїтивно зрозумілим порівняно з новими рішеннями LMS, що може вплинути на взаємодію з користувачем. Крім того, більші установки можуть зіткнутися з проблемами продуктивності, що потребує надійної серверної інфраструктури.

Moodle використовується різними організаціями, включаючи університети та коледжі, які керують онлайн-курсами, школи K-12, які застосовують стратегії змішаного навчання, і компанії, що впроваджують програми навчання співробітників. Його адаптивність робить його придатним вибором для різних освітніх контекстів.

Підводячи підсумок, можна сказати, що Moodle — це потужна та гнучка LMS, яка завоювала популярність завдяки настроюваним функціям і масштабованості. Хоча він пропонує численні переваги для управління освітніми процесами, потенційні користувачі повинні знати про технічні вимоги, пов’язані з його впровадженням і обслуговуванням [15].

Розглянемо інше програмне рішення на ринку.

eLearn — це інноваційна онлайн-платформа для навчання, розроблена для полегшення навчального досвіду як для студентів, так і для викладачів. Він надає повний набір інструментів і функцій, які підтримують різні аспекти онлайн-навчання, включаючи керування курсами, доставку контенту, оцінювання та співпрацю.

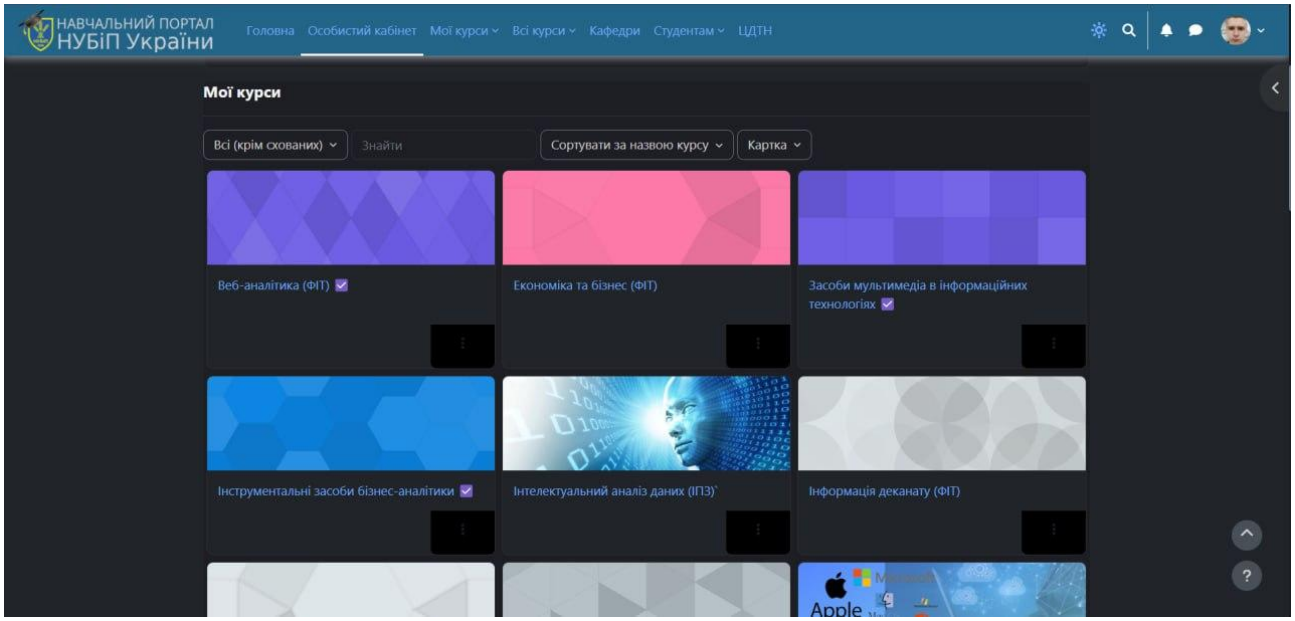

Рис. 2 Інформаційна система «eLearn»

В основі eLearn лежить зручний інтерфейс, який дозволяє викладачам легко створювати курси та керувати ними. Викладачі можуть завантажувати різні типи вмісту, такі як відео, документи, тести та інтерактивні дії, роблячи процес навчання привабливим і динамічним. Платформа підтримує різні формати навчання, включаючи курси самостійного навчання, живі сесії та змішані навчальні середовища.

Однією з видатних особливостей eLearn є його можливості оцінювання. Викладачі можуть розробляти та проводити тести, іспити та завдання для оцінки прогресу студентів. Система забезпечує миттєвий зворотний зв'язок, дозволяючи студентам зрозуміти їх результативність і області, які потрібно вдосконалити. Крім того, eLearn пропонує надійні інструменти оцінювання, які дозволяють викладачам відстежувати досягнення студентів і надавати персоналізований зворотний зв’язок [2].

Співпраця є важливим елементом досвіду eLearn. Платформа включає дискусійні форуми, групові проекти та функції обміну повідомленнями, сприяючи спілкуванню та взаємодії між студентами, а також між студентами та викладачами. Ці інструменти для спільної роботи допомагають створити відчуття спільності, що є важливим для успішного онлайн-навчання.

На додаток до своїх освітніх функцій, eLearn надає інструменти аналітики та звітності, які допомагають інструкторам відстежувати залученість і успішність студентів. Ці знання дають змогу педагогам приймати рішення на основі даних і адаптувати свої стратегії навчання відповідно до потреб своїх учнів.

Загалом, eLearn — це універсальна й ефективна платформа онлайн-навчання, яка покращує навчальний досвід як для вчителів, так і для студентів. Його поєднання управління курсами, інструментів оцінювання, функцій співпраці та аналітики робить його цінним ресурсом для установ, які прагнуть застосувати цифрове навчання. Завдяки своїй зосередженості на взаємодії з користувачем і доступності, eLearn добре підходить для широкого кола освітніх контекстів, від шкіл K-12 до вищої освіти та корпоративних навчальних програм [16].

1.3 Постановка завдання

Основною метою цього проекту є розробка інформаційної системи для викладачів кафедр, яка б ефективно керувала різними аспектами навчального процесу. Ця система повинна дозволити вчителям вести детальний облік критичних показників, пов’язаних з управлінням курсом і успішністю студентів. Щоб досягти цього, потрібна надійна база даних для зберігання важливої інформації, такої як деталі курсу, завдання, роботи студентів, оцінки та відгуки.

Програмна система, яка розробляється, має на меті надати викладачам швидкий і легкий доступ до важливої інформації про:

- організація та структура дисциплін, що викладаються на кафедрі;

- управління та відстеження завдань, включаючи терміни та статуси подання;
- оцінювання успішності студентів, виставлення оцінок та коментарі до поданих робіт;
- уміння ефективно спілкуватися зі студентами щодо матеріалів курсу та відгуків.

Програма матиме інтуїтивно зрозумілий інтерфейс, який дозволить викладачам легко переміщатися по меню для керування курсами та завданнями. Крім того, система включатиме функціональні можливості для створення звітів про успішність студентів і ефективність курсу, що дозволить викладачам приймати обґрунтовані рішення для підвищення ефективності навчання.

Впроваджуючи цю інформаційну систему, освітяни матимуть можливість оптимізувати свої адміністративні завдання, покращити спілкування зі студентами та, зрештою, сприяти більш продуктивному освітньому середовищу.

1.4 Функціональні та нефункціональні вимоги

Функціональні вимоги:

1. система повинна дозволяти створення, зміну та видалення облікових записів як для викладачів, так і для студентів;
2. викладачі повинні мати можливість створювати, оновлювати та видаляти курси (дисципліни) у системі;
3. викладачі повинні вміти складати різні види робіт (лабораторні роботи, практичні роботи, самостійна робота, модульні контрольні роботи, іспити);
4. система повинна дозволяти вчителям встановлювати терміни виконання кожного завдання;
5. студенти повинні мати можливість надсилати свої роботи для виконання завдань через систему;

6. система повинна відстежувати статуси подання (наприклад, подано, очікує на розгляд, із запізненням);
7. вчителі повинні вміти оцінювати подані завдання, виставляти оцінки та коментарі;
8. система повинна автоматично оновлювати загальну оцінку студента на основі оцінок.

Нефункціональні вимоги:

1. система повинна мати інтуїтивно зрозумілий і зручний інтерфейс, щоб забезпечити зручність навігації як викладачам, так і студентам;
2. система повинна обробляти одночасний доступ користувачів без значних затримок або зниження продуктивності;
3. час реакції на дії користувача (наприклад, завантаження сторінок, відправка завдань) не повинен перевищувати трьох секунд;
4. дані користувача мають бути захищені безпечними методами автентифікації, включаючи шифрування пароля;
5. система повинна відповідати правилам захисту даних, щоб забезпечити конфіденційність інформації студентів;
6. система повинна бути розроблена таким чином, щоб враховувати все більшу кількість користувачів і курсів, не вимагаючи істотних змін в архітектурі;
7. він повинен підтримувати майбутні вдосконалення та додавання нових функцій.

Задовольняючи як функціональні, так і нефункціональні вимоги, інформаційна система для викладачів факультету прагне забезпечити комплексне та ефективне рішення, яке покращує навчальний процес, забезпечуючи при цьому позитивний досвід користувача.

2 МОДЕЛЮВАННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

2.1 Загальні відомості

Уніфікована мова моделювання (UML) — це широко визнана мова моделювання, яка використовується в розробці програмного забезпечення для візуалізації, специфікації, конструювання та документування різних компонентів програмної системи. Надаючи стандартизований набір методів графічної нотації, UML полегшує створення абстрактних моделей, дозволяючи розробникам, архітекторам і зацікавленим сторонам краще розуміти та передати дизайн і структуру системи [9].

UML охоплює різноманітні діаграми, які можна розділити на дві основні групи: структурні діаграми та поведінкові діаграми. Кожен тип виконує певну функцію в моделюванні різних аспектів системи. Структурні діаграми зосереджені на статичній структурі системи, демонструючи зв’язки та атрибути класів. Наприклад, діаграми класів відображають різні класи в системі разом із їхніми зв’язками, тоді як діаграми об’єктів фіксують знімки екземплярів цих класів у певні моменти. Діаграми компонентів ілюструють організацію та залежності програмних компонентів, надаючи розуміння архітектури системи, тоді як діаграми розгортання показують, як програмні артефакти фізично розподіляються між апаратними вузлами [12].

З іншого боку, поведінкові діаграми підкреслюють динамічну поведінку системи. Діаграми варіантів використання окреслюють взаємодію між користувачами та системою, визначають ключові функціональні вимоги та основні варіанти використання. Діаграми послідовності детально описують взаємодію між об’єктами в певному порядку, зосереджуючись на повідомленнях, якими обмінюються під час певних сценаріїв. Діаграми діяльності представляють робочі процеси в системі, ілюструючи послідовність дій і моментів прийняття рішень, залучених до процесу. Діаграми станів фіксують

різні стани об’єкта та переходи між цими станами, пропонуючи комплексне уявлення про поведінку об’єкта з часом [14].

Прийняття UML пропонує кілька переваг. Будучи стандартизованою мовою, вона сприяє узгодженості моделювання в різних командах і проектах, спрощуючи розуміння та обмін дизайнами. Візуальне представлення діаграм UML полегшує розуміння складних систем і одночасно сприяє ефективній комунікації між зацікавленими сторонами, включаючи розробників, бізнес-аналітиків і керівників проектів. Крім того, UML служить цінним інструментом документації, що фіксує рішення щодо проектування та поведінку системи для довідки протягом життєвого циклу розробки програмного забезпечення [17].

UML знаходить застосування на різних етапах розробки програмного забезпечення. Він відіграє вирішальну роль у зборі вимог, допомагаючи ідентифікувати та документувати системні вимоги за допомогою діаграм варіантів використання. На етапі проектування створюються діаграми класів і діаграми компонентів, щоб визначити структуру й архітектуру системи. На етапі реалізації ці моделі спрямовують розробників під час кодування, а також підтримують створення тестових випадків на основі діаграм UML, щоб переконатися, що система відповідає запланованим вимогам.

Таким чином, UML є потужним і універсальним інструментом у розробці програмного забезпечення, який допомагає моделювати, проектувати та документувати програмні системи. Його здатність надавати чіткі та стандартизовані візуальні представлення робить його безцінним ресурсом для розробників і зацікавлених сторін протягом усього процесу розробки програмного забезпечення.

2.2 Об’єктне та функціональне моделювання

2.2.1 Діаграма прецедентів. Діаграма варіантів використання надає графічне представлення того, як користувачі, яких називають акторами, взаємодіють із системою. Будучи життєво важливим елементом уніфікованої

мови моделювання (UML), він пропонує високорівневий огляд функціональних можливостей у системі, підкреслюючи конкретні цілі, яких користувачі прагнуть досягти під час взаємодії. Ілюструючи ці взаємодії, діаграми варіантів використання відіграють вирішальну роль у визначенні функціональних вимог системи [19].

В основі діаграми прецедентів лежить кілька ключових компонентів. Актори представляють різних користувачів або зовнішні системи, які взаємодіють з основною системою, що моделюється. Це можуть бути люди-користувачі, такі як адміністратори, викладачі та студенти, а також інші системи, які спілкуються з цільовою системою. На схемі актори зазвичай зображені у вигляді фігурок [24].

Варіанти використання позначають конкретні функції або дії, які система виконує у відповідь на введення актора. Кожен варіант використання описує окрему мету, яку актор прагне досягти під час використання системи. На схемі вони представлені у вигляді овалів і позначені описовими назвами, щоб чітко передати їх призначення.

Межа системи визначає область аналізованої системи та зображена у вигляді прямокутника, який охоплює всі варіанти використання. Ця межа допомагає розмежувати внутрішні функціональні можливості системи та зовнішні актори, уточнюючи, що входить до можливостей системи.

Відносини на діаграмі показують, як взаємодіють актори та варіанти використання. Найпоширенішим типом зв’язку є асоціація, представлена простою лінією, що з’єднує актора з варіантом використання, що вказує на пряму взаємодію. Інші зв’язки включають «include», що означає, що один варіант використання завжди є частиною іншого, і «extend», який показує, як один варіант використання може бути покращений іншим за певних умов. Ці зв’язки проілюстровано пунктирними стрілками, які вказують на відповідні випадки використання.

Спроєктована діаграма прецедентів представлена на рис.3.

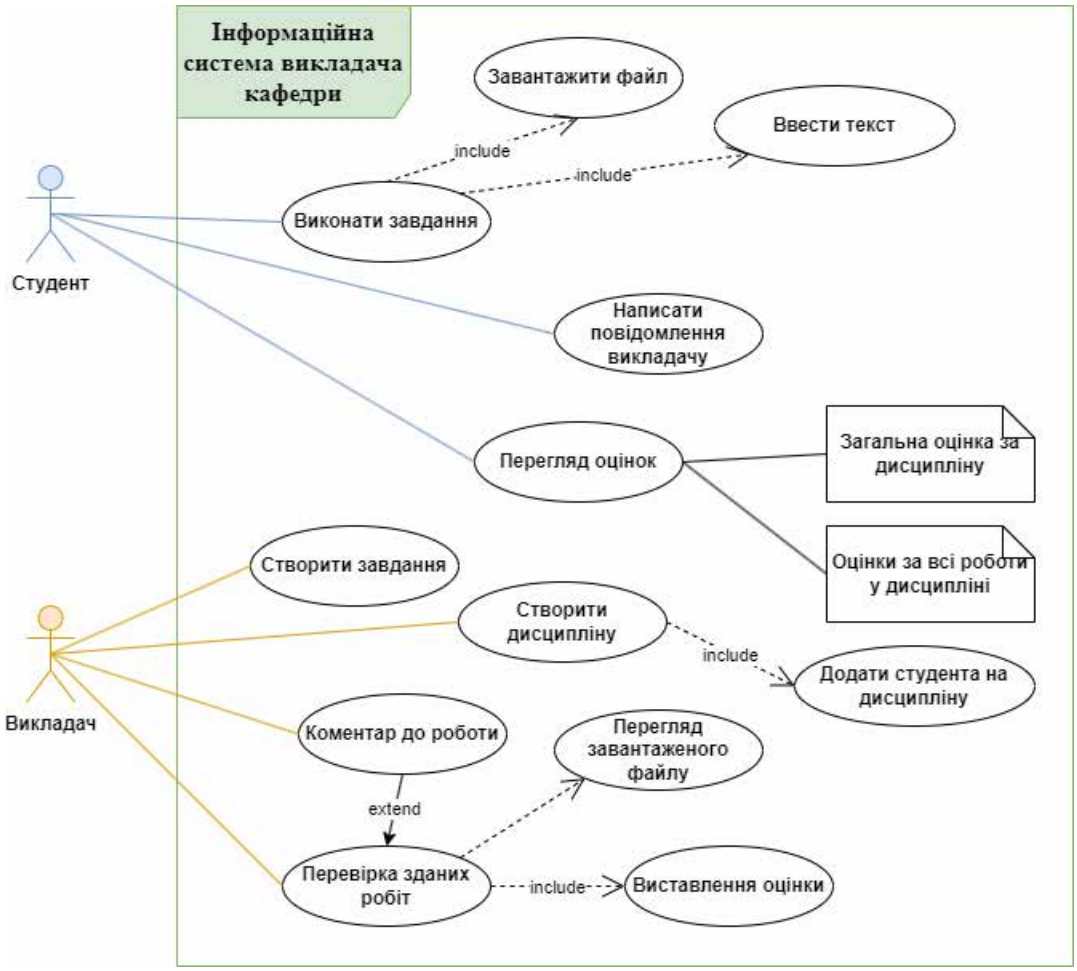

Рис. 3 Діаграма прецедентів

Створена діаграма прецедентів містить акторів:

- “Студент”;
- “Викладач”.

Актор «Студент» включає такі прецеденти:

- виконати завдання;
- завантажити файл;
- ввести текст;
- написати повідомлення викладачу;
- перегляд оцінок.

Актор «Викладач» включає такі прецеденти:

- створити завдання;
- створити дисципліну;
- додати студента до дисципліни;

- перевірка зданих робіт;
- виставлення оцінки;
- коментар до роботи;
- перегляд завантаженого файлу.

Прецеденти певним чином залежать одне від одного.

Прецедент “Виконати завдання” доповнюється іншими такими прецедентами як “Завантажити файл” та “Ввести текст”.

Прецедент “Перегляд оцінок” включає в себе такі анотації як “Загальна оцінка за дисципліну”, “Оцінки за всі роботи у дисципліни”.

Розглянемо детальніше вищеописані прецеденти.

Назва випадку використання: створити завдання

Актор: Викладач

Передумови: Викладач повинен бути авторизований в системі. Викладач має доступ принаймні до одного курсу (дисципліни), де буде поставлено завдання.

Основний потік:

1. викладач переходить до розділу керування курсом системи;
2. викладач обирає конкретний курс, для якого хоче створити нове завдання;
3. система відображає деталі курсу та можливість створити нове завдання;
4. учитель натискає кнопку «Створити завдання»;
5. система представляє форму для введення деталей завдання;
6. учитель заповнює бланк необхідною інформацією;
7. викладач переглядає введені дані, щоб переконатися в точності;
8. викладач надсилає форму, натискаючи кнопку «Зберегти» або «Надіслати»;
9. система обробляє запит і створює нове завдання в базі даних;
10. відображається повідомлення про підтвердження, яке вказує на успішне створення завдання;

11. система автоматично сповіщає студентів, зарахованих на курс, про нове завдання, включаючи його назву та дату виконання.

Альтернативні потоки:

1. якщо вчитель не заповнює всі необхідні поля, система сповіщає вчителя повідомленнями про помилку із зазначенням відсутньої інформації. Викладач має виправити помилки та повторно подати форму;
2. якщо вчитель у будь-який момент вирішить скасувати процес створення завдання, він може натиснути кнопку «Скасувати», і система поверне його на сторінку керування курсом без збереження введеної інформації.

У цьому сценарії описано етапи використання сценарію «Створити завдання», деталізовано взаємодію між викладачем і системою під час створення завдання та надання його учням.

Розглянемо інший сценарій.

Назва випадку використання: Надіслати завдання

Актор: Студент

Передумови: Студент повинен бути авторизований в системі. Студент повинен бути зарахований до курсу (дисципліни), на який ставиться завдання. Завдання має бути доступним для подання та не простроченим.

Основний потік:

1. студент переходить до розділу керування курсом системи;
2. студент обирає конкретний курс, який містить завдання, яке вони бажають подати;
3. система відображає деталі курсу разом зі списком завдань, включаючи їх назви, описи та терміни подання;
4. студент визначає завдання, яке він виконав, і натискає кнопку «Надіслати завдання», пов’язану з цим завданням;
5. система представляє форму подання, пропонуючи студенту завантажити виконану роботу;

6. учень натискає кнопку «Завантажити», щоб вибрати файл зі свого пристрою;
7. студент переглядає деталі завдання та підтверджує, що він готовий подати завдання;
8. учень здає завдання, натиснувши кнопку «Відправити»;
9. система обробляє подання, зберігаючи файл і пов’язуючи його з обліковим записом студента та конкретним завданням;
10. відображається повідомлення про підтвердження, яке вказує на те, що завдання успішно надіслано;
11. система оновлює статус завдання, щоб відобразити його надсилання;
12. вчителя сповіщають про нове подання, включаючи ім’я студента та назву завдання.

Альтернативні потоки:

1. якщо студент намагається надіслати завдання після встановленого терміну, система відображає повідомлення про помилку, яке вказує на те, що подання більше не приймається;
2. якщо студенту не вдається завантажити файл перед надсиланням, система запропонує студенту завантажити файл, запобігаючи надсиланню, доки завдання не буде належним чином вкладено;
3. якщо студент вирішить скасувати процес подання, він може натиснути кнопку «Скасувати», і система поверне його на сторінку керування курсом без збереження будь-яких змін.

У цьому сценарії описано етапи використання сценарію «Надіслати завдання», докладно описуючи взаємодію між учнем і системою, коли завдання надсилається на оцінку.

2.2.2 Діаграма послідовності. Діаграма послідовності — це тип діаграми взаємодії в уніфікованій мові моделювання (UML), яка зображує, як об’єкти взаємодіють протягом певного часу в певній послідовності подій. Ця діаграма підкреслює порядок, у якому обмінюються повідомленнями між

об’єктами, надаючи чітке уявлення про динамічну поведінку системи. Це особливо корисно для візуалізації потоку керування під час конкретного випадку використання або сценарію.

Ключові компоненти діаграми послідовності включають лінії життя, блоки активації, повідомлення, зворотні повідомлення та альтернативні шляхи. Лінії життя представляють різні об’єкти або учасників, залучених у взаємодію, зображені вертикальними пунктирними лініями з назвою об’єкта вгорі. Ці лінії простягаються вниз, щоб вказати плин часу, із взаємодіями, що відбуваються вздовж ліній. Поля активації або специфікації виконання — це прямокутники, розміщені на лініях життя, щоб вказати тривалість, протягом якої об’єкт активний і контролює потік взаємодії [13].

Повідомлення ілюструють зв’язок між об’єктами, представлені у вигляді горизонтальних стрілок, що з’єднують лінії життя. Кожна стрілка вказує на конкретну взаємодію, таку як виклик методу або відповідь, і зазвичай позначена назвою операції та будь-якими відповідними параметрами. Зворотні повідомлення, показані пунктирними стрілками, означають відповідь від об’єкта після обробки повідомлення, часто включаючи значення, що повертається. Крім того, діаграми послідовності можуть зображати альтернативні шляхи за допомогою таких конструкцій, як комбіновані фрагменти, які ілюструють умовні потоки, цикли та паралельні процеси [3].

Діаграми послідовності пропонують численні переваги. Вони забезпечують чітку візуалізацію того, як об’єкти взаємодіють з часом, спрощуючи розуміння потоку керування в системі. Зосереджуючись на порядку повідомлень, ці діаграми допомагають визначити потенційні проблеми з часом і взаємодії, які потребують координації. Вони також служать цінними інструментами документування, фіксуючи динамічну поведінку системи в легко зрозумілому форматі як для технічних, так і для нетехнічних зацікавлених сторін. Крім того, діаграми послідовності допомагають на етапі проектування, допомагаючи розробникам і архітекторам визначити необхідні взаємодії між об’єктами, що зрештою призводить до більш ефективної архітектури системи.

Діаграма послідовностей, зображена на рис. 4, містить такі об’єкти:

- “Студент”;
- “Викладач”;
- “Завдання”;
- “Оцінки”.

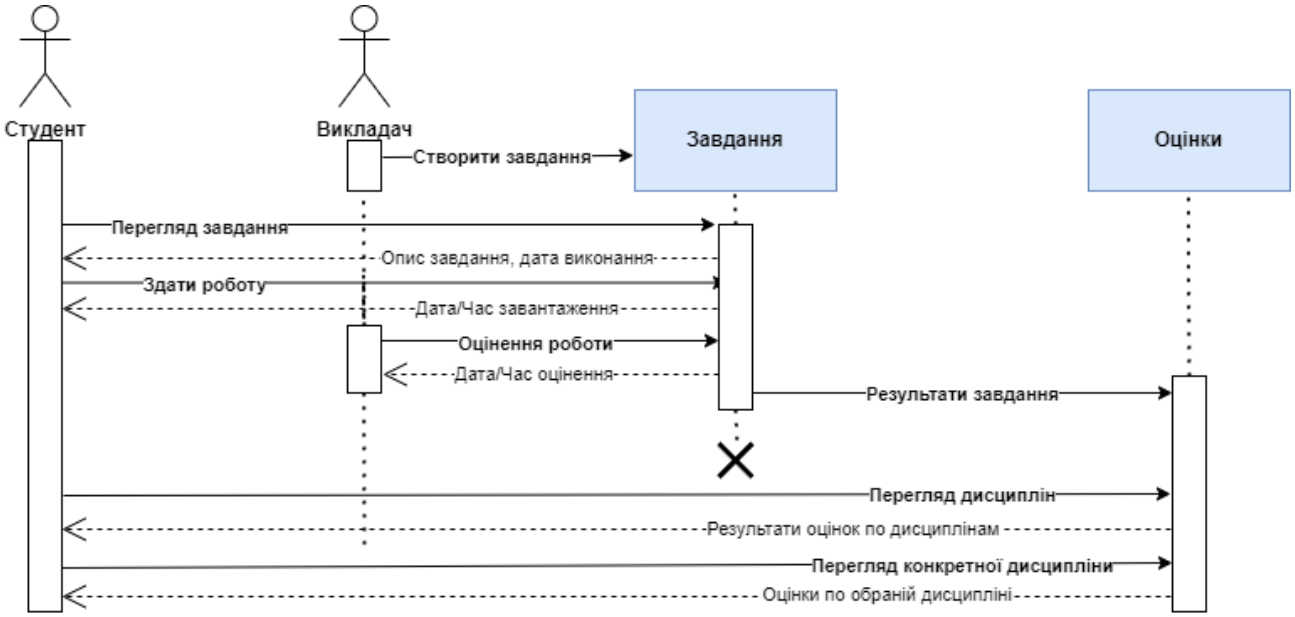

Рис. 4 Діаграма послідовності

Об’єкт «Студент» надсилає запит до об’єкту “Завдання” на перегляд завдання та отримує у відповідь опис завдання та дату виконання. Після цього він надсилає запит на здання роботи і отримує у відповідь дату та час завантаження.

Далі об’єкт «Викладач» надсилає запит до об’єкту “Завдання” на оцінення роботи і отримує у відповідь дату та час оцінення роботи.

Після цього об’єкт «Завдання» завершує свою роботу.

Наступним кроком Об’єкт «Завдання» надсилає запит до об’єкту “Оцінки” з результатами завдань.

Об’єкт «Студент» надсилає запит до об’єкту “Оцінки” на перегляд дисциплін та отримує у відповідь результати оцінок по дисциплінам. Після цього він надсилає запит на перегляд конкретної дисципліни і отримує у відповідь оцінки по обраній дисципліни.

2.2.3 Діаграма активності. Діаграма діяльності — це поведінкова

діаграма в рамках Уніфікованої мови моделювання (UML), яка візуально представляє потік дій або дій у системі чи процесі. Він ефективно ілюструє робочі процеси, показуючи, як послідовно виконуються завдання та як вони взаємодіють одне з одним. Діаграми діяльності особливо корисні для моделювання бізнес-процесів, варіантів використання та потоків керування складною системою.

Основні компоненти діаграми діяльності включають дії, переходи, початкові та кінцеві вузли, вузли прийняття рішень, розгалуження та з'єднання, а також доріжки. Діяльність позначає завдання або дії, які виконуються в процесі, представлені у вигляді округлених прямокутників, які часто супроводжуються описом дій, які виконуються. Переходи — це стрілки, що з'єднують ці дії, вказуючи на перехід від одного завдання до іншого, а також можуть містити умови або тригери, які диктують, коли відбувається перехід [18].

Початковий вузол позначає початкову точку діаграми активності, зображену у вигляді зафарбованого чорного кола, тоді як кінцевий вузол позначає кінець потоку діяльності, показаний у вигляді заповненого чорного кола, обведеного іншим колом. Вузли прийняття рішень представляють розгалуження в робочому процесі на основі конкретних умов, зображених у вигляді ромбів, які спрямовують потік до різних дій залежно від оцінки цих умов. Розгалуження та з'єднання дозволяють виконувати паралельну обробку; розгалуження розбиває один потік на кілька одночасних дій, тоді як об'єднання об'єднує кілька потоків назад в один шлях.

Доріжки — це горизонтальні або вертикальні поділки на діаграмі, які організовують діяльність за ролями, відділами чи системами, уточнюючи обов'язки та взаємодію між учасниками.

Діаграми діяльності пропонують численні переваги в системному моделюванні та аналізі. Вони забезпечують чітке візуальне представлення складних робочих процесів, полегшуючи розуміння послідовності дій і взаємодій у системі. Підкреслюючи потік контрольних точок і моментів прийняття рішень, ці діаграми допомагають виявити потенційні вузькі місця та

неефективність, що веде до областей для покращення.

Крім того, діаграми діяльності служать ефективними інструментами спілкування, сприяючи обговоренню між зацікавленими сторонами, такими як розробники, бізнес-аналітики та керівники проектів. Вони долають розрив між технічною та нетехнічною аудиторією, представляючи процеси в легкозасвоюваному форматі. Крім того, ці діаграми сприяють роботі з документуванням, фіксуючи динамічну поведінку системи, роблячи їх корисними посиланнями протягом життєвого циклу розробки програмного забезпечення. Вони також є цінними на етапі збору вимог, допомагаючи визначити та прояснити взаємодії, задіяні в конкретних випадках використання.

Розроблена діаграма активності представлена на рис.5.

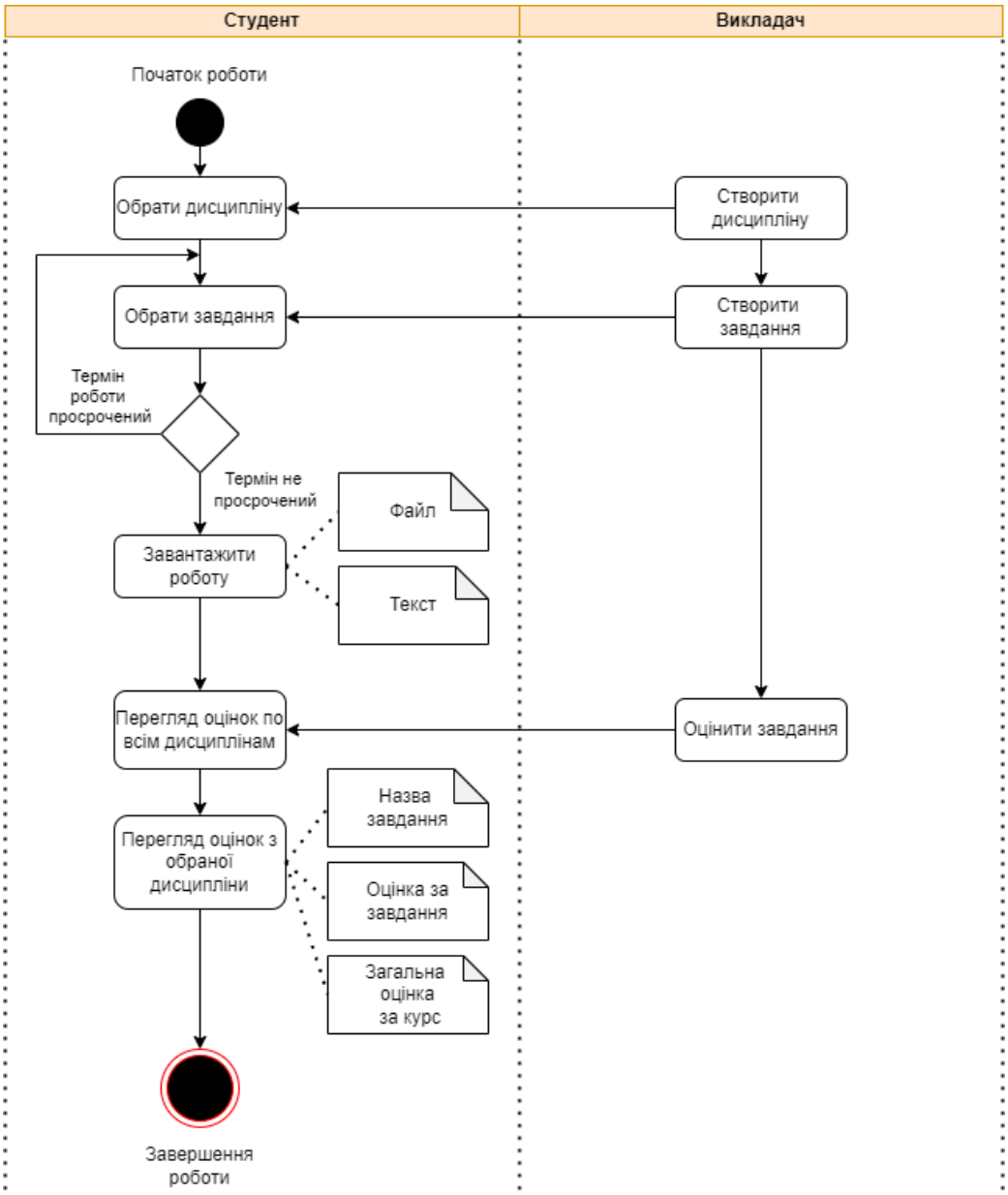

Рис. 5 Діаграма активності

2.2.4 Функціональна діаграма. Розробка, орієнтована на особливості (FDD) — це гнучка методологія, яка наголошує на постачанні відчутного робочого програмного забезпечення багаторазово та швидко за допомогою низки чітко визначених функцій. Діаграма FDD візуально представляє процес і структуру розробки, керованої функціями, зосереджуючись на ключових елементах, залучених до планування, проектування та впровадження функцій у проекті.

Основні компоненти діаграми FDD включають функції, класи та загальну архітектуру проекту. Функції – це конкретні функціональні можливості

або можливості, які забезпечують цінність для кінцевого користувача. Зазвичай вони пишуться в форматі, орієнтованому на користувача, наприклад «Як [роль користувача], я хочу [ціль], щоб [користь]». Кожна функція розроблена так, щоб бути достатньо маленькою, щоб її можна було завершити за короткий проміжок часу, зазвичай від кількох днів до тижня.

Класи представляють об’єкти домену, які підтримують реалізацію функцій. На діаграмі FDD класи організовані таким чином, що демонструє їхні стосунки та взаємодію один з одним. Це допомагає розробникам зрозуміти, як різні класи працюють разом, щоб забезпечити необхідну функціональність. Діаграми класів, часто частина FDD, ілюструють атрибути та методи кожного класу, надаючи розуміння структури системи.

Загальна архітектура проекту є критично важливим аспектом діаграми FDD, оскільки вона описує, як функції інтегровані в ширшу систему. Ця архітектура визначає задіяні компоненти, їх взаємодію, а також загальний потік даних і контроль у всій програмі. Надаючи чітке уявлення про структуру проекту, діаграма FDD допомагає гарантувати, що всі члени команди розуміють свої ролі та обов’язки щодо надання функцій.

Однією з ключових переваг використання діаграм FDD є їх зосередженість на поступовому досягненні цінності. Розбиваючи проект на менші керовані функції, команди можуть визначати пріоритетність роботи на основі потреб користувачів і швидко адаптуватися до змін. Цей підхід також сприяє співпраці між членами команди, оскільки розробники можуть працювати над різними функціями одночасно, зменшуючи вузькі місця та підвищуючи загальну продуктивність.

На даній діаграмі (рис. 6) проаналізовано функціонал системи.

Дана діаграма на показує основні процеси нашої системи:

- управління користувачами;
- завдання;
- дисципліна;
- оцінки.

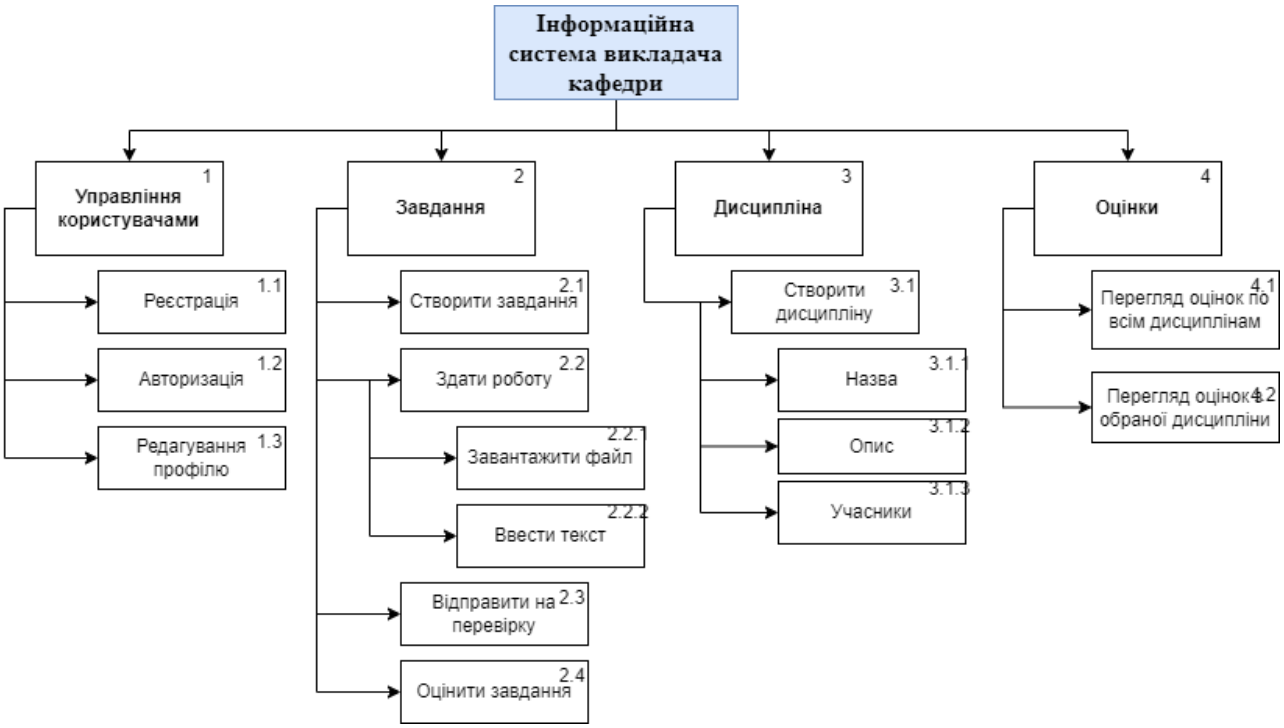

Рис. 6 FDD-діаграма

Перший процес “Управління користувачами” включає в себе:

1. реєстрація;
2. авторизація;
3. редагування профілю.

Другий процес “Завдання” включає в себе:

1. створити завдання;
2. здати роботу;
3. завантажити файл;
4. ввести текст;
5. надіслати на перевірку;
6. оцінити роботу.

Третій процес “Дисципліна” включає в себе:

1. створити дисципліну;
2. назва;
3. опис;
4. учасники.

Четвертий процес “Оцінки” включає в себе:

1. перегляд оцінок по всім дисциплінам;
2. перегляд оцінок з обраної дисципліни.

2.3 Абстракції предметної області

Абстракції предметної області відносяться до спрощених представлень або моделей реальних концепцій, сутностей або процесів у певній області. Ці абстракції фіксують основні характеристики та зв’язки між елементами цієї області, що дозволяє зосереджено розуміти проблемний простір.

У розробці програмного забезпечення абстракції служать для спрощення складності конкретного домену та забезпечують концептуальну основу для проектування та реалізації програмних систем. Виявляючи та моделюючи ключові концепції та їх взаємодію, абстракції домену допомагають узгодити програмні рішення з проблемами реального світу, які вирішуються.

Форми абстракцій домену можуть змінюватися залежно від використовуваного підходу моделювання. В об’єктно-орієнтованому програмуванні, наприклад, ці абстракції зазвичай представлені як класи та об’єкти, які інкапсулюють відповідні дані та поведінку. Такі абстракції часто виникають у результаті аналізу предметної області та використання предметно-специфічних знань, фіксуючи фундаментальні елементи та зв’язки в проблемному просторі [25].

Використовуючи абстракції, розробники можуть створювати програмні системи, які більш ефективно відповідають конкретним вимогам і характеристикам цільового домену. Цей підхід покращує розуміння, спрощує технічне обслуговування та покращує зв’язок між експертами домену та командами розробників програмного забезпечення.

Для системи, що розробляється, були розроблені конкретні абстракції, які зображені на малюнку 7.

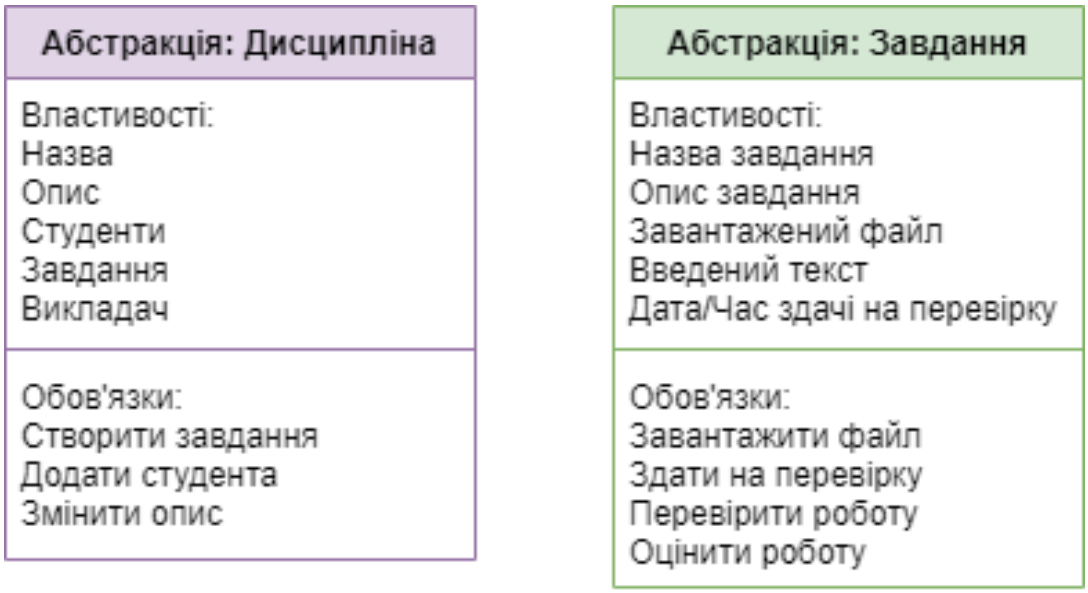

Рис. 7 Абстракції «дисципліна» та “Завдання”

2.4 Діаграма класів

Діаграма класів є ключовим компонентом уніфікованої мови моделювання (UML), яка пропонує візуальне представлення класів у системі та їхніх взаємозв’язків. Ця діаграма служить базовим планом для архітектури системи, наголошуючи на різних компонентах та їх взаємодії. Діаграми класів широко використовуються в об’єктно-орієнтованому проектуванні та розробці програмного забезпечення для ефективного моделювання структури системи.

В основі діаграми класів лежать класи, які діють як основні будівельні блоки об’єктно-орієнтованої системи. Кожен клас представлений у вигляді прямокутника, поділеного на три частини: у верхній частині відображається ім’я класу, у середній частині представлені атрибути (властивості), пов’язані з класом, а в нижній частині детально описуються методи (функції або поведінка), які може виконувати клас. Цей упорядкований макет передає важливу інформацію про кожен клас та його функціональність [28].

Діаграми класів також ілюструють зв’язки, які існують між класами, які можуть мати різні форми. Загальні типи зв’язків включають асоціацію, агрегацію, композицію, успадкування та залежність.

Було створено власну діаграму класів за об'єктно орієнтованим підходом для даного веб-сайту (рис. 8).

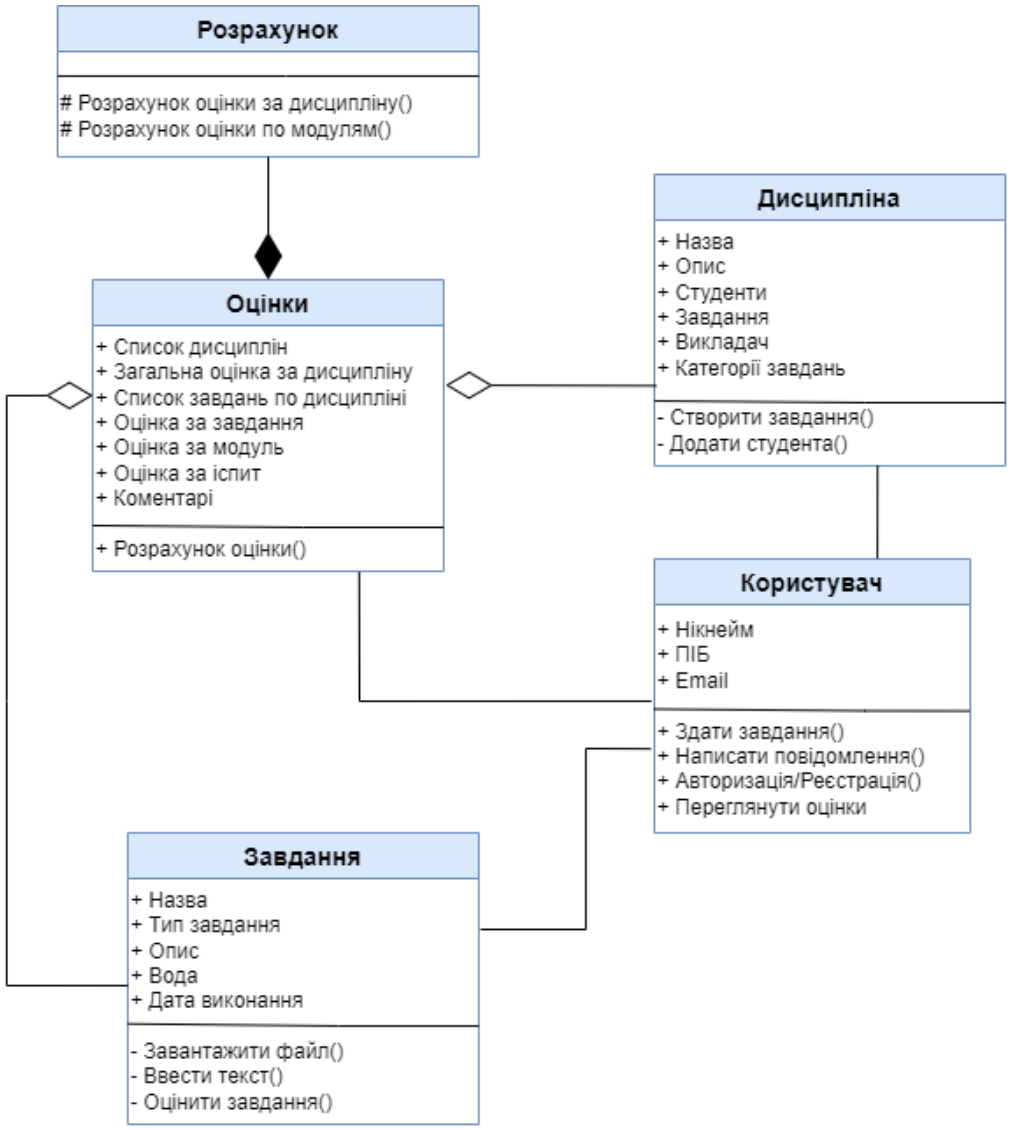

Рис. 8 Діаграма класів

3 ПРОЄКТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

3.1 Логічна модель даних

ERwin — це потужний інструмент моделювання даних, який широко використовується для проектування, візуалізації та керування базами даних. Він забезпечує комплексне середовище для створення діаграм сутності та зв’язку (ER), які є важливими для розуміння та структурування архітектури даних системи. ERwin є особливо корисним для адміністраторів баз даних, архітекторів даних і розробників, яким потрібно проектувати та впроваджувати ефективні рішення для баз даних.

Однією з ключових особливостей ERwin є його здатність полегшувати створення детальних діаграм ER, які візуально представляють сутності, атрибути та зв’язки в базі даних. Користувачі можуть легко визначати сутності, вказувати їхні атрибути та встановлювати зв’язки між ними. Це візуальне представлення допомагає зрозуміти модель даних і підтримує ефективне спілкування між членами команди та зацікавленими сторонами.

ERwin підтримує різні системи керування базами даних (СУБД), дозволяючи користувачам генерувати логічні та фізичні моделі, сумісні з такими популярними платформами, як Oracle, SQL Server, MySQL тощо. Ця сумісність гарантує, що користувачі можуть проектувати свої бази даних відповідно до конкретних вимог і особливостей обраної ними СУБД.

На додаток до своїх можливостей моделювання, ERwin пропонує функції для зворотного проектування існуючих баз даних. Це дозволяє користувачам імпортувати існуючу схему бази даних і автоматично генерувати діаграму ER, полегшуючи розуміння та документування застарілих систем. Ця функція є безцінною для організацій, які хочуть модернізувати свою архітектуру даних або перейти на нові технології баз даних.

ERwin також включає можливості керування даними та метаданими. Він допомагає організаціям підтримувати якість і узгодженість даних, надаючи

інструменти для визначення стандартів даних, керування походженням даних і відстеження змін у моделі даних. Ці функції мають важливе значення для забезпечення відповідності нормам і збереження цілісності даних в організації.

Співпраця є ще однією сильною стороною ERwin. Інструмент підтримує командне моделювання, дозволяючи кільком користувачам працювати над однією моделлю даних одночасно. Цей спільний підхід оптимізує процес проектування, сприяє спілкуванню між членами команди та гарантує, що кожен узгоджується з архітектурою даних.

Крім того, ERwin надає функції звітності та документування, що дозволяє користувачам створювати повну документацію для своїх моделей даних. Ця документація може містити деталі про сутності, атрибути, зв’язки та визначення даних, що полегшує для команд розуміння та підтримку бази даних з часом [11].

Таким чином, ERwin — це надійний інструмент моделювання даних, який дозволяє користувачам проектувати, візуалізувати та ефективно керувати базами даних. Завдяки підтримці створення діаграм ER, сумісності з різними СУБД, можливостям зворотного проектування, функціям керування даними та інструментам співпраці, ERwin є безцінним ресурсом для організацій, які прагнуть оптимізувати свою архітектуру даних і забезпечити цілісність даних.

Логічна модель системи представлена на рис. 9.

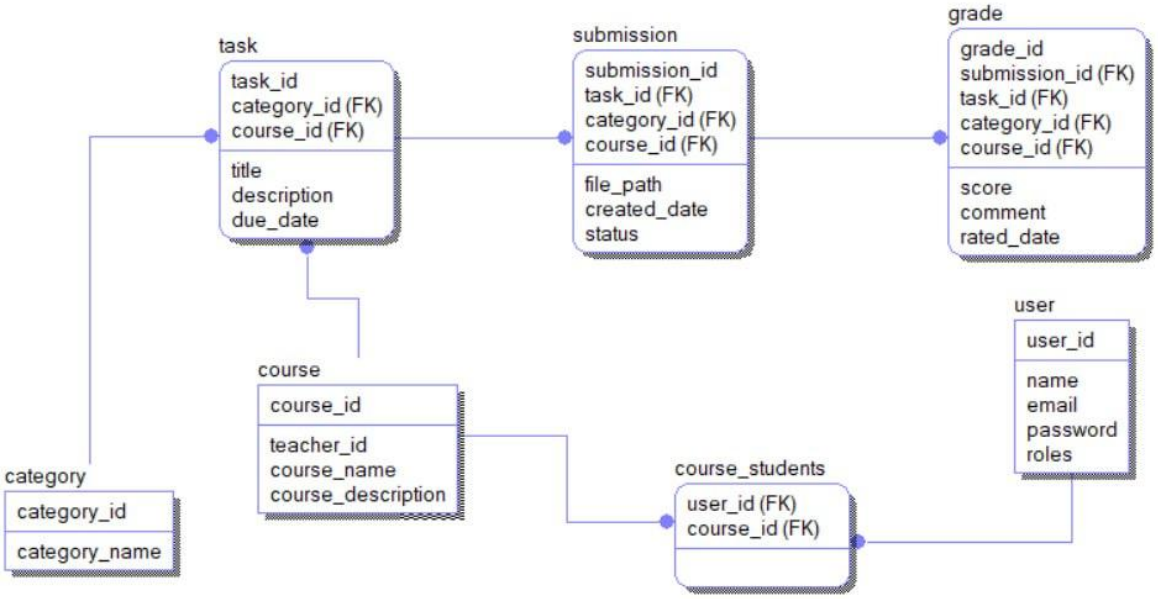

Рис. 9 ER-діаграма

Логічна модель складається з таких сутностей:

User:

- id: int - Унікальний ідентифікатор користувача;
- name: string - Ім'я користувача;
- email: string - Email користувача;
- password: string - Хешований пароль користувача;
- roles: array - Ролі користувача (наприклад, ROLE_ADMIN, ROLE_STUDENT).

Course:

- id: int - Унікальний ідентифікатор дисципліни;
- courseName: string - Назва дисципліни;
- teacher: User – викладач, який веде дисципліну.

CourseStudents:

- id: int - Унікальний ідентифікатор запису про студента в дисципліні;
- course_id: Course – Дисципліна;
- student_id: User – Студент.

Task:

- id: int - Унікальний ідентифікатор завдання;
- title: string - Назва завдання;
- category: string - Категорія завдання;
- dueDate: DateTime - Дата, до якої необхідно здати завдання;
- course: Course.

Submission:

- id: int - Унікальний ідентифікатор виправки завдання;
- filePath: string - завантажений файл завдання;
- student: User – Студент;
- task: Task – Завдання;
- status: int - Статус відповіді.

Grade:

- id: int - Унікальний ідентифікатор оцінки;
- score: float - Оцінка за виконання завдання;
- comment: string – коментар;
- submission: Submission - Зв'язок з відправкою завдання.

У сутності Course є зв'язок з User через атрибут teacher, що вказує на викладача, який веде цю дисципліну. Сутність CourseStudents пов'язує студентів та дисципліни: кожен студент (посилання на User) може бути доданий у кілька дисциплін, а кожна дисципліна може містити безліч студентів.

Кожна Course може мати кілька Task, що означає завдання, що стосуються цієї дисципліни. Кожне завдання може бути надіслано кількома студентами, що здійснюється через суть Submission. Сутність Submission пов'язує студентів (посилання на User) та завдання (посилання на Task) та зберігає інформацію про статус відправки та файл, завантажений студентом.

Сутність Grade пов'язує оцінки з відправками завдань, вказуючи на конкретну Submission та надаючи оцінку та коментар викладача. Це створює повний ланцюжок від викладача до студента через завдання та оцінки.

3.2 Вибір системи управління базою даних та її реалізація

Реляційна база даних — це структурована система, яка організовує дані в таблицях, використовуючи рядки та стовпці для створення чіткого та ефективного формату. Кожна таблиця представляє окрему сутність, наприклад клієнтів, замовлення або продукти, і містить дані, що стосуються цієї сутності. Ця модель, представлена Едгаром Ф. Коддом у 1970-х роках, базується на принципах реляційної алгебри.

У реляційних базах даних дані зберігаються в таблицях, які складаються з окремих записів, представлених рядками, і атрибутів, представлених стовпцями. Кожна таблиця має первинний ключ, який унікально ідентифікує кожен запис, гарантуючи, що кожен запис можна відрізнити від інших. Цей унікальний ідентифікатор сприяє ефективному пошуку та маніпулюванню даними.

Для встановлення зв’язків між таблицями реляційні бази даних використовують зовнішні ключі. Зовнішній ключ в одній таблиці посилається на первинний ключ в іншій таблиці, таким чином створюючи зв’язок між ними. Ця структура дозволяє користувачам об’єднувати дані з кількох таблиць, забезпечуючи повний перегляд пов’язаної інформації.

Нормалізація — це важливий процес у реляційних базах даних, спрямований на зменшення надмірності та підвищення цілісності даних. Це передбачає розбиття великих таблиць на менші, взаємопов’язані та визначення зв’язків між ними. Упорядковуючи дані таким чином, нормалізація забезпечує ефективне зберігання та сприяє легкому оновленню без внесення невідповідностей.

Структурована мова запитів (SQL) служить стандартним методом для взаємодії з реляційними базами даних. Користувачі можуть виконувати різноманітні операції, зокрема запитувати дані, вставляти нові записи, оновлювати існуючі та видаляти записи. Крім того, SQL дозволяє користувачам визначати структуру бази даних і керувати правами доступу.

Реляційні бази даних пропонують численні переваги. Вони підтримують цілісність даних за допомогою первинних і зовнішніх ключів, забезпечуючи точну та послідовну інформацію в усій системі. Гнучкість цих баз даних дозволяє легко модифікувати їх структуру, наприклад додавати або видаляти таблиці та атрибути, без суттєвого порушення існуючих даних.

Крім того, SQL надає можливість виконувати складні запити та маніпулювати даними, завдяки чому можна швидко отримувати певну інформацію з великих наборів даних. Реляційні бази даних також є масштабованими, здатними обробляти значні обсяги даних, підтримуючи одночасний доступ кількох користувачів до системи [21].

Типовими прикладами систем керування реляційними базами даних (RDBMS) є MySQL, PostgreSQL, Microsoft SQL Server і Oracle Database. Ці системи пропонують необхідні інструменти та інфраструктури для ефективного проектування, впровадження та керування реляційними базами даних, що задовольняють широкий спектр програм у різних галузях.

Рішення обрати MySQL для інформаційно-управляючої системи викладача ґрунтувалося на кількох переконливих факторах, які відповідають вимогам і цілям проекту.

По-перше, MySQL — це система керування реляційною базою даних з відкритим вихідним кодом, яка забезпечує значні економічні переваги для розробки та розгортання. Ця характеристика забезпечує бюджетне впровадження без шкоди для якості чи функціональності.

По-друге, MySQL відома своєю надійністю та міцністю. Він має перевірений досвід роботи з великими обсягами даних і підтримки додатків із високим трафіком, що робить його придатним для освітньої системи, яка може зазнавати різного навантаження від користувачів, зокрема студентів і викладачів.

Крім того, MySQL пропонує чудову продуктивність, особливо в операціях з інтенсивним читанням, що є важливим для інформаційної системи, яка потребує швидкого доступу до записів студентів, оцінок та інформації про курси. Його здатність оптимізувати продуктивність запитів за допомогою

індексування та кешування ще більше підвищує його придатність для нашого проекту.

Крім того, сильна підтримка спільноти MySQL і обширна документація полегшують розробку. Розробники можуть легко знайти ресурси, навчальні посібники та форуми, щоб допомогти у вирішенні проблем та оптимізації бази даних. Ця мережа підтримки є безцінною для забезпечення плавного процесу розробки та вирішення будь-яких проблем, які можуть виникнути.

Ще однією важливою перевагою MySQL є її сумісність з різними мовами програмування та фреймворками, включаючи PHP, який використовується для розробки нашого проекту. Ця сумісність забезпечує безперебійну інтеграцію між базою даних і програмою, спрощуючи робочий процес розробки.

Безпека також має важливе значення при виборі MySQL. Система керування базами даних забезпечує надійні функції безпеки, включаючи автентифікацію користувачів, контроль доступу та шифрування даних. Ці функції допомагають захистити конфіденційну інформацію студентів і викладачів, забезпечуючи дотримання правил захисту даних.

Нарешті, MySQL підтримує масштабованість, що має вирішальне значення для нашого проекту, оскільки він може вмістити майбутнє зростання. Оскільки база користувачів розширюється та генерується більше даних, MySQL можна легко масштабувати, щоб задовольнити зростаючі вимоги, не вимагаючи повної переробки системи.

Підсумовуючи, рішення використовувати MySQL для інформаційної системи викладача кафедри було обумовлено її економічною ефективністю, надійністю, продуктивністю, підтримкою спільноти, сумісністю з інструментами розробки, функціями безпеки та масштабованістю. Ці атрибути роблять MySQL ідеальним вибором для створення надійної та ефективної освітньої системи.

Під час проєктування таблиць у фізичній моделі основою слугували сутності з логічної моделі. Атрибути цих сутностей були використані для розробки структури таблиці. Отриману схему бази даних зображено на рис. 10.

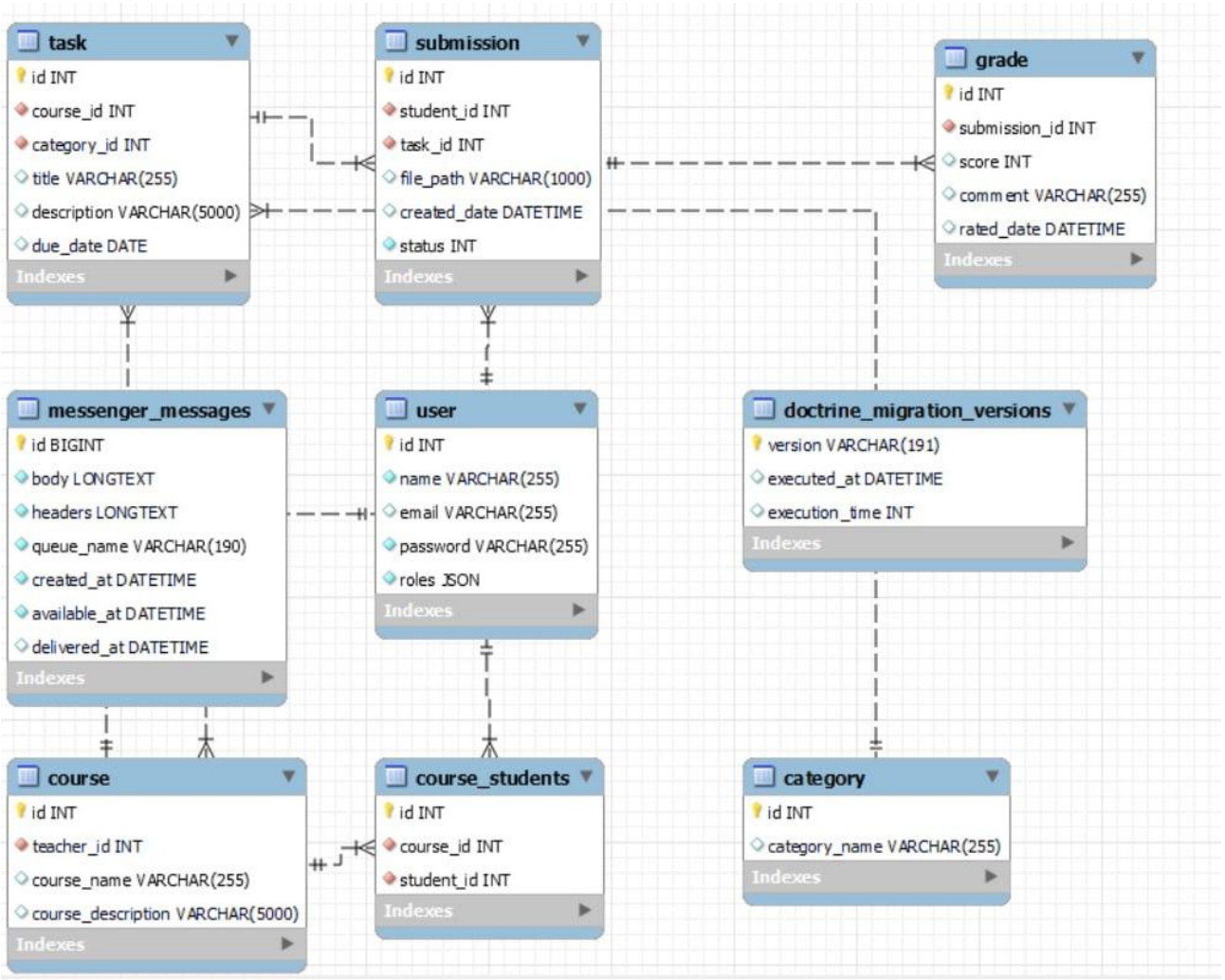

Рис. 10 База даних системи

3.3 Архітектура програмного забезпечення

Архітектура програмного забезпечення відноситься до фундаментальної структури програмної системи, окреслюючи її компоненти та їх взаємодію. Можна використовувати різні архітектурні стилі на основі конкретних вимог, потреб у масштабованості та цілей щодо ремонтопридатності проекту [6].

Монолітна архітектура складається з єдиної уніфікованої програми, де всі компоненти, включаючи інтерфейс користувача, бізнес-логіку та доступ до даних, взаємопов’язані та розгорнуті разом. Такий підхід спрощує розробку та розгортання; однак це може стати складним для масштабування та підтримки в міру зростання програми [5].

Навпаки, архітектура мікросервісів розбиває програму на менші незалежні служби, кожна з яких зосереджена на певній функціональності. Це дозволяє незалежну розробку, розгортання та масштабування кожного мікросервісу, сприяючи гнучкості та стійкості. Команди можуть працювати над різними сервісами одночасно, підвищуючи загальну швидкість розробки [27].

Рівнева архітектура, також відома як n-рівнева архітектура, поділяє програму на окремі рівні з певними обов’язками, наприклад рівень презентації, рівень бізнес-логіки та рівень доступу до даних. Таке поділ проблем покращує технічне обслуговування, оскільки оновлення можна робити на окремих рівнях, не впливаючи на всю систему.

Архітектура, керована подіями, зосереджена на компонентах, які спілкуються через події. Коли компонент генерує подію, інші компоненти можуть підписатися на цю подію та реагувати на неї. Це роз’єднує компоненти, сприяючи більшій масштабованості та гнучкості, і особливо підходить для систем, які потребують обробки в реальному часі або асинхронного зв’язку.

Сервісно-орієнтована архітектура (SOA) має спільні риси з мікросервісами, але зазвичай включає більші, складніші сервіси, які можна повторно використовувати в різних програмах. Служби обмінюються даними через стандартизовані протоколи та повідомлення, сприяючи взаємодії та гнучкості. SOA часто використовується в програмах корпоративного рівня, які потребують інтеграції з різними системами.

Безсерверна архітектура дозволяє розробникам писати код, який виконується у відповідь на події, не керуючи серверною інфраструктурою. Хмарні постачальники забезпечують виконання, масштабування та керування ресурсами, що дозволяє розробникам зосередитися на кодуванні. Ця архітектура є економічно ефективною та масштабованою, що робить її ідеальною для програм із змінним навантаженням.

Архітектура клієнт-сервер розділяє програму на два основні компоненти: клієнт і сервер. Клієнт взаємодіє з користувачем і надсилає запити

на сервер, який обробляє ці запити та повертає результати. Ця модель забезпечує централізоване керування та обробку даних, пропонуючи зручний інтерфейс.

Архітектура однорангового зв’язку (P2P) дозволяє кожному вузлу в мережі діяти і як клієнт, і як сервер, обмінюючись ресурсами та послугами безпосередньо з іншими вузлами. Цей децентралізований підхід підвищує стійкість і оптимізує використання ресурсів, що зазвичай використовується в програмах для обміну файлами та технологіях блокчейн.

Нарешті, архітектура модель-подання-контролер (MVC) — це шаблон проектування, який поділяє програму на три взаємопов’язані компоненти: модель (дані), подання (інтерфейс користувача) і контролер (бізнес-логіка). Ця структура дозволяє легше керувати складними програмами та сприяє паралельній розробці.

Вибір відповідної архітектури програмного забезпечення має життєво важливе значення для успіху проекту, оскільки це суттєво впливає на продуктивність системи, масштабованість, придатність до обслуговування та загальну взаємодію з користувачем. Вибір часто залежить від конкретних потреб програми, цілей, досвіду та ресурсів команди розробників.

Рішення прийняти клієнт-серверну архітектуру для інформаційної системи викладача кафедри було зумовлене кількома ключовими міркуваннями, які узгоджуються з цілями та вимогами проекту.

Однією з головних причин вибору клієнт-серверної архітектури є її здатність розділити проблеми між інтерфейсом користувача та серверною обробкою. У цій моделі клієнт обробляє презентаційний рівень, взаємодіючи безпосередньо з користувачами та збираючи їхні вхідні дані, тоді як сервер керує зберіганням даних, обробкою та бізнес-логікою. Цей чіткий поділ спрощує розробку та полегшує технічне обслуговування та оновлення, оскільки зміни на клієнті чи сервері можна вносити незалежно один від одного, не впливаючи на одне інше.

Масштабованість є ще одним важливим фактором, який вплинув на вибір клієнт-серверної архітектури. Оскільки база користувачів інформаційної

системи зростає, сервер можна масштабувати відповідно до зростаючих вимог. Ця архітектура дозволяє додавати більше серверів або ресурсів за потреби, гарантуючи, що система може обробляти більші навантаження без шкоди для продуктивності. Така гнучкість є важливою для освітньої платформи, яка може зазнавати різного рівня активності користувачів протягом навчального року.

Крім того, клієнт-серверна архітектура підвищує безпеку. Централізувавши зберігання та обробку даних на сервері, можна краще захистити конфіденційну інформацію, таку як записи студентів і оцінки. Сервер може впроваджувати надійні заходи безпеки, включаючи автентифікацію та контроль доступу, гарантуючи, що лише авторизовані користувачі можуть отримати доступ до даних або маніпулювати ними. Цей рівень безпеки має вирішальне значення для підтримки цілісності інформаційної системи та захисту конфіденційності користувачів.

Ще однією перевагою клієнт-серверної моделі є підтримка одночасного доступу користувачів. Кілька клієнтів можуть підключатися до сервера одночасно, що дозволяє багатьом користувачам одночасно взаємодіяти з системою. Ця можливість є життєво важливою для інформаційної системи, розробленої як для викладачів, так і для студентів, що дозволяє їм отримувати доступ до матеріалів курсу, подавати завдання та переглядати оцінки без затримок або збоїв.

Крім того, клієнт-серверна архітектура сприяє ефективному управлінню даними. Сервер може обробляти складні запити та завдання обробки, дозволяючи клієнтам швидко й ефективно отримувати інформацію. Цей централізований підхід також спрощує процеси резервного копіювання та відновлення даних, забезпечуючи постійну доступність і захист критичної інформації.

Нарешті, клієнт-серверна архітектура добре узгоджується з технологіями та інструментами, які використовуються в проекті. Завдяки тому, що система розробляється на PHP і MySQL, клієнт-серверна модель забезпечує

безперебійну інтеграцію між інтерфейсом і бекендом. Ця сумісність оптимізує розробку та забезпечує взаємодію користувача 7.

Підсумовуючи, на вибір клієнт-серверної архітектури для інформаційної системи викладача кафедри вплинула її здатність розділяти проблеми, масштабованість, підвищена безпека, підтримка одночасних користувачів, ефективне управління даними та сумісність з обраними технологіями. Ці атрибути роблять архітектуру клієнт-сервер ідеальною для розробки надійної та ефективної освітньої платформи.

Нижче буде продемонстровано 3-шарова архітектура даного програмного забезпечення (рис. 11)

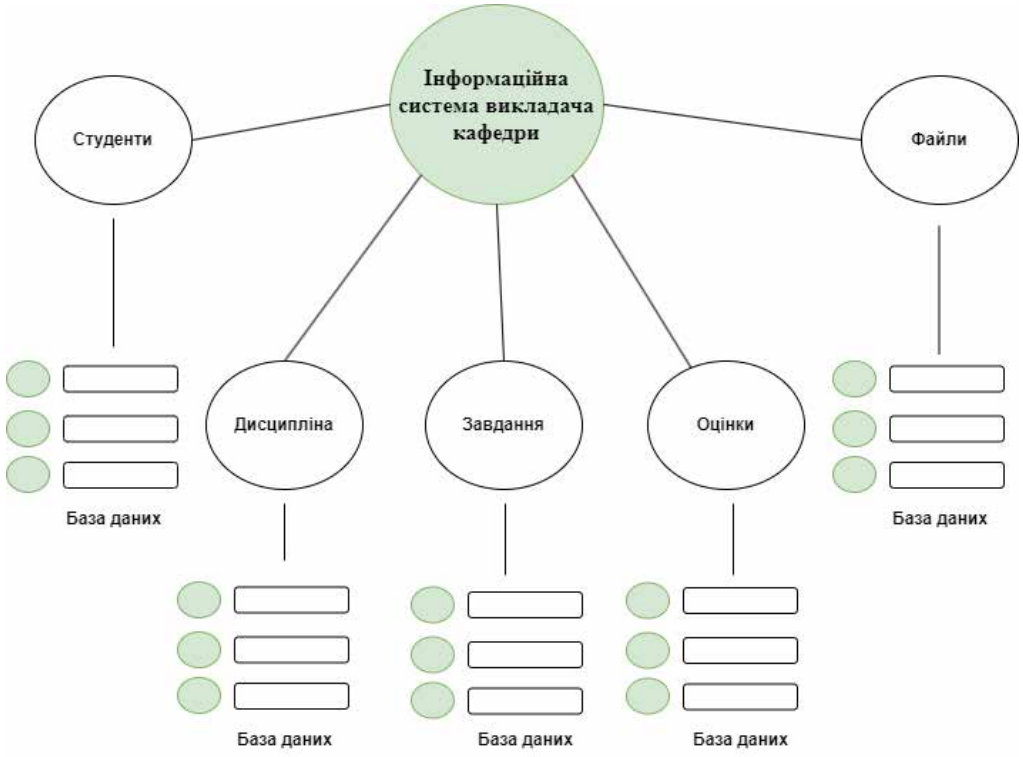

Рис. 11 Тришарова архітектура веб-сайту

З рисунка вище ми бачимо, що наша система побудована на 3-рівневій багатошаровій архітектурі, що складається з презентаційного рівня, бізнес-рівня та рівня бази даних.

Інформація в системі поступово переходить від рівня до наступного рівня по черзі.

Перший рівень - це інтерфейс користувача, тобто екран комп'ютера або телефона.

Другий рівень - це основний функціонал програми, з яким працює користувач. На цьому рівні він вносить свої дані, може створювати дисципліни та завдання, здавати роботи на перевірку, оцінювати роботи та переглядати оцінки.

І третій рівень, безпосередньо, база даних, в який зберігається вся інформація.

3.4 Організаційна структура програмного забезпечення

Діаграма пакетів — це структурна діаграма на уніфікованій мові моделювання (UML), яка організовує та ілюструє зв’язки між різними пакетами в програмній системі. Пакети служать контейнерами для пов’язаних класів, інтерфейсів та інших елементів, що дозволяє розробникам керувати складністю та підвищувати модульність у великомасштабних програмах [29].

Основна функція діаграми пакетів — показати, як ці пакунки взаємодіють один з одним, і зобразити їхні залежності. Це візуальне представлення допомагає зрозуміти загальну архітектуру системи, підкреслюючи зв’язки та взаємодію між її різними компонентами. Основні аспекти пакетних діаграм включають наступне:

Пакети зображені у вигляді прямокутників із вкладкою вгорі, що містить пов’язані класи, інтерфейси та, можливо, інші пакунки. Така організація сприяє інкапсуляції та допомагає підтримувати чітку структуру всередині системи [8].

Залежності представлені пунктирними стрілками, які вказують на те, що один пакет залежить від іншого. Цей зв’язок передбачає, що зміни в залежному пакеті можуть вплинути на пакет, який залежить від нього, підкреслюючи важливість розуміння цих зв’язків .

Пакети також можуть визначати рівні видимості свого вмісту, наприклад загальнодоступний, приватний і захищений. Ця функція має вирішальне значення для інкапсуляції, оскільки вона контролює доступ до класів та інтерфейсів, забезпечуючи цілісність системи.

Діаграма пакетів (рис. 12) для даного програмного забезпечення використовується для візуального представлення архітектури та організації системи. Діаграма пакетів зображує різні пакети, які складають програмне забезпечення, і ілюструє зв’язки та залежності між ними [10].

Рис. 12 Діаграма пакетів

На етапі проектування розробки програмного забезпечення діаграми пакетів виявляються особливо цінними. Вони забезпечують загальне уявлення про архітектуру системи, допомагаючи зацікавленим сторонам зрозуміти взаємодію між різними компонентами та те, як структурувати систему для оптимальної продуктивності та зручності обслуговування. Використовуючи діаграми пакетів, розробники можуть визначити потенційні області для рефакторингу, підвищити модульність і покращити спілкування між членами команди [23].

Підсумовуючи, діаграми пакетів є ефективними інструментами для візуалізації організації та залежностей пакетів у програмній системі. Вони відіграють вирішальну роль у допомозі при проектуванні, документуванні та сприянні спілкуванню між розробниками.

3.5 Вибір інструментарію для створення програмного забезпечення

Вибір інструментів розробки для інформаційної системи викладача кафедри був здійснений з урахуванням вимог проекту, досвіду команди та потреби у надійній та масштабованій програмі. Вибрані інструменти включають PHP, Symfony, MySQL, Doctrine ORM, HTML, CSS, JavaScript і Figma, кожен з яких відіграє вирішальну роль у процесі розробки.

PHP було обрано як основну мову програмування для серверної розробки через її широке використання у веб-розробці та потужну підтримку спільноти. Його здатність легко інтегруватися з різними базами даних і фреймворками робить його ідеальним вибором для створення динамічних веб-додатків. Крім того, простота та гнучкість PHP дозволяють швидко розробляти та легко підтримувати, що є важливим для освітньої платформи [30].

Symfony було обрано як основу для оптимізації процесу розробки. Symfony, відома своєю модульною архітектурою та багаторазовими компонентами, підвищує продуктивність, забезпечуючи міцну основу для створення складних програм. Його вбудовані функції, такі як маршрутизація, створення шаблонів і безпека, дозволяють розробникам зосередитися на реалізації бізнес-логіки, а не на повторюваних завданнях. Крім того, надійна екосистема Symfony і сильна підтримка спільноти полегшують вирішення проблем і надають доступ до великої кількості ресурсів.

MySQL було обрано як систему керування базами даних через її надійність, продуктивність і простоту використання. Як одна з найпопулярніших систем реляційних баз даних, MySQL пропонує перевірену історію роботи з великими обсягами даних і підтримки одночасних користувачів. Його сумісність із PHP і Symfony забезпечує безперебійну взаємодію з даними та керування ними, що робить його ідеальним вибором для зберігання та отримання інформації, пов’язаної з курсами, завданнями та взаємодією користувачів.

Щоб спростити взаємодію з базою даних і підвищити придатність коду, Doctrine ORM було інтегровано в проект. Doctrine забезпечує рівень об’єктно-реляційного відображення (ORM), що дозволяє розробникам працювати із записами бази даних як об’єктами PHP. Ця абстракція спрощує маніпулювання даними та сприяє більш чистому коду, зменшуючи ймовірність помилок і покращуючи загальну ефективність розробки.

Для розробки інтерфейсу було вибрано HTML, CSS і JavaScript, щоб створити адаптивний і зручний інтерфейс. HTML забезпечує структуру веб-сторінок, а CSS використовується для стилізації та забезпечення візуально привабливого дизайну. JavaScript додає інтерактивності та покращує взаємодію з користувачем, увімкнувши динамічне оновлення вмісту та перевірку на стороні клієнта. Разом ці технології дозволяють створити інтуїтивно зрозумілу та привабливу платформу як для вчителів, так і для учнів.

Figma була обрана як інструмент розробки для полегшення співпраці між дизайнерами та розробниками. Хмарна платформа Figma забезпечує співпрацю в режимі реального часу, що полегшує членам команди обмінюватися ідеями щодо дизайну, надавати відгуки та змінювати інтерфейс користувача. Цей інструмент оптимізує процес проектування, гарантуючи, що кінцевий продукт відповідає очікуванням користувачів і забезпечує зручність роботи.

Таким чином, вибір засобів розробки інформаційної системи викладача кафедри був зумовлений потребою в надійній, масштабованій та зручній програмі. Використовуючи PHP, Symfony, MySQL, Doctrine ORM, HTML, CSS, JavaScript і Figma, команда розробників прагне створити ефективну платформу, яка відповідає вимогам як викладачів, так і студентів, зрештою покращуючи навчальний досвід [26].

4 ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЯ СИСТЕМИ

4.1 Вимоги до апаратного та програмного забезпечення

Діаграма розгортання — це структурна діаграма в рамках уніфікованої мови моделювання (UML), яка візуально представляє фізичне розгортання компонентів програмного забезпечення на різних вузлах системи. Його основна функція полягає в тому, щоб проілюструвати, як різні елементи програмного забезпечення розподіляються в апаратному та програмному середовищах, надаючи розуміння архітектури та конфігурації системи.

На діаграмі розгортання показано фізичне розміщення як програмного, так і апаратного забезпечення, демонструючи взаємодію між різними компонентами в контексті реального світу. Це включає деталізацію зв’язків між програмними артефактами, такими як виконувані файли, бібліотеки та бази даних, і вузлами, які можуть бути серверами, пристроями або віртуальними машинами, де розміщені ці артефакти.

Одним із ключових елементів діаграми розгортання є вузли, які представляють фізичне або віртуальне обладнання, на якому розміщені компоненти програмного забезпечення. Зображені у вигляді тривимірних кубів або прямокутників, вузли можуть включати сервери, робочі станції або хмарні екземпляри, що ілюструє інфраструктуру, на якій працює додаток.

Артефакти є ще одним важливим компонентом, який представляє фізичні файли або виконувані файли, розгорнуті на вузлах. Вони можуть варіюватися від бінарних файлів програми та конфігураційних файлів до баз даних та інших критичних елементів системи. На діаграмі артефакти зазвичай відображаються як прямокутники, з’єднані з відповідними вузлами.

Крім того, діаграми розгортання ілюструють зв’язки між вузлами та артефактами, включаючи шляхи зв’язку, такі як мережеві з’єднання та залежності, які вказують, як різні компоненти взаємодіють один з одним. Стереотипи також можуть бути застосовані до вузлів і артефактів, щоб

забезпечити контекст щодо їхніх ролей у системі, наприклад, позначити вузол як «Веб-сервер» або «Сервер бази даних», щоб уточнити його функцію.

Діаграма може додатково визначати конфігурацію розгортання, деталізуючи кількість екземплярів для конкретного вузла або артефакту, а також будь-які заходи для реплікації або кластеризації.

Діаграми розгортання особливо цінні на етапі розгортання розробки програмного забезпечення. Вони допомагають командам візуалізувати інфраструктуру, необхідну для запуску програми, сприяючи кращому розумінню того, як компоненти взаємодіятимуть у робочому середовищі. Це допомагає в плануванні та координації зусиль команд розвитку, операцій та інфраструктури.

На рис. 13 зображено діаграму розгортання даної системи. Клієнт-серверна архітектура реалізована на двох окремих серверах.

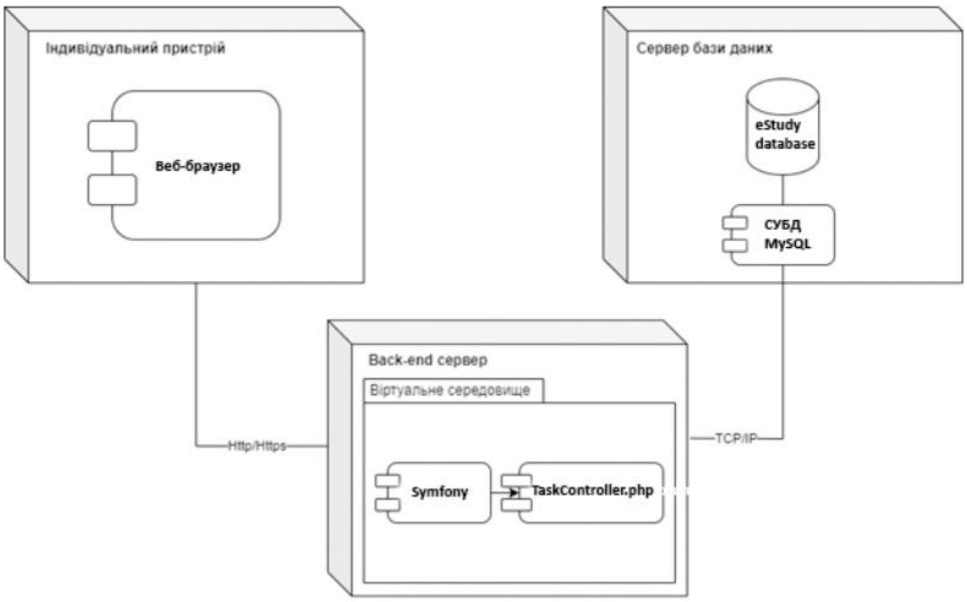

Рис. 13 Діаграма розгортання

Апаратні вимоги які необхідні бути у комп’ютера для функціонування програмного забезпечення:

- процесор із принаймні двома ядрами, наприклад Intel Core i3 або AMD Ryzen 3l
- оперативна пам’ять 4 ГБ;

- SSD 10 ГБ або більше.

4.2 Тестування системи

Запустивши систему, бачимо головну сторінку програмного забезпечення (рис. 14-15). На ній ми бачимо список дисциплін, до яких ми маємо доступ і внизу бачимо календар, на якому зображені роботи дедлайн яких стоїть до певного дня [4].

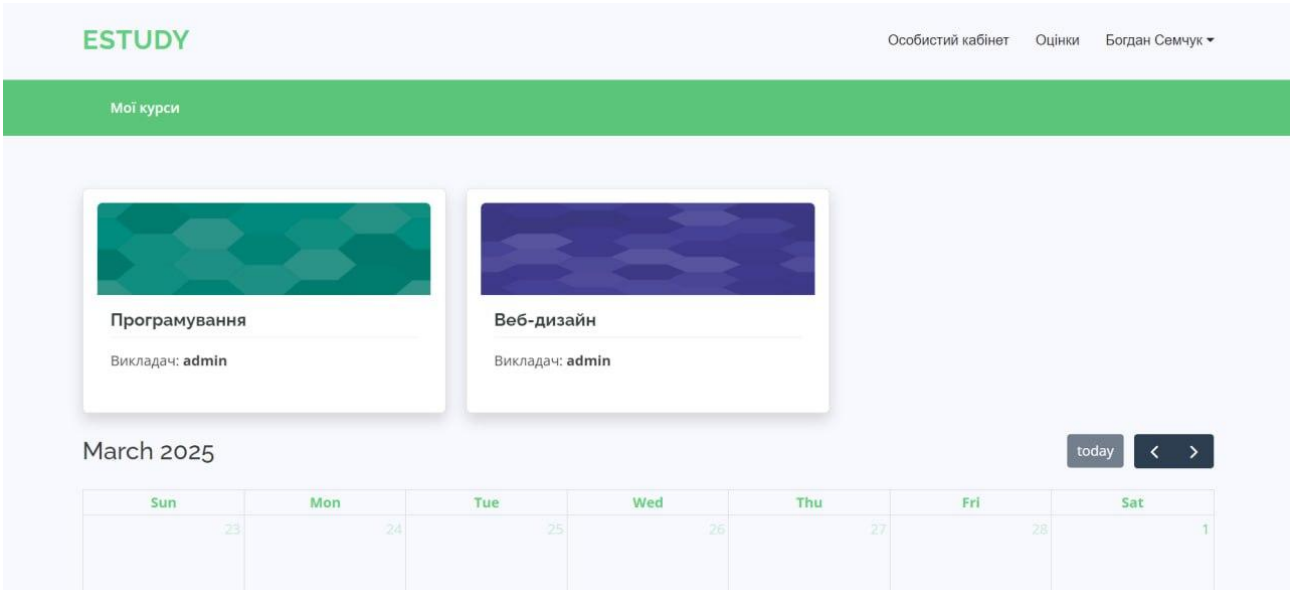

Рис. 14 Головна сторінка

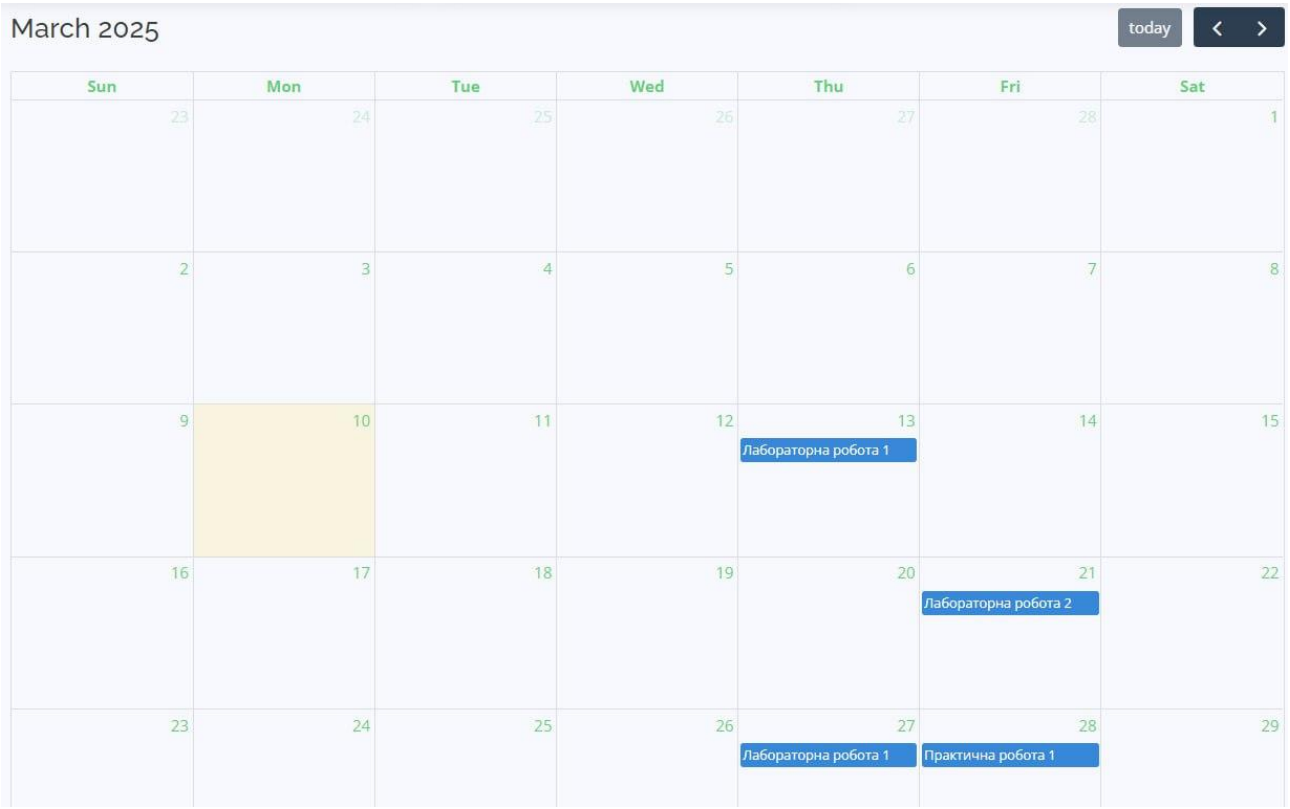

Рис. 15 Календар

На рис. 16 представлена форма авторизації в системі, яку необхідно заповнити при першому вході в систему.

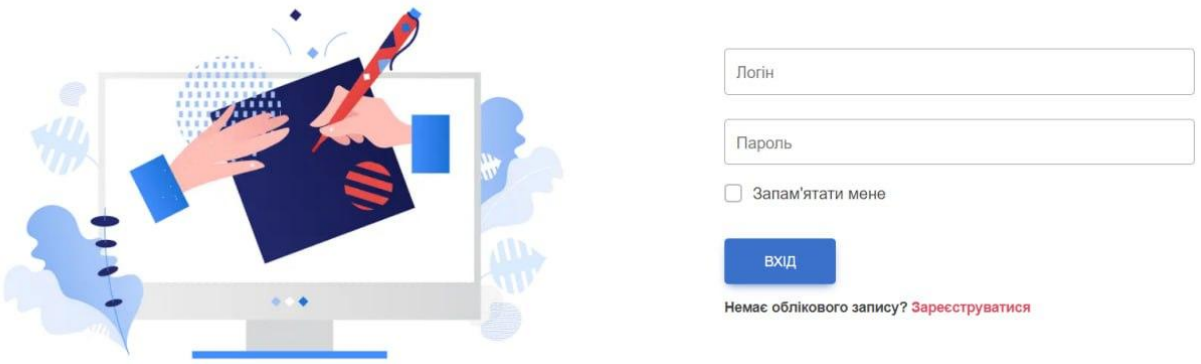

Рис. 16 Авторизація

Увійшовши до системи під користувачем з правами викладача, ми можемо створити дисципліну та у вибраній дисципліні створити завдання. Створення завдання подано на рис. 17

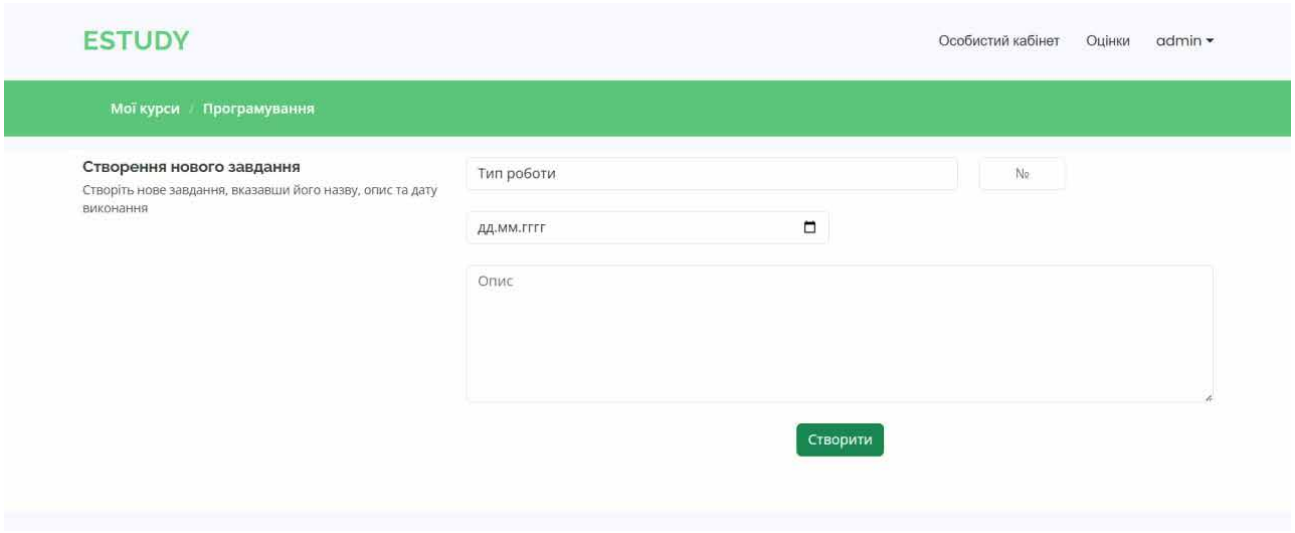

Рис. 17 Форма створення завдання

Ми вибираємо тип роботи (лабораторна робота, практична робота, самостійна робота, модульний тест, іспит), назва роботи, опис та дату до якої потрібно здати роботу.

Тепер ми можемо зайти в дисципліну та побачити перед собою її опис та список доданих робіт з їхніми дедлайнами. Сторінка з дисципліною представлена на рис. 18.

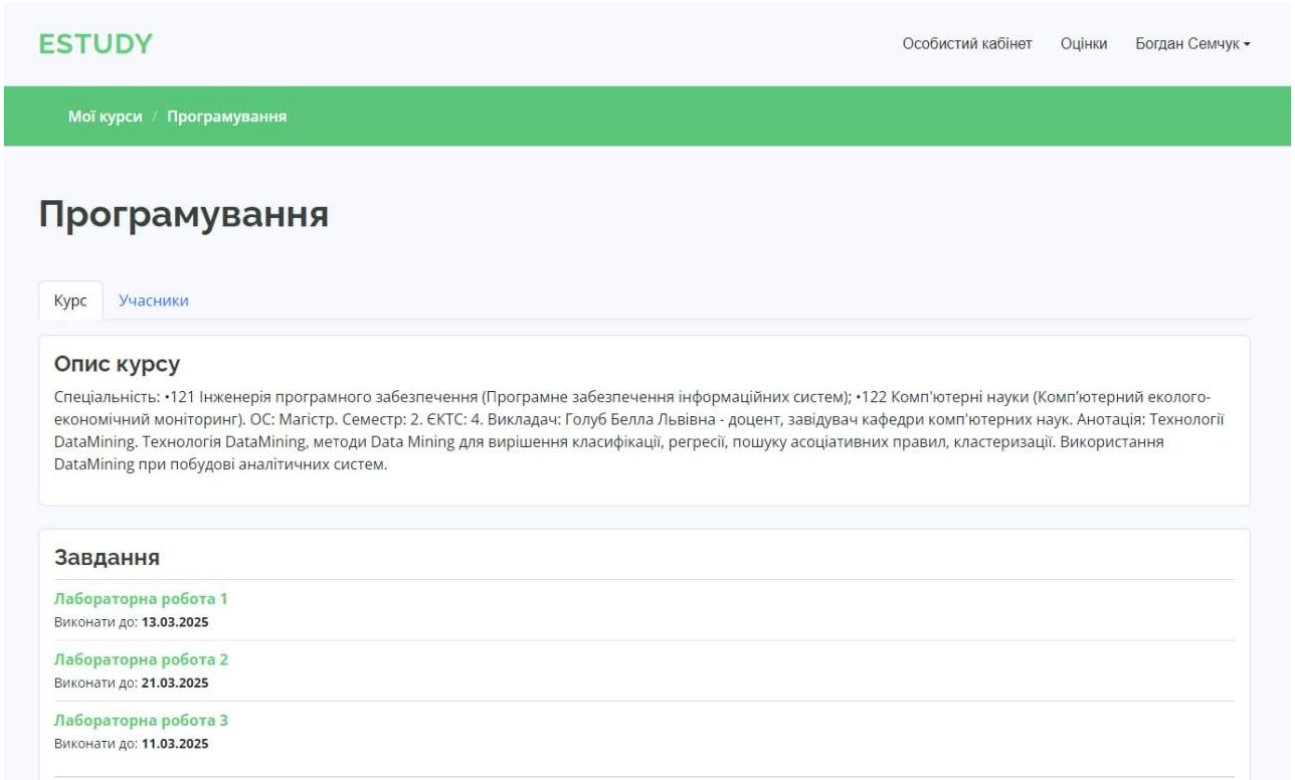

Рис. 18 Сторінка дисципліни

Вибираємо якесь завдання та переходимо на сторінку з цим завданням (рис. 19). Тут у нас показується також опис завдання, термін здачі роботи, статус роботи та кнопка завантаження файлу.

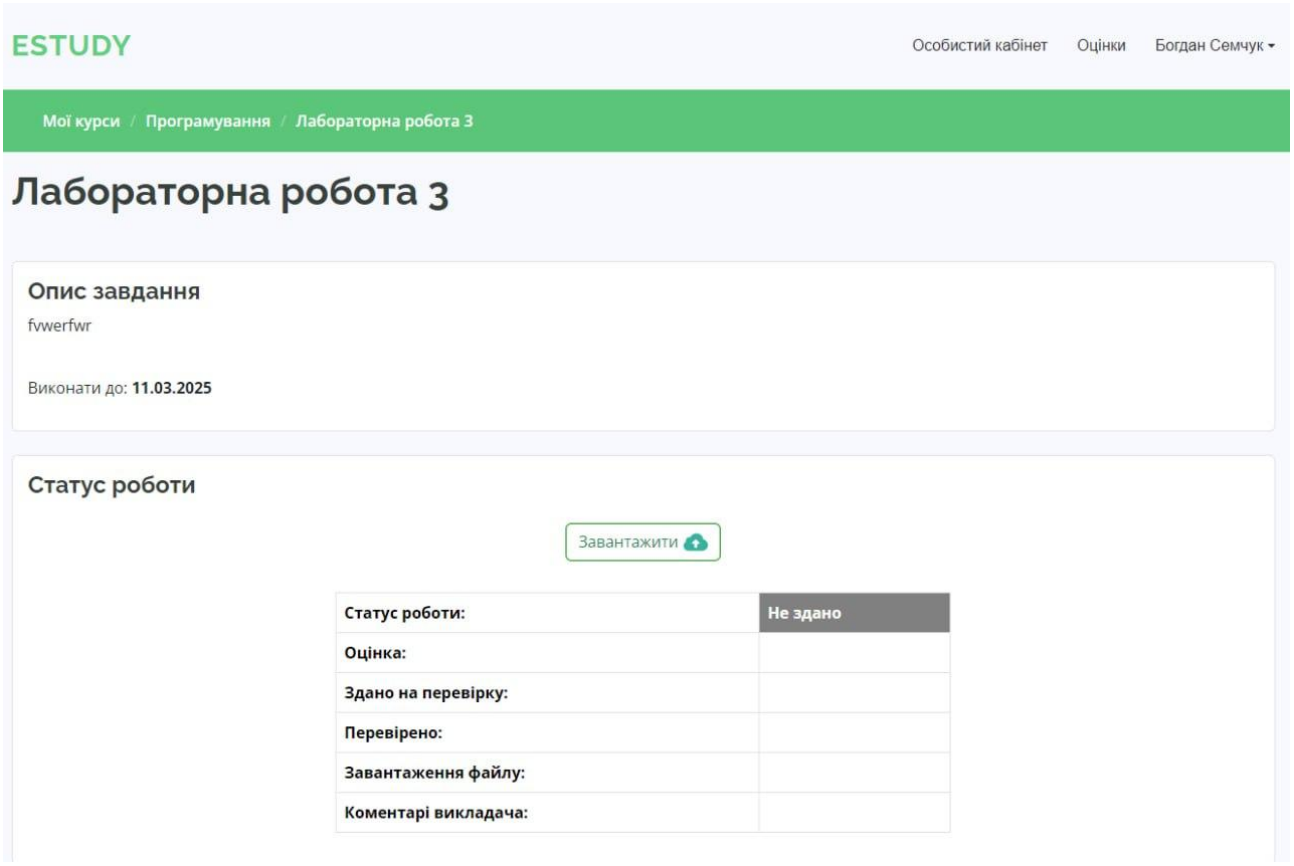

Рис. 19 Сторінка завдання

Після завантаження файлу та відправки роботи на перевірку, у нас змінюється статус роботи (рис. 20) і після цього у викладача з'являється ця робота у списку на перевірку (рис. 21).

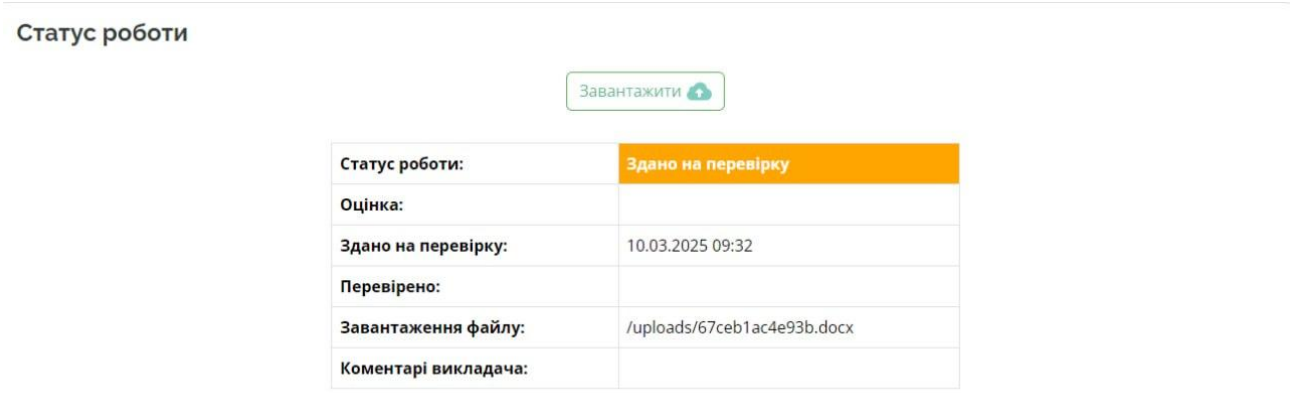

Рис. 20 Зміна статусу роботи

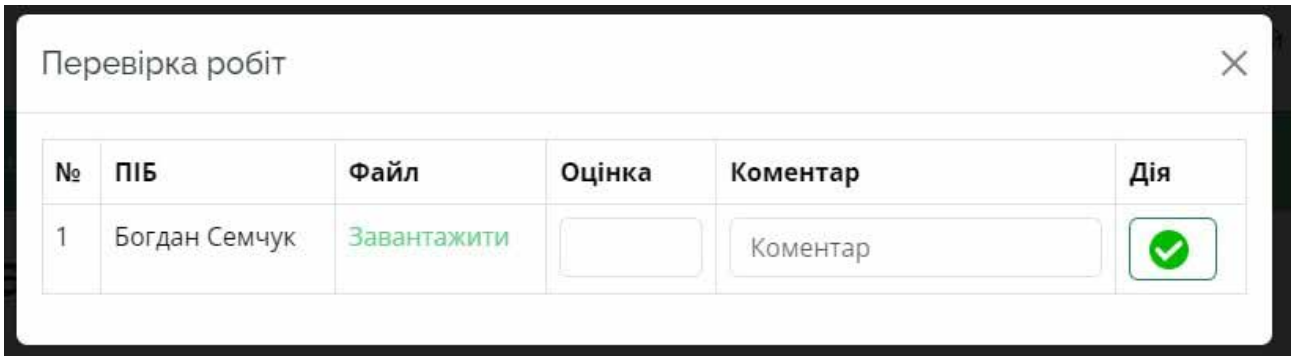

Рис. 21 Перевірка робіт у викладача

Далі ми можемо зайти на сторінку з оцінками (рис. 22) та переглянути загальні оцінки за кожну дисципліну, на яку ми були додані.

Рис. 22 Сторінка “Оцінки”

І вибравши певну дисципліну, ми переходимо на сторінку з оцінками за кожну роботу з цієї дисципліни. Тут у нас показується оцінка за завдання, коментар та внизу загальний бал за курс (рис. 23).

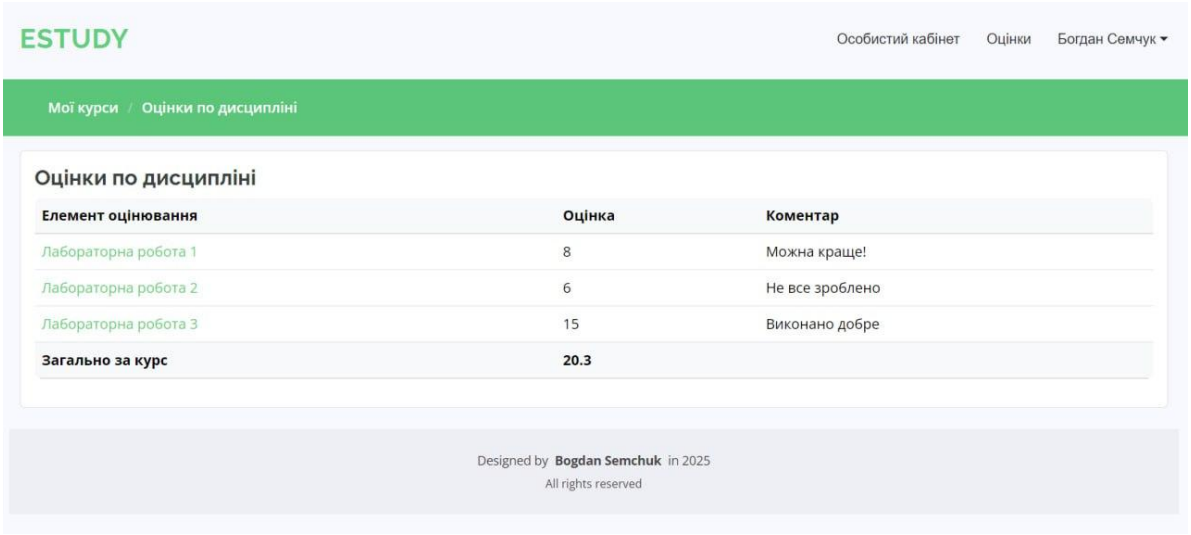

Рис. 23 Оцінки по обраній дисципліни

ВИСНОВКИ

Розробка інформаційної системи для викладачів навчальних закладів є значним кроком вперед у покращенні освітнього досвіду як для студентів, так і для викладачів. Використовуючи сучасні технології та методології, ця система спрямована на оптимізацію процесів управління навчальними курсами, призначення різних видів завдань та їх оцінювання, що сприяє спрощенню процесу навчання.

Створення власної інформаційно-управляючої системи розпочалося з аналізу предметної області та визначення її основної проблематики, що потребує вирішення. Дослідження існуючих рішень дозволило побачити недоліки схожих систем та уникнути їх при проектуванні власного програмного забезпечення, спрямованого на покращення сфери навчання. Це стало відправною точкою для подальшої розробки.

На основі отриманих вимог була створена логічна модель даних, яка відображає ключові інформаційні зв’язки та структуру програмного забезпечення системи. Вона слугувала базою для побудови архітектури нашої веб-орієнтованої системи. Подальший етап передбачав розробку фізичної моделі даних, що забезпечує оптимальне зберігання інформації та швидкий доступ до неї, використовуючи систему управління базами даних.

Під час виконання проєкту, особлива увага приділялася розробці та реалізації різних компонентів, включаючи інтерфейс користувача, роботу з базою даних і функціональність серверної частини. Вибір інструментів, таких як PHP, Symfony, MySQL, Doctrine ORM, HTML, CSS, JavaScript і Figma, відіграв важливу роль у створенні надійної та масштабованої програми, яка відповідає конкретним потребам викладачів.

Інтеграція клієнт-серверної архітектури гарантує, що система здатна обслуговувати декілька користувачів одночасно, зберігаючи цілісність і безпеку даних. Крім того, модульна конструкція програми дозволяє майбутні

вдосконалення та адаптації, гарантуючи, що вона може розвиватися разом із прогресуючою освітньою сферою.

Зосереджуючись на принципах дизайну, орієнтованого на користувача, система визначає пріоритети потреб як викладачів, так і студентів, надаючи інтуїтивно зрозумілий інтерфейс, який полегшує бездоганну взаємодію з платформою. Цей наголос на зручності використання має вирішальне значення для того, щоб система була легкозрозумілою для цільової аудиторії.

Проєкт не лише вирішує поточні виклики, з якими стикаються викладачі під час керування навчальним процесом, але й закладає основу для майбутніх розробок освітніх технологій. У міру розгортання та використання системи, постійний зворотний зв’язок буде необхідним для визначення областей, що потребують вдосконалення та подальших нововведень.

Зрештою, інформаційно-управляюча система діяльності викладача навчального закладу є цінним ресурсом, який покращує комунікацію, організацію та співпрацю в освітньому контексті. Ми сподіваємося, що ця система сприятиме більш захоплюючому та ефективнішому навчанню, надаючи можливість студентам і викладачам досягати своїх академічних цілей.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Documentation Bootstrap https://getbootstrap.com/documentation (дата звернення: 09.05.2025).
2. eLearn https://elearn.nubip.edu.ua (дата звернення: 09.04.2025).
3. Entity Relationship Diagram – Data Modeling https://www.visualparadigm.com/VPGallery/datamodeling/EntityRelationshipDiagram.html (дата звернення: 17.05.2025).
4. Logical Data Model. Dataedo https://dataedo.com/kb/data-glossary/logical-data-model (дата звернення: 25.04.2025).
5. Logical Data Model. Techopedia https://www.techopedia.com/definition/23969/logical-data-model (дата звернення: 23.04.2025).
6. Microsoft Docs. Layered architecture pattern https://docs.microsoft.com/en-us/azure/architecture/patterns/layered (дата звернення: 06.05.2025).
7. Moodle https://moodle.org (дата звернення: 11.04.2025).
8. Object Diagram in UML: Definition & Examples https://study.com/academy/lesson/object-diagram-in-uml-definition-examples.html (дата звернення: 21.04.2025).
9. UML - Activity Diagrams – Tutorialspoint https://www.tutorialspoint.com/uml/uml_activity_diagram.htm (дата звернення: 08.05.2025).
10. UML 2 Tutorial - Package Diagram - Sparx Systems https://sparxsystems.com/resources/tutorials/uml2/package-diagram.html (дата звернення: 10.05.2025).
11. W3Schools https://www.w3schools.com (дата звернення: 11.05.2025).

12. What is Component Diagram https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-component-diagram/ (дата звернення: 13.05.2025).

13. What is Sequence Diagram https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-sequence-diagram/ (дата звернення: 02.05.2025).

14. What is Unified Modeling Language (UML) https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/ (дата звернення: 21.05.2025).

15. Андерсон Т Орієнтований на користувача дизайн у розробці освітнього програмного забезпечення Міжнародний журнал взаємодії людини та комп'ютера – 2021 – Т 37 № 5 – С 550–564.

16. Браун Л К, Грін Т Д Роль технологій у сучасній освіті: тенденції та інновації Журнал розвитку та обміну освітніми технологіями – 2018 – Т 11 № 1 – С 15–29.

17. Гарсія Р Створення ефективного середовища онлайн-навчання: стратегії успіху Journal of Online Learning and Teaching – 2019 – Т 15 № 4 – С 12–22.

18. Джонсон М Розробка інтерфейсів користувача для некомерційних програм Journal of Nonprofit Technology – 2019 – Т 10 № 2 – С 45–62.

19. Діаграма прецедентів https://flexberry.github.io/ru/fd_use-case-diagram.html (дата звернення: 16.05.2025).

20. Льюїс К, Патель А Найкращі методи розробки онлайн-курсів: вичерпний посібник Журнал дистанційної освіти – 2020 – Т 22 № 2 – С 27–38.

21. Міллер С, Тернер П Електронне навчання та його вплив на залучення студентів Journal of Educational Research and Practice – 2021 – Т 8 № 2 – С 67–79.

22. Побудова діаграми класів https://flexberry.github.io/ru/gpg_class-diagram.html (дата звернення: 15.05.2025).

23. Проєктування ER-діаграми https://nationalteam.worldskills.ru/skills/proektirovanie-er-diagrammy/ (дата звернення: 19.05.2025).

24. Проєктування Use Case діаграми Визначення функціональних можливостей системи https://nationalteam.worldskills.ru/skills/proektirovanie-use-case-diagrammy-opredelenie-funktsionalnykh-vozmozhnostey-sistemy/ (дата звернення: 28.04.2025).

25. Робертс Е Мобільне навчання у вищій освіті: виклики та можливості Journal of Mobile Learning Technologies – 2018 – Т 5 № 3 – С 15–23.

26. Сміт Дж А Впровадження систем управління навчанням у вищій освіті: практичний приклад Міжнародний журнал освітніх технологій – 2020 – Т 15 № 3 – С 32–50.

27. Типи архітектури програмного забезпечення https://medium.com/nuances-of-programming/4-типа-архитектуры-программного-обеспечения-917133174724

28. Томпсон Х Майбутнє освітніх технологій: тенденції, на які варто звернути увагу Журнал освітніх технологій – 2020 – Т 20 № 1 – С 24–30.

29. UML-діаграми класів https://prog-cpp.ru/uml-classes/ (дата звернення: 15.05.2025).

ДОДАТОК А

Фрагменти програмного коду. Функція створення завдання

```
class TaskController extends AbstractController
{
    #[Route('/course/{id}/task/create', name: 'task_create')]
    public function createTask(
        int $id,
        Request $request,
        EntityManagerInterface $entityManager,
        CourseRepository $courseRepository,
        CategoryRepository $categoryRepository
    ) {
        $course = $courseRepository->find($id);
        $categories = $categoryRepository->findAll();

        if ($request->isMethod('POST')) {
            $title = $request->request->get('title');
            $description = $request->request->get('description');
            $categoryId = $request->request->get('category_id');
            $dueDateString = $request->request->get('due_date');

            if (!$course) {
                throw $this->createNotFoundException('Курс не знайдено');
            }

            $dueDate = new \DateTime($dueDateString);

            $task = new Task();
            $task->setCourse($course);
            $task->setTitle($title);
            $task->setDescription($description);
            $task->setCategory($categoryRepository->find($categoryId));
            $task->setDueDate($dueDate);

            $entityManager->persist($task);
            $entityManager->flush();

            return $this->redirectToRoute('course_show', ['id' => $id]);
        }

        return $this->render('task/create.html.twig', [
            'course' => $course,
            'categories' => $categories
        ]);
    }

    #[Route('/task/{id}', name: 'task_show')]
```

```
    public function showTask(int $id, EntityManagerInterface $entityManager,
Security $security): Response
    {
        $task = $entityManager->getRepository(Task::class)->find($id);

        if (!$task) {
            throw $this->createNotFoundException('Завдання не знайдено');
        }

        $user = $security->getUser();
        if (!$user) {
            throw $this->createAccessDeniedException('Ви не авторизовані');
        }

        $submission = $entityManager->getRepository(Submission::class)-
>findOneBy(['task' => $task, 'student' => $user]);
        $grade = $submission ? $entityManager->getRepository(Grade::class)-
>findOneBy(['submission' => $submission]) : null;

        $submissionsToCheck = $entityManager-
>getRepository(Submission::class)->findBy(['task' => $task, 'status' => 1]);

        return $this->render('task/show.html.twig', [
            'task' => $task,
            'submission' => $submission,
            'grade' => $grade,
            'course' => $task->getCourse(),
            'submissionsToCheck' => $submissionsToCheck,
        ]);
    }
}
```

ДОДАТОК Б

Фрагменти програмного коду. Функція оцінювання завдання

```
#[Route('/submission/{id}/grade', name: 'submission_grade', methods: ['POST'])]
    public function gradeSubmission(
        int $id,
        Request $request,
        EntityManagerInterface $entityManager
    ): JsonResponse {
        $submission = $entityManager->getRepository(Submission::class)->find($id);

        if (!$submission) {
            return new JsonResponse(['success' => false, 'message' => 'Робота не
знайдена'], 404);
        }

        $data = json_decode($request->getContent(), true);
        $score = $data['score'] ?? null;
        $comment = $data['comment'] ?? null;

        if ($score === null || $score < 0 || $score > 100) {
            return new JsonResponse(['success' => false, 'message' => 'Некоректна
оцінка'], 400);
        }

        $grade = new Grade();
        $grade->setSubmission($submission);
        $grade->setComment($comment);
        $grade->setScore($score);
        $grade->setRatedDate(new \DateTime());

        $submission->setStatus(2);

        $entityManager->persist($grade);
        $entityManager->flush();

        return new JsonResponse(['success' => true]);
    }
```

ДОДАТОК В

Фрагменти програмного коду. Розрахунок оцінок студентів

```
class GradeController extends AbstractController

{

    #[Route('/grades', name: 'grades')]

    public function index(EntityManagerInterface $entityManager, Security
$security, GradeService $gradeService): Response

    {

        $user = $security->getUser();

        if (!$user) {

            return $this->redirectToRoute('login');

        }


        $courseStudents = $entityManager->getRepository(CourseStudents::class)-
>findBy(['student' => $user]);

        $submissionRepository = $entityManager->getRepository(Submission::class);


        $grades = [];


        foreach ($courseStudents as $courseStudent) {

            $course = $courseStudent->getCourse();

            $submissions = $submissionRepository-
>findSubmissionsByStudentAndCourse($user, $course);


            $finalGrade = $gradeService->calculateFinalGrade($submissions);


            $grades[] = [

                'course' => $course,

                'finalGrade' => $finalGrade,

            ];
```

```
        }

        return $this->render('grades/show.html.twig', [
            'grades' => $grades,
        ]);
    }

    #[Route('/grades/course/{courseId}', name: 'grades_course')]
    public function courseGrades(EntityManagerInterface $entityManager, Security
$security, GradeService $gradeService, int $courseId): Response
    {
        $user = $security->getUser();
        if (!$user) {
            return $this->redirectToRoute('login');
        }

        $submissionRepository = $entityManager->getRepository(Submission::class);
        $submissions = $submissionRepository-
>findSubmissionsByStudentAndCourse($user, $courseId);

        $grades = [];

        foreach ($submissions as $submission) {
            $grades[] = [
                'task' => $submission->getTask(),
                'grade' => $submission->getGrade(),
            ];
        }
        $finalGrade = $gradeService->calculateFinalGrade($submissions);
        return $this->render('grades/course.html.twig', [
            'grades' => $grades,
```

```
            'finalGrade' => $finalGrade,
        ]);
    }
class GradeService
{
    /**
     * Розрахунок загального балу студента за дисципліну
     *
     * @param Submission[] $submissions
     * @return float
     */
    public function calculateFinalGrade(array $submissions): float
    {
        $totalScore = 0;
        foreach ($submissions as $submission) {
            $grade = $submission->getGrade();
            if ($grade) {
                $category = $submission->getTask()->getCategory();
                $score = $grade->getScore();
                if (in_array($category->getId(), [1, 2, 3, 4])) {
                    $totalScore += $score * 0.7;
                } elseif ($category->getId() === 5) {
                    $totalScore += $score;
                }
            }
        }
        return $totalScore;
    }
}
```