НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ПОГОДЖЕНО	**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ**
Декан факультету	**Завідувач кафедри комп'ютерних наук**
_____Ігор БОЛБОТ	**_____Белла ГОЛУБ**
(підпис)	(підпис)
«___» _____2026 р.	**«___» _____2026 р.**

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Програмне забезпечення системи керування процесом розповсюдження інформації серед учасників навчального процесу»

Спеціальність «121 Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення»

Гарант освітньої програми к.т.н., доцент	_____ (підпис)	**Ганна ВАЙГАНГ**
Керівник бакалаврської кваліфікаційної роботи к.ф.-м.н., доцент	_____ (підпис)	**Віктор КИРИЧЕНКО**
Виконала	_____ (підпис)	**Вероніка КОНДРАТЕНКО**

Київ-2026

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук
к.т.н., доцент _____Белла ГОЛУБ
(підпис)
«___» _____2025 р.

ЗАВДАННЯ

ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ ЗДОБУВАЧУ

КОНДРАТЕНКО ВЕРОНІКА ДМИТРІВНА

Спеціальність «121 Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення»
Тема бакалаврської кваліфікаційної роботи
«Програмне забезпечення системи керування процесом розповсюдження інформації серед учасників навчального процесу» затверджена наказом від «19» грудня 2025 р. №3204 «С»
Термін подання завершеної роботи на кафедру «22» травня 2026 р.

Вихідні дані до бакалаврської кваліфікаційної роботи: предметна область інформаційно-комунікаційної взаємодії між учасниками навчального процесу; вимоги до навчальної вебплатформи; рольова модель доступу; структура БД для студентів, викладачів, груп, дисциплін, розкладу, завдань, відвідуваності та повідомлень; стек Python, FastAPI, PostgreSQL, HTML, CSS, JavaScript.

Перелік питань, що підлягають дослідженню:

1. Аналіз предметної області та існуючих освітніх вебплатформ;
2. Формування вимог до програмного забезпечення та постановка завдання;
3. Проєктування логічної моделі даних, UML-моделей і архітектури системи;
4. Розробка інформаційної бази, серверної та клієнтської частин вебплатформи;
5. Інтеграція комунікаційного модуля, чат-бота та AI-маршрутизації звернень;
6. Тестування системи та підготовка рекомендацій щодо впровадження.

Перелік графічних документів (за необхідності). ER-діаграма бази даних; UML-діаграми прецедентів, активності, послідовності, класів, пакетів, компонентів і розміщення; скріншоти інтерфейсу системи.
Дата видачі завдання «19» грудня 2025 р.

Керівник бакалаврської кваліфікаційної роботи _____ (підпис) **Віктор КИРИЧЕНКО**

Завдання взяла до виконання _____ (підпис) **Вероніка КОНДРАТЕНКО**

Зміст

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

3NF (Third Normal Form) - Третя нормальна форма реляційної бази даних.

ACID (Atomicity, Consistency, Isolation, Durability) - Набір вимог до транзакційної системи баз даних.

AI (Artificial Intelligence) - Штучний інтелект.

API (Application Programming Interface) - Інтерфейс прикладного програмування.

ASGI (Asynchronous Server Gateway Interface) - Асинхронний інтерфейс шлюзу вебсе+рвера.

CORS (Cross-Origin Resource Sharing) - Механізм спільного використання ресурсів з різних джерел.

CPU / vCPU (Central Processing Unit) - Центральний процесор / Віртуальний центральний процесор.

CSS3 (Cascading Style Sheets) - Каскадні таблиці стилів 3-ї версії.

DDL (Data Definition Language) - Мова опису даних.

DOM (Document Object Model) - Об'єктна модель документа.

DTO (Data Transfer Object) - Об'єкт передачі даних.

ER-діаграма (Entity-Relationship) - Діаграма «сутність-зв'язок» для моделювання баз даних.

HTML5 (HyperText Markup Language) - Мова розмітки гіпертексту 5-ї версії.

HTTP / HTTPS (HyperText Transfer Protocol) - Протокол передачі гіпертексту / Безпечний протокол передачі гіпертексту.

I/O (Input/Output) - Операції вводу-виводу.

IP (Internet Protocol) - Інтернет-протокол.

JS (JavaScript) - Мова програмування JavaScript.

JSON (JavaScript Object Notation) - Текстовий формат обміну даними.

JWT (JSON Web Token) - Вебтокен формату JSON для безпечної передачі

інформації.

LMS (Learning Management System) - Система управління навчанням.

MVP (Minimum Viable Product) - Мінімально життєздатний продукт.

NVMe (Non-Volatile Memory express) - Протокол доступу до твердотільних накопичувачів.

ORM (Object-Relational Mapping) - Об'єктно-реляційне відображення для зв'язку баз даних з об'єктно-орієнтованим кодом.

RBAC (Role-Based Access Control) - Модель управління доступом на основі ролей.

REST (Representational State Transfer) - Архітектурний стиль взаємодії компонентів розподіленого вебзастосунку.

ROM (Read-Only Memory) - Постійний запам'ятовуючий пристрій.

SPA (Single Page Application) - Односторінковий вебзастосунок.

SQL (Structured Query Language) - Структурована мова запитів.

SSD (Solid-State Drive) - Твердотільний накопичувач.

UML (Unified Modeling Language) - Уніфікована мова моделювання.

WSGI (Web Server Gateway Interface) - Синхронний інтерфейс шлюзу вебсервера.

Guard Clauses (Багаторівнева фільтрація) - патерн проєктування, що передбачає ранній вихід із функції при невідповідності вхідних даних.

Long Polling - технологія постійного опитування сервера клієнтом для отримання нових даних.

Stateless-архітектура - архітектурна парадигма, за якої сервер не зберігає стан сесії клієнта.

Thin Client (Тонкий клієнт) - архітектурний підхід, за якого клієнтський рівень виконує мінімум обчислень.

Vendor lock-in (Технологічна залежність) - прив'язка архітектури проєкту до екосистеми конкретного постачальника.

Webhook - механізм розширення поведінки вебзастосунку за допомогою HTTP-викликів.

БД - База даних.

ВПО - Внутрішньо переміщені особи.

ОС - Освітній ступінь.

ПЗ - Програмне забезпечення.

СУБД - Система управління базами даних.

ШІ - Штучний інтелект.

Ескалація - алгоритмічний процес автоматичного перенаправлення нестандартного запиту від ШІ-маршрутизатора до викладача.

Модульний моноліт - архітектурний патерн з єдиною базою коду, що розділена на логічно ізольовані модулі.

Промпт-інжиніринг - набір методів конструювання інструкцій для великих мовних моделей.

Токен - криптографічний маркер для ідентифікації та авторизації клієнта.

ВСТУП

Сучасна вища школа функціонує в умовах форсованої цифрової трансформації, де інформаційно-комунікаційні технології остаточно втратили статус факультативних інновацій. Вони перетворилися на безальтернативний інструмент забезпечення життєздатності освітнього процесу. Проте швидка міграція в онлайн оголила критичні архітектурні недоліки існуючої інфраструктури.

Більшість академічних інституцій продовжують експлуатувати традиційні монолітні системи управління навчанням (LMS) на кшталт Moodle. Незважаючи на їхню масштабність, ці платформи проєктувалися в епоху домінування синхронної взаємодії. Сьогодні такий підхід провокує технологічний спротив: перевантажені інтерфейси та глибока вкладеність меню генерують надлишкове когнітивне навантаження на всіх учасників процесу. Найбільш вразливим вузлом виявилася комунікація. Дистанційний формат вимагає оперативного зворотного зв'язку, тоді як класичні LMS пропонують інертні форуми або email-листування. Це призводить до затримок в отриманні відповідей на рутинні питання та створює безперервний інформаційний шум для викладача, який змушений працювати в режимі диспетчера.

Вирішення цієї проблеми лежить у площині відмови від монолітних гігантів на користь легковагових, спеціалізованих платформ. Делегування алгоритмам штучного інтелекту (великим мовним моделям) функції первинного маршрутизатора здатне автоматизувати закриття типових інформаційних потреб студентів. Викладач залучатиметься виключно для вирішення нетипових інцидентів шляхом механізму ескалації. Паралельне винесення частини операційної рутини (фіксація присутності, розклад) у звичне середовище месенджерів радикально знижує поріг входження. Відповідно, проєктування модульної інформаційної системи, що поєднує

строгу рольову безпеку вебзастосунку з гнучкістю AI-маршрутизації, є об'єктивно необхідним та своєчасним науково-практичним завданням.

Мета і завдання дослідження. Метою бакалаврської кваліфікаційної роботи є проєктування та програмна реалізація комплексної навчальної вебплатформи з інтегрованим інтелектуальним комунікаційним ядром для оптимізації інформаційних потоків між суб'єктами освітнього процесу.

Відповідно до поставленої мети сформульовано наступні завдання:

1. Здійснити системний аналіз проблемної області, виявити архітектурні прогалини існуючих LMS та формалізувати бізнес-процеси за моделями «Як є» (As-Is) та «Як має бути» (To-Be).

2. Спроєктувати логічну та фізичну структуру реляційної бази даних (3NF), здатну транзакційно обслуговувати контингент, розклад, процес оцінювання та історію ескалації повідомлень.

3. Обґрунтувати та імплементувати серверну архітектуру за патерном модульного моноліту, відокремивши бізнес-логіку від зовнішніх API-інтерфейсів за допомогою шаблонів Repository та Strategy.

4. Розробити програмне ядро (Backend) засобами асинхронного фреймворку, гарантувавши безпеку через Stateless-автентифікацію (JWT) та сувору рольову ізоляцію доступу (RBAC).

5. Спроєктувати комунікаційний пайплайн з підключенням API великої мовної моделі для семантичного аналізу тексту та інтеграцією месенджер-бота для автоматизації телеметрії відвідуваності.

6. Реалізувати адаптивний клієнтський рівень (Frontend) за принципами Thin Client для мінімізації апаратних вимог до пристроїв кінцевих користувачів.

7. Виконати верифікацію програмного продукту за допомогою функціонального тестування та сформувати регламент розгортання системи.

Об'єкт дослідження - процес інформаційно-комунікаційної взаємодії між учасниками дистанційного та змішаного освітнього процесу.

Предмет дослідження - методи, архітектурні патерни та інструментальні

засоби розробки навчальної програмної платформи з інтеграцією алгоритмів штучного інтелекту.

Методи та технології дослідження. Дослідження спирається на комплекс методів програмної інженерії. Для формалізації архітектури застосовано методи системного аналізу, теорії баз даних та об'єктно-орієнтованого проєктування з візуалізацією в нотації UML. Конфігурування комунікаційного ядра здійснювалося методами промпт-інжинірингу (Prompt Engineering). Валідація результатів проводилася шляхом функціонального тестування за методом «чорного ящика». Технологічний стек включає мову Python (3.10+), мікрофреймворк FastAPI, СУБД PostgreSQL та нативні вебтехнології (HTML5, Vanilla JS).

Апробація результатів. Ключові аспекти проєктування навчальної вебплатформи з інтегрованим інтелектуальним комунікаційним ядром були опубліковані на Науково-практичній конференції студентів і аспірантів Торетичні та прикладні аспекти розробки комп’ютерних систем 2026 «Проєктування навчальної вебплатформи з інтегрованим інтелектуальним комунікаційним ядром» (м.Київ, 2026р.)

Структура та обсяг пояснювальної записки. Пояснювальна записка до бакалаврської кваліфікаційної роботи складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатків. Загальний обсяг роботи становить 82 сторінки машинописного тексту. Робота ілюстрована 24 рисунками, містить 14 таблиць, список використаних джерел із 44 найменувань та 4 додатки.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВИБІР ТЕХНОЛОГІЇ

1.1 Опис предметної області та базових ролей користувачів

Глобальна освітня система переживає етап вимушеної структурної перебудови. Перехід до цифрових парадигм зумовлений швидше об’єктивними викликами зовнішнього середовища, аніж еволюційним технологічним прогресом. В українських реаліях цифровізація вищої школи давно перейшла з категорії факультативних інновацій у площину критичного виживання академічних інституцій [1]. Дистанційний та змішаний формати закріпилися як домінантні моделі взаємодії. Отримані аналітичні дані дозволяють обережно судити про розширення віртуальної мобільності: здобувачі отримують доступ до ресурсів без жорсткої прив'язки до географічного розташування [1, 2]. Згадана тенденція диктує вимогу щодо розгортання надійних централізованих платформ для структурування навчального контенту.

Впровадження інформаційно-комунікаційних технологій генерує передумови для індивідуалізації освітніх траєкторій. Проста заміна паперових носіїв їхніми цифровими аналогами в даному контексті демонструє нульову ефективність. Вимагається радикальний перегляд традиційних методик викладання та систем оцінювання. Практика показує, що взаємодія з сучасними платформами стимулює розвиток цифрових компетентностей у всіх учасників процесу. Це стає фактором професійної виживаності в інформаційному суспільстві [2]. З іншого боку, форсована цифровізація несе цілком реальні деструктивні наслідки. Фіксується стійке психологічне вигорання від надлишку віртуальної комунікації, яке додатково поглиблюється періодичними апаратними збоями та нерівномірним покриттям мережі [1, 3].

Зазначені протиріччя формують гостру потребу в проєктуванні оптимізованих програмних рішень, здатних мінімізувати когнітивне навантаження на користувача.

Таблиця 1.1

Фактори впливу цифровізації на сучасний освітній простір

Фактори впливу цифровізації на освітній простір	**Позитивні наслідки**	**Виклики та ризики**
Географічна незалежність	Доступність освіти для мешканців віддалених регіонів та ВПО	Проблеми стабільності зв’язку та електропостачання
Гнучкість графіку	Можливість поєднання навчання з іншими видами діяльності	Необхідність високого рівня самодисципліни та тайм-менеджменту
Централізація контенту	Створення єдиного репозиторію навчальних матеріалів	Ризик інформаційного перевантаження та втрати фокуса
Автоматизація процесів	Прискорення перевірки знань та збору статистики	Зниження рівня безпосереднього міжособистісного контакту

Існують підстави вважати, що дистанційний формат нівелює частину соціальних бар'єрів на старті, вирівнюючи можливості здобувачів [2]. Проте механічне перенесення лекційної рутини у формат відеоконференцій має вкрай низьку ефективність. Критичної ваги набуває архітектура спеціалізованого програмного забезпечення. Саме вона регламентує правила та межі взаємодії між суб’єктами [1, 4].

1.2 Аналіз вимог до навчальної веб-платформи

Архітектурним ядром сучасного університету виступає система управління навчанням (LMS). Її вплив виходить за межі суто технічної

трансляції контенту, глибоко зачіпаючи соціально-педагогічні аспекти. Повноцінна навчальна вебплатформа мусить підтримувати цілісний життєвий цикл освітнього процесу: управління структурою дисциплін, публікацію актуального розкладу, облік успішності та контроль відвідуваності [4]. У межах платформи формується жорстко структурований простір, де навчальні матеріали, інструменти моніторингу та канали зворотного зв'язку зведені в єдиний інтерфейс. Відбувається зсув від дискретної взаємодії до безперервного циклу. Студент працює з даними та отримує консультації переважно в асинхронному режимі.

Експертні оцінки викладацького складу вказують на зміну парадигми викладання: лектор трансформується у фасилітатора [4]. Серед очевидних переваг фіксують прозорий моніторинг успішності та автоматизацію перевірки типових завдань. Водночас платформи часто критикують за комунікаційну "стерильність". Системі об'єктивно бракує емоційної залученості та миттєвої реакції, притаманної живій аудиторії [3, 5].

Користувацька задоволеність прямо пропорційна стабільності системи. Екстраполюючи модель DeLone та McLean на освітній контекст, визначальними метриками успіху стають якість архітектури та валідність даних. Затримки відгуку інтерфейсу або заплутана навігація неминуче провокують технологічний спротив [5]. Дослідження підтверджують, що надлишкова вкладеність меню різко знижує інтенсивність роботи викладачів [5].

Фундаментом безпеки освітньої інфраструктури залишається суворе та послідовне розмежування привілеїв користувачів. Оптимальним рішенням для складних академічних ієрархій вбачається застосування рольової моделі доступу (RBAC), яка забезпечує контрольоване надання прав відповідно до функцій учасників системи. Дозволи делегуються не окремим користувачам, а абстрактним ролям, прив’язаним до посади, статусу або рівня відповідальності [6, 7].

Таблиця 1.2

Розподіл прав доступу в системі за базовими ролями

Роль у системі	**Об’єкти доступу**	**Дозволені дії**
Адміністратор	Конфігурація платформи, профілі	Повне управління, налаштування безпеки
Викладач	Навчальні матеріали, журнали	Створення, редагування, перевірка
Студент	Ресурси курсу, власні оцінки	Перегляд, завантаження відповідей [7]

Такий підхід мінімізує інциденти безпеки завдяки принципу найменших привілеїв [6]. Також архітектура повинна підтримувати динамічне групування користувачів для перерозподілу доступу до курсів [7].

З огляду на глобальні тенденції, проєктування зобов'язане спиратися на принципи адаптивного дизайну. Безперебійний доступ з мобільних пристроїв перейшов у статус базового стандарту [9]. Сучасні рішення відмовляються від громіздких монолітних інтерфейсів на користь мінімалізму навігації.

Разом з тим, традиційні канали сповіщення (наприклад, email) демонструють системну неефективність. Альтернативою виступають месенджер-боти. Високий ступінь їхньої інтеграції в щоденний побут гарантує максимальний відсоток прочитання [10]. Бот перебирає на себе операційну рутину: від розсилки розкладу до агрегації даних про присутність на заняттях [10, 11].

Важливим архітектурним рішенням при проєктуванні такої взаємодії виступає вибір патерну між сервером та API месенджера. Механізм Long Polling (постійне опитування сервера) створює надлишкове навантаження на апаратні потужності, що становить критичний недолік для систем з великим контингентом. Натомість архітектура Webhook використовує концепцію зворотного виклику. Сервер освітньої платформи реєструє унікальний ендпоінт. Коли студент відправляє повідомлення, сервери месенджера самостійно ініціюють POST-запит. Цей подієво-орієнтований підхід мінімізує затримки та раціонально використовує ресурси бекенду.

1.3 Моделювання предметної області

Першим кроком на шляху від неструктурованих бізнес-процесів до реального програмного коду є системна декомпозиція предметної області. Аналіз організаційної структури класичного університету дозволив сформувати концептуальну модель платформи, виділивши її базові абстракції. Кожна сутність у цій моделі детермінується через два ключові аспекти: набір структурних атрибутів (дані, які вона зберігає) та регламентовану поведінку.

Цей архітектурний підхід дав змогу логічно ізолювати профілі фізичних акторів (адміністратора, викладача, студента) від суто інформаційних об'єктів навчального процесу (груп, дисциплін, завдань) та комунікаційних модулів. Суворе розмежування зон відповідальності кожної сутності виступає обов'язковою інженерною передумовою для проєктування нормалізованої реляційної бази даних. Крім того, це створює надійний базис для налаштування рольової моделі доступу та превентивно захищає архітектуру від виникнення антипатерну дублювання бізнес-логіки на етапі розробки бекенду.

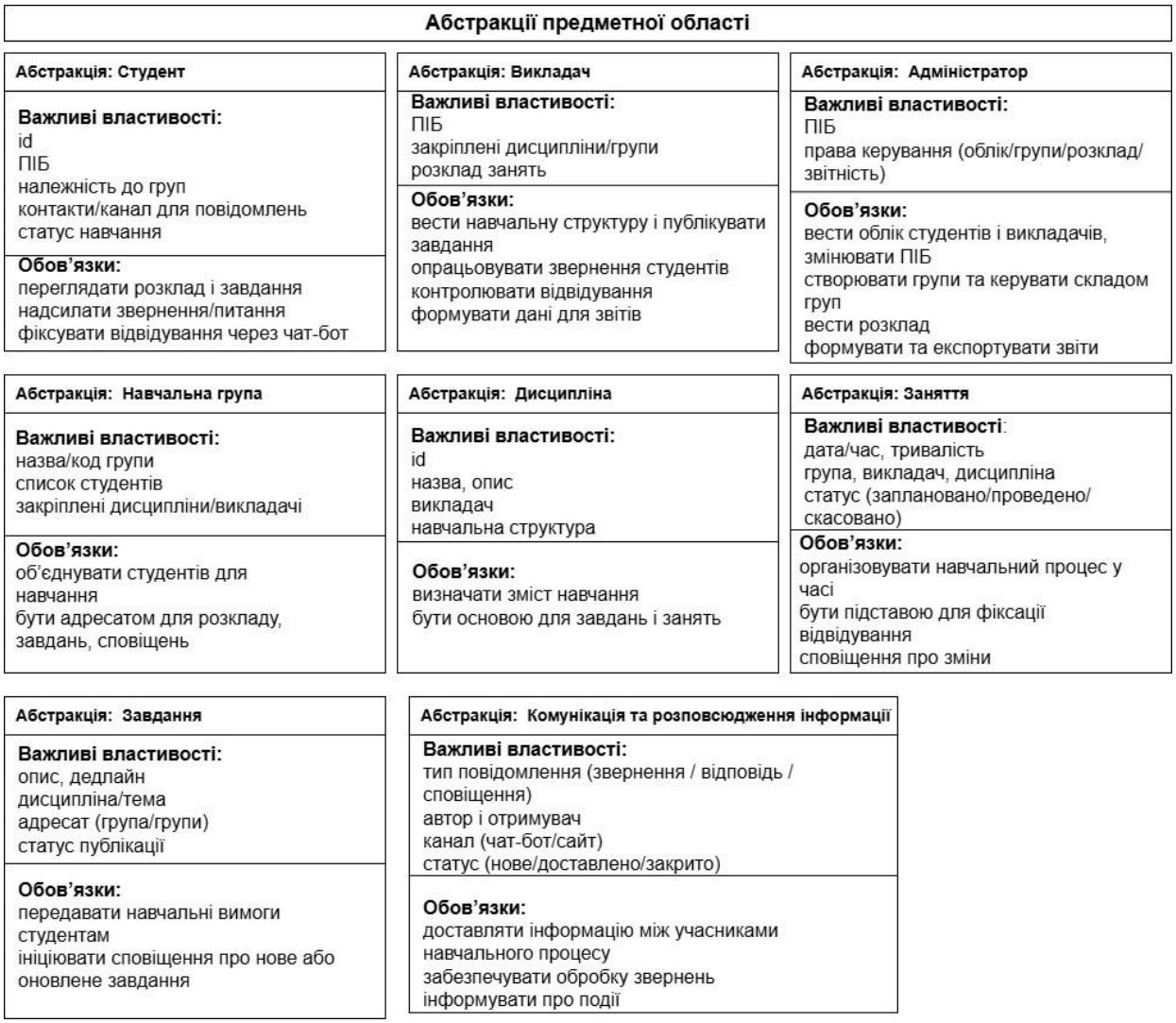


Рис. 1.1 – Абстракції предметної області та їхні функціональні обов'язки

Точність відображення реальних організаційних процесів у коді прямо визначає життєздатність кінцевого продукту. Візуалізація інформаційних потоків за допомогою стандартних нотацій моделювання дає змогу виявити логічні конфлікти на ранніх етапах [12, 13]. Аналіз предметної області охоплює моделювання кількох критичних контурів: управління контингентом, організацію розкладу, контроль успішності та комунікацію.

Алгоритмізація таких задач є базовою вимогою для уникнення колізій та дублювання функцій [12]. Утім, системний аналіз вказує на те, що найбільш вузьким місцем, яке генерує критичні затримки в дистанційному навчанні, залишається процес комунікації між студентом та викладачем.

Для наочної ілюстрації проблеми інформаційного перевантаження змодельовано процес обробки звернень. На рисунку 1.2 зображено модель бізнес-процесу «Як є» (As-Is). Відсутність проміжного фільтра призводить до того, що викладач стає єдиним вузлом обробки всього масиву повідомлень. Це неминуче викликає часові затримки та втрату даних.

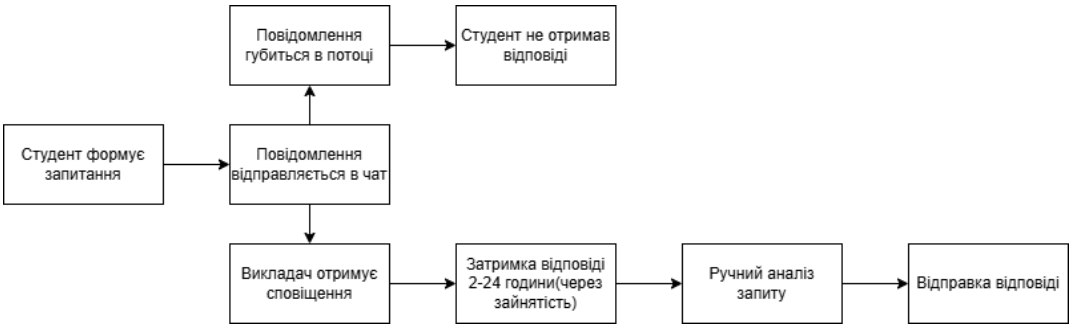


Рис. 1.2 – Модель обробки звернень «Як є»

Моделювання цільового стану «Як має бути» (To-Be) передбачає реінжиніринг процесу за рахунок інтеграції AI-модуля та месенджер-бота (рис. 1.3). У цій моделі нейромережа виконує функцію первинного диспетчера. Викладач залучається виключно для вирішення нетипових академічних питань.

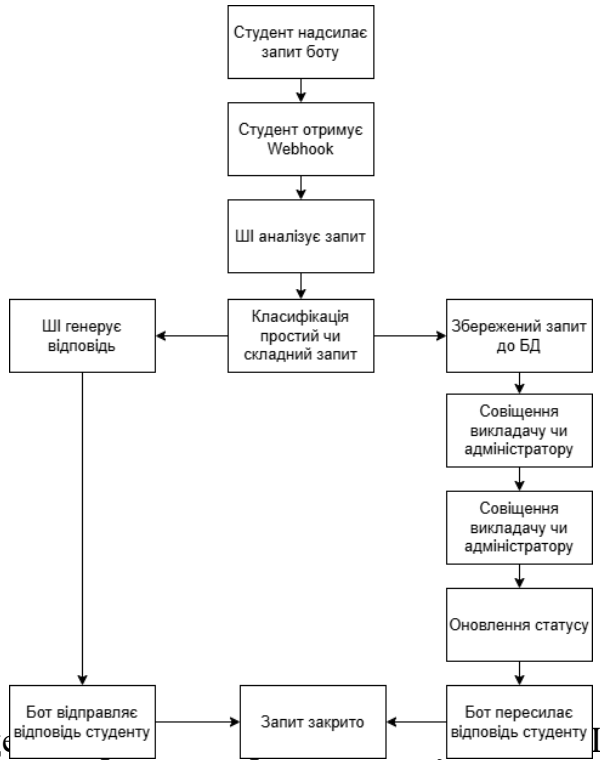


Рис. 1.3 – Цільова моде ШІ «Як має бути»

Мета цільового моделювання - спроєктувати архітектуру таким чином, щоб інформація між модулями мігрувала автоматично, ініціюючи відповідні тригери без втручання оператора [12, 13, 14].

Інтеграція алгоритмів штучного інтелекту розглядається передусім як інструмент зниження операційного навантаження [15, 16]. Замість ручної обробки сотень однотипних запитів щодо розкладу, інтелектуальний модуль бере на себе первинну маршрутизацію. Асистент аналізує семантику: якщо патерн відомий (організаційне питання) - генерується відповідь з бази знань, інакше запит ескалюється до робочого кабінету викладача [11, 16].

Впровадження машинного навчання для пріоритезації вхідного трафіку мінімізує "пошуковий шум". Викладач отримує змогу реагувати на тригери предметно, зберігаючи когнітивний ресурс для педагогічної діяльності [11, 17].

1.4 Огляд інформаційних джерел та існуючих рішень

Вибір архітектурного патерну виступає визначальним етапом для життєвого циклу проєкту. Класичні системи, маючи монолітну кодову базу, критично ускладнюють локальні оновлення та горизонтальне масштабування [18, 19].

Альтернативний мікросервісний підхід ізолює функціональні блоки один від одного. Для платформи, що передбачає підключення зовнішніх месенджер-ботів та AI-модулів, цей шлях виглядає найбільш технічно обґрунтованим [18, 19]. Академічні установи регулярно постають перед вибором між розгортанням готових open-source продуктів та власною розробкою. Універсальні системи схильні перетворюватися на "технологічну в'язницю" через перевантаженість модулями, які фактично не використовуються [9, 20]. Спроби адаптувати такі масивні платформи під унікальні бізнес-правила конкретного університету часто супроводжуються

конфліктами ядра та невиправдано збільшують вартість підтримки.

Більшість тиражованих систем концептуально проєктувалися в епоху домінування синхронної веб-взаємодії. Їхнє ядро чинить об'єктивний опір при спробах імплементації подієво-орієнтованої архітектури, яка є критичною для стабільного функціонування месенджер-ботів. Замість нативної інтеграції доводиться конструювати нестабільні ланцюжки проміжних плагінів.

Зрештою, це формує ефект жорсткої прив'язки до екосистеми конкретного вендора (vendor lock-in). Реалізація нетипових вимог – як жорстке лімітування кількості активних груп для одного здобувача - вимагає прямого та ризикованого втручання у базу даних стороннього продукту.

Для обґрунтування доцільності кастомної розробки проведено порівняльний аналіз пропонованого рішення з типовими монолітними системами (табл. 1.3).

Таблиця 1.3

Порівняльний аналіз існуючих освітніх систем та розроблюваної вебплатформи

Критерій порівняння	**Moodle**	**Google Classroom**	**Розроблювана платформа**
Архітектурний підхід	Важкий моноліт	Хмарний моноліт	Легка мікросервісна архітектура
Інтеграція AI-модуля	Відсутня	Відсутня	Нативна
Канали комунікації	Внутрішні форуми та чати	Коментарі до завдань, Email-розсилка	Месенджер-бот + вебвіджет
Контроль відвідуваності	Ручний або через сторонні модулі	Ручний	Автоматизований (через бота)
Рольова гнучкість	Надмірно складна	Обмежена (викладач / студент)	Оптимізована

Зведені дані свідчать, що традиційні рішення програють у швидкості

комунікації, змушуючи користувача щоразу проходити цикл авторизації на сайті для отримання базової інформації. Хмарні аналоги не надають інструментарію для глибокої автоматизації специфічних обмежень, наприклад жорсткий ліміт на кількість груп для одного студента.

Розроблювана платформа нівелює ці прогалини, фокусуючись виключно на критично важливих функціях. Кастомна розробка формує довгострокову конкурентну перевагу, залишаючи кінцевий продукт стійким до навантажень [9, 18, 20].

1.5 Постановка завдання

Враховуючи результати попереднього аналізу предметної області та виявлені структурні прогалини наявних програмних рішень, головна мета кваліфікаційної роботи полягає у проєктуванні спеціалізованого програмного забезпечення. Йдеться про створення платформи керування інформаційними потоками між суб'єктами освітнього процесу. Проєктований вебзастосунок вимагає глибокої інтеграції зовнішніх комунікаційних модулів та алгоритмів штучного інтелекту для делегування рутинних організаційних завдань машині.

Відповідно до сформульованої мети виникає об'єктивна потреба у вирішенні низки науково-практичних завдань. Передусім слід розробити функціональну архітектуру з чітким розмежуванням прав доступу на базі рольової моделі. Видається доцільним імплементувати ізольовані робочі простори для трьох ключових категорій користувачів. Адміністратор системи керуватиме обліковими записами, формуватиме академічні групи, складатиме розклад і генеруватиме статистичну звітність. Викладацький склад отримає інструментарій для модерування закріплених дисциплін, публікації матеріалів та моніторингу активності. Своєю чергою, студентський інтерфейс має фокусуватися на доступі до актуального розкладу, ієрархічно структурованого контенту та механізмах фіксації власної присутності на лекціях.

Окремої уваги потребує проєктування специфічної бізнес-логіки. Критичною вимогою виступає обмеження належності одного здобувача освіти не більше ніж до двох активних академічних груп одночасно. Згадане обмеження підлягає жорсткій валідації на рівні серверної архітектури та бази даних. Також вимагається налаштувати механізми каскадного оновлення атрибутів. Зміна ідентифікаційних даних особи повинна миттєво реплікуватися в усіх суміжних сутностях. Це унеможливить виникнення інформаційних аномалій.

Наступним вектором роботи вбачається розробка комунікаційного ядра. Оперативна маршрутизація повідомлень вимагає розгортання двох незалежних каналів. Інтегрований вебвіджет візьме на себе первинну взаємодію з неавторизованими користувачами. Водночас спеціалізований месенджер-бот функціонуватиме як основний шлюз для верифікованих акаунтів, приймаючи звернення та фіксуючи явку. Інтеграція модуля штучного інтелекту в цей процес вбачається найскладнішим етапом. Модель машинного навчання має аналізувати семантику вхідних повідомлень, класифікувати їх за ступенем терміновості та генерувати релевантні відповіді на шаблонні запити. Нетипові інциденти алгоритм перенаправлятиме компетентному викладачу без часових затримок.

Фінальний комплекс завдань охоплює агрегацію статистичних даних навчального процесу та архітектурне планування. Програмний комплекс повинен накопичувати телеметрію відвідуваності й успішності з опцією експорту масивів даних у формати стандартизованих електронних таблиць.

Щодо технологічного стеку, серверну логіку доцільно реалізувати засобами мови Python у зв'язці з реляційною СУБД PostgreSQL. Клієнтський інтерфейс розроблятиметься з огляду на вимоги повної адаптивності під різні типи дисплеїв. Проєктний код підлягає суворому структуруванню для полегшення майбутнього масштабування продукту.

2 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1 Логічна модель даних у вигляді ER-діаграми

Проєктування архітектури даних виступає фундаментальним етапом життєвого циклу розробки, оскільки структурні помилки на рівні реляційної схеми вкрай важко піддаються рефакторингу на пізніших стадіях. Для розроблюваної платформи управління навчанням було спроєктовано нормалізовану реляційну базу даних. Логічна структура свідомо приведена до третьої нормальної форми (3NF) включно [21, 22]. З інженерної точки зору, відмова від денормалізації на початковому етапі продиктована необхідністю жорсткого запобігання аномаліям оновлення та видалення. В освітніх системах, де консистентність академічних записів є критичною, надмірність даних неприпустима.

Візуалізацію розробленої логічної структури та базових реляційних зв'язків між сутностями платформи наведено у вигляді ER-діаграми на рисунку 2.1.

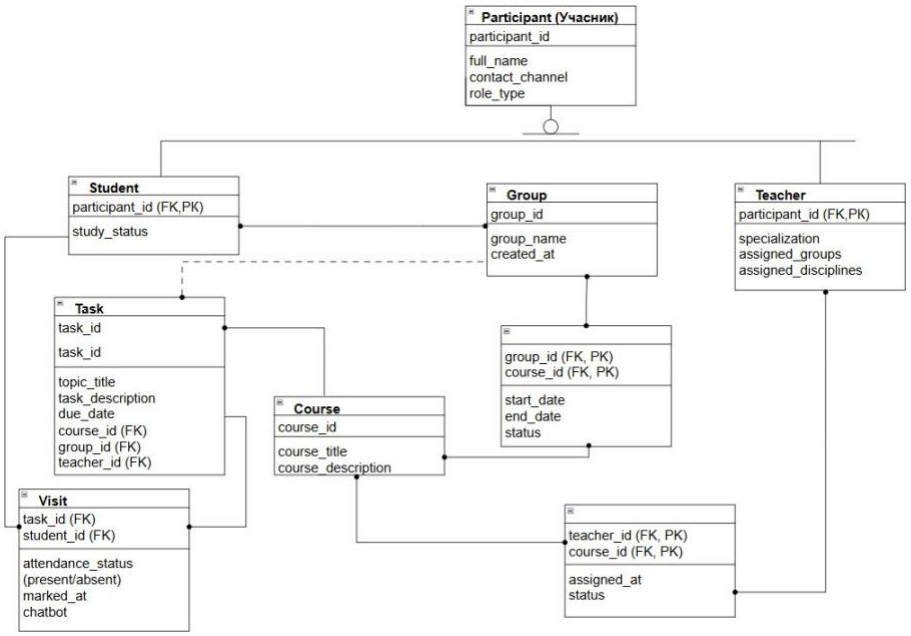


Рис. 2.1 – Логічна ER-модель бази даних системи

Проєктування архітектури даних базується на суворому врахуванні ієрархічної структури користувачів, що є критичним для елімінації надлишковості та гарантування цілісності інформаційних доменів. На розробленій логічній ER-діаграмі реалізовано патерн успадкування на рівні сутностей: базовий об'єкт «Учасник» (Participant) акумулює уніфіковані ідентифікаційні атрибути та рольові метадані суб'єктів. Спеціалізовані сутності «Студент» (Student) та «Викладач» (Teacher) розширюють базову модель через ідентифікуючі зв'язки за первинними ключами. Такий підхід забезпечує гнучке управління академічними статусами та профілізацією без денормалізації персональних даних.

Реляційна модель інтегрує повний цикл освітнього контуру: від формування контингенту навчальних груп (Group) та детермінації дисциплін (Course) до операційної фіксації занять (Task) і моніторингу присутності (Visit). Складні зв'язки типу «багато-до-багатьох», зокрема у вузлі взаємодії груп та навчальних курсів, імплементовані через асоціативні таблиці-шлюзи (Course_Offering). Це гарантує математичну точність обліку навчального навантаження та виключає виникнення аномалій при модифікації розкладу. Окремим функціональним сегментом виступає підсистема відвідуваності:

вона інтегрована з комунікаційним месенджер-ботом і забезпечує транзакційне збереження фактів присутності здобувача освіти із обов'язковою фіксацією часових міток на рівні серверного ядра.

Розмежування прав доступу та персоналізація профілів реалізовані через механізм типізованого успадкування сутностей. Базова таблиця Participant акумулює спільні дані (ПІБ, email, роль), що дозволяє уникнути дублювання полів аутентифікації. Водночас специфічні атрибути виділені в окремі пов'язані сутності Student та Teacher. Такий архітектурний підхід суттєво спрощує логіку перевірки токенів доступу на рівні бекенду, зберігаючи при цьому можливість гнучкого розширення профілю кожної ролі без засмічення загальної структури.

Академічний блок представлений сутностями курсів (Courses) та навчальних груп (StudyGroups). Взаємодія між здобувачами освіти та конкретними групами реалізована через проміжну асоціативну таблицю. Тут варто наголосити на використанні логіки каскадного видалення (ON DELETE CASCADE). Якщо навчальна група розформовується, система автоматично очищає всі пов'язані записи участі, що вивільняє адміністратора від необхідності ручної чистки "мертвих" даних.

Окремим викликом стала імплементація специфічного бізнес-правила: лімітування участі одного студента максимумом у двох активних групах одночасно. Замість перенесення цієї логіки виключно на рівень клієнтського застосунку, обмеження імплементоване глибоко в ядрі. Організація розкладу та моніторинг відвідуваності ізольовані в сутності Schedule та Attendance.

Подальша еволюція логічної архітектури вимагала імплементації підсистеми моніторингу практичної успішності. Для покриття цієї потреби реляційну модель було розширено трьома взаємозалежними сутностями: Assignments (академічні завдання), AssignmentSubmissions (результати виконання) та Grades (журнал оцінок). Базова сутність завдань інкапсулює критичні метадані, включаючи жорсткі часові ліміти (due_date). З огляду на тенденцію до децентралізації освітнього контенту, до моделі інтегровано

спеціалізований атрибут link_to_test. Це дозволяє викладачам делегувати процес оцінювання зовнішнім сервісам без штучного роздування локальної бази даних. Своєю чергою, збереження студентських робіт у таблиці подань вимагало жорсткого запобігання появі дублікатів. Цю колізію вирішено на фізичному рівні СУБД шляхом накладання складеного унікального обмеження UNIQUE(student_id, assignment_id). Завдяки цьому реляційна модель математично гарантує існування лише одного активного запису здачі для конкретного здобувача. Будь-які спроби повторного відправлення роботи ініціюють транзакцію оновлення, а не створення нового рядка. Замикає цей контур сутність оцінок, яка агрегує підсумковий бал та текстовий зворотний зв'язок викладача.

Що стосується комунікаційного пайплайну та інтелектуального модуля, для їхнього обслуговування спроєктовано сутність Messages. Це не просто таблиця для збереження історії листування, а повноцінний журнал аудиту. Вона фіксує ідентифікатор сесії, текст звернення, векторні метадані для ШІ та поточний статус обробки. Враховуючи асинхронну природу спілкування через ботів, транзакційність операцій запису стає критичною вимогою. База даних налаштована на використання механізмів відкату транзакцій (Rollback) у разі розриву з'єднання під час збереження масиву повідомлень, що гарантує абсолютну цілісність даних [23, 24].

2.2 Діаграма класів та кооперацій

Трансляція вимог предметної області у програмний код вимагає використання стандартизованих візуальних мов, серед яких UML залишається індустріальним еталоном [25]. Трансформація концептуальної моделі у площину об'єктно-орієнтованого проєктування вимагає суворої формалізації архітектури за допомогою UML-діаграми класів (повну схему архітектури класів та їх кооперацій наведено у Додатку В, рис. В.1). Запропонована модель віддзеркалює статичну структуру інформаційної платформи, інкапсульовані

атрибути сутностей та контракти їхньої взаємодії. Ядром програмного рішення виступає ієрархія користувацьких профілів. Використання класичного механізму успадкування дозволило екстрагувати базові атрибути автентифікації у загальний абстрактний суперклас. Своєю чергою, похідні класи («Студент», «Викладач», «Адміністратор») розширюють його функціонал специфічними методами, що унеможливило дублювання коду та заклало гнучкий фундамент для масштабування рольової моделі [26].

Спроєктовані класи пов'язані мережею асоціацій, які строго детермінують логіку освітнього процесу. Особливої уваги заслуговує інтеграція комунікаційного ядра: сутність «Повідомлення» виступає центральним транзакційним вузлом, що забезпечує маршрутизацію звернень та підтримку алгоритмів ескалації. Крім того, діаграма на рівні об'єктів інкапсулює критичні бізнес-правила (зокрема, жорстке лімітування належності здобувача освіти до академічних груп), переносячи відповідальність за гарантування цілісності даних безпосередньо в архітектуру прикладного застосунку.

Навчальні сутності об'єднані складними асоціативними зв'язками, які безпосередньо віддзеркалюють бізнес-логіку університету. Проте найбільш цікавим з інженерної точки зору є застосування структурних патернів проєктування.

Для абстрагування логіки доступу до даних від бізнес-правил було імплементовано патерн Repository. Безпосередні SQL-запити або ORM-виклики не змішуються з контролерами. Репозиторій створює проміжний шар, приймаючи на вхід об'єкти домену і самостійно вирішуючи, як зберегти їх у реляційній структурі [27]. Це робить бізнес-логіку платформи повністю незалежною від конкретної СУБД та створює ідеальні умови для модульного тестування (unit testing), оскільки роботу бази даних можна легко зімітувати (mocking).

Паралельно, для модуля маршрутизації звернень на базі штучного інтелекту, було застосовано патерн Strategy. Ринок великих мовних моделей

еволюціонує вкрай стрімко. Жорстка прив'язка коду до API конкретного провайдера, до прикладу Google Gemini, створює ризик технологічної залежності (vendor lock-in). Патерн Strategy дозволяє визначити загальний інтерфейс AIProviderInterface.

Відповідно, система може динамічно перемикатися між різними алгоритмами обробки або провайдерами моделей, не вимагаючи втручання у базовий код комунікаційного маршрутизатора [28].

А сама архітектура комунікації побудована на REST що забезпечує безпеку для вибраного патерну [29,30].

Такий архітектурний підхід є не лише безпечнішим з точки зору ізоляції компонентів та відмовостійкості, але й оптимально відповідає парадигмі ітеративної розробки програмного забезпечення. Зокрема, це є критично важливим при створенні мінімально життєздатного продукту (MVP) з подальшим поетапним нарощуванням функціоналу без ризику руйнування існуючої бізнес-логіки.

Наступним етапом об'єктно-орієнтованого проєктування виступає аналіз динамічної поведінки системи. Для візуалізації протоколів обміну повідомленнями між екземплярами класів під час виконання бізнес-сценаріїв побудовано діаграми кооперацій (Communication Diagrams). На розробленій комплексній схемі формалізовано три критичні транзакції: публікацію навчальних завдань із тригером розсилки сповіщень, обробку адміністративних інцидентів та алгоритм автоматизованої фіксації відвідуваності (детальну візуалізацію моделей кооперацій винесено до Додатка В, рис. В.3).

Архітектурною домінантою наведених кооперацій є подієво-орієнтований патерн взаємодії. Месенджер-бот виступає тут не просто пасивним інтерфейсом, а повноцінним системним вузлом (брокером). Він ініціює виклики методів та синхронізує стан між фізичними акторами (студентом, викладачем, модератором) і доменними сутностями платформи («Курс», «Завдання», «Заняття»). Подібна декомпозиція прикладних процесів

виконує функцію архітектурної верифікації: вона практично доводить, що спроєктована раніше діаграма класів є об'єктивно повною, а закладених атрибутів та методів достатньо для транзакційного обслуговування всього навчального циклу без введення надлишкових сутностей.

2.2.1 Сценарії використання (Use Cases) системи

Формалізація поведінкових патернів виступає невіддільною складовою об'єктно-орієнтованого аналізу. Для розуміння глобальної архітектури розроблено специфікацію прецедентів, яка регламентує взаємодію ключових акторів із платформою. Базова модель охоплює чотири основні ролі: «Студент», «Викладач», «Адміністратор» та допоміжну систему «Чат-бот». Кожен актор наділений специфічним набором прав та обмежень. Адміністратор виступає ядром управління, маючи доступ до ведення каталогу дисциплін, управління навчальними групами та загальним розкладом.

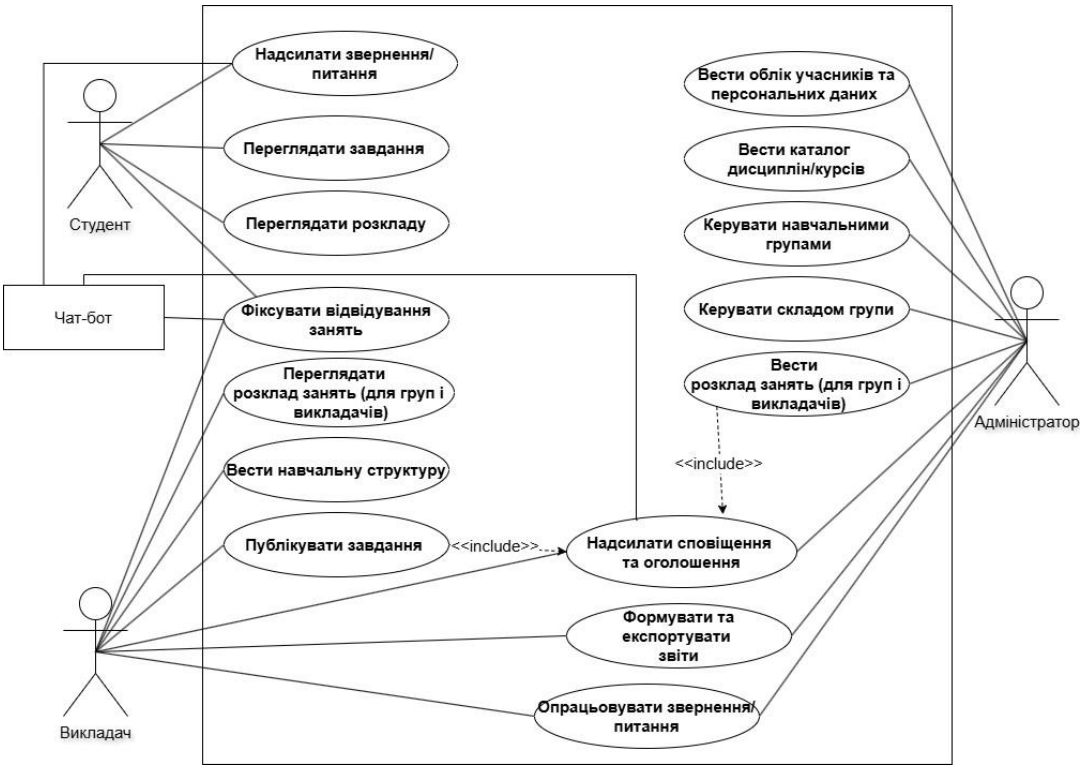


Рис. 2.2 – UML-діаграма прецедентів (Use Case)

Зі свого боку, викладач та студент взаємодіють безпосередньо в рамках навчального процесу. Сценарії викладача сфокусовані на публікації завдань, фіксації відвідуваності та обробці запитів. Студент є кінцевим споживачем контенту, чиї прецеденти обмежуються переглядом розкладу, завдань та надсиланням звернень. Інтеграція чат-бота розширює функціонал, дозволяючи автоматизувати рутинні процеси, такі як швидкий доступ до розкладу занять. Такий розподіл ролей забезпечує надійну stateless-архітектуру та чітке розмежування зон відповідальності.

З метою деталізації поведінкової моделі платформи та часової синхронізації процесів було побудовано загальну UML-діаграму послідовності (детальну візуалізацію часової взаємодії компонентів системи наведено у Додатку В, рис. В.2). Вона відображає динаміку взаємодії базових акторів (студента, викладача, адміністратора) через центральний комунікаційний вузол - модульний месенджер-бот. Цей архітектурний підхід підтверджує подієво-орієнтований характер системи: бот виконує роль асинхронного шлюзу, який маршрутизує інформаційні запити, транслює сповіщення про зміни в розкладі та забезпечує моніторинг відвідуваності.

Запропонована модель розкриває повний життєвий цикл інформаційних потоків. Процес розпочинається з первинного налаштування середовища адміністратором (внесення контингенту, формування груп та розкладу) і логічно переходить до етапу безпосередньої операційної діяльності. Зокрема, алгоритм фіксує механізми доставки сповіщень про нові навчальні завдання, логіку делегованої реєстрації присутності та дворівневу систему ескалації звернень. Остання дозволяє програмному ядру автоматично перенаправляти запити до адміністратора або профільного викладача залежно від семантичного контексту повідомлення.

Для деталізації операційних потоків та логіки виконання бізнес-процесів, закладених у прецедентах, спроєктовано UML-діаграму активності з розбиттям на зони відповідальності (swimlanes). Візуалізація алгоритмічних кроків дозволяє чітко відстежити життєвий цикл освітнього процесу в межах

платформи: від первинної конфігурації облікових записів, навчальних груп (з урахуванням обмеження належності студента до двох груп) та розкладу адміністратором до фінального генерування звітності. Застосування доріжок відповідальності (Студент, Чат-бот, Викладач, Адміністратор) наочно демонструє механізм делегування повноважень та строгу stateless-архітектуру системи.

Особлива увага на діаграмі приділяється паралельним процесам, де інтегрований чат-бот виступає ключовим фасилітатором. Він не лише ретранслює системні сповіщення щодо нових завдань чи змін у розкладі, а й бере на себе первинну обробку транзакцій. Коли студент ініціює запит або відмічає присутність, система перенаправляє ці дії через прошарок бота. Механізм розгалуження гарантує, що запити маршрутизуються виключно цільовому адресату (адміністратору для організаційних питань або викладачу для академічних), що суттєво знижує інформаційний шум. Цикл завершується синхронізованим формуванням масивів даних викладачами для подальшого експорту зведеного звіту адміністратором.

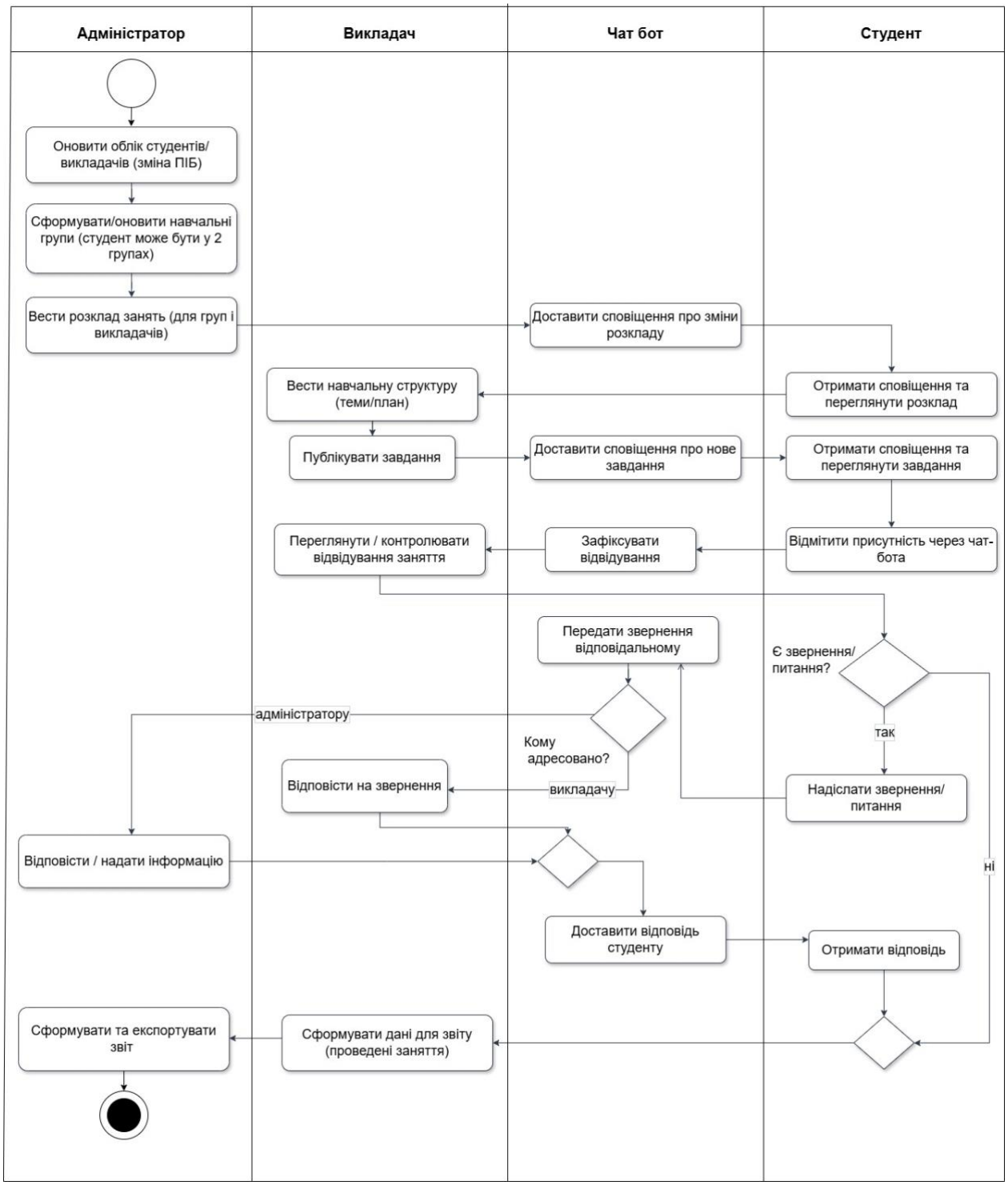


Рис. 2.3 – UML-діаграма активності базових бізнес-процесів системи

2.3 Діаграма пакетів

При виборі глобальної архітектури перед розробником постає дилема між класичним монолітом та мікросервісами. Мікросервісна архітектура, попри її популярність, часто впроваджується невиправдано, приносячи із

собою колосальні накладні витрати на оркестрацію контейнерів та управління розподіленими транзакціями. З огляду на специфіку освітньої платформи, було обрано компромісний, але інженерно зрілий підхід - модульний моноліт [31, 32].

Ця архітектурна парадигма передбачає фізичне розгортання єдиного застосунку, однак його внутрішня структура жорстко поділена на логічно ізольовані модулі. Комунікація між модулями відбувається виключно через контракти внутрішніх інтерфейсів, а не через пряме звернення до чужих баз даних.

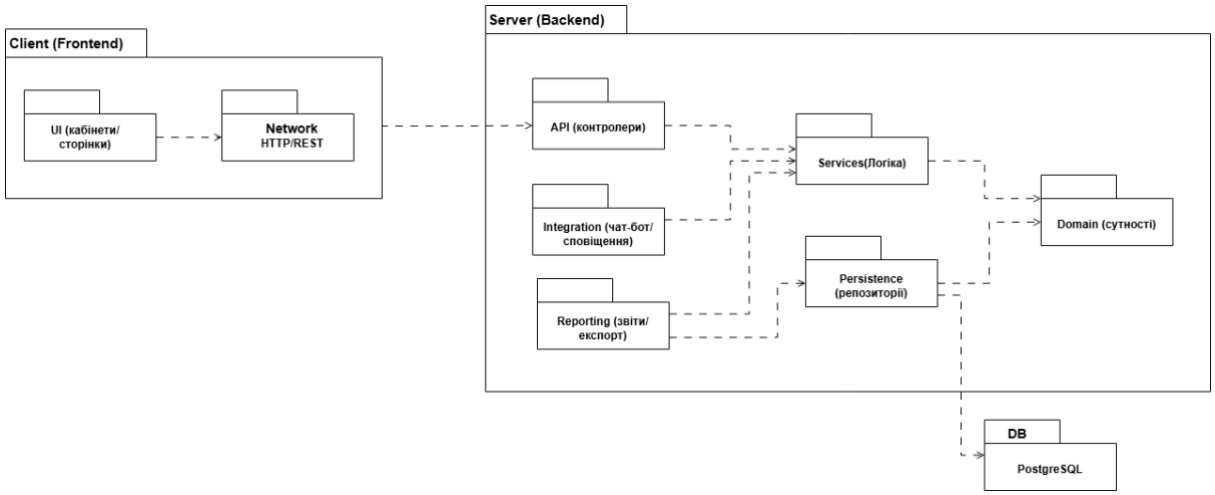


Рис. 2.4 – UML-діаграма пакетів

Серверний контур системи спроєктований у вигляді сукупно ізольованих фундаментальних пакетів, кожен з яких наділений строгою зоною відповідальності. Єдиною точкою входу для вхідного трафіку виступає пакет API (контролери). Він консолідує в собі маршрутизатори, функціонал яких обмежений прийомом HTTP-запитів від клієнтського рівня (Network та UI) та первинною обробкою Webhook-повідомлень від серверів месенджера.

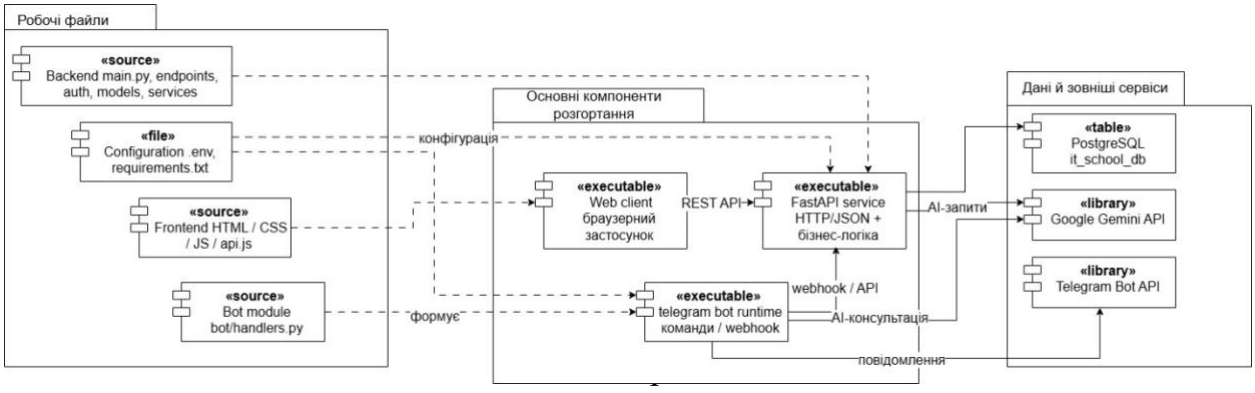
Пакет Domain (сутності) інкапсулює ключові структури даних та декларативні моделі об'єктно-реляційного відображення, тоді як шар Persistence (репозиторії) повністю абстрагує механізми взаємодії з базою даних PostgreSQL. Центральним обчислювальним вузлом платформи є пакет Services (Логіка). Саме до цього рівня контролери делегують очищені дані для

безпосередньої реалізації бізнес-правил. Даний сегмент, у синергії з модулями Integration (чат-бот/сповіщення) та Reporting (звіти/експорт), містить сервіси управління контингентом, алгоритми генерації розкладу та ізольовану підсистему AI-маршрутизації на базі мовних моделей [33, 34].

Застосована декомпозиція кодової бази гарантує архітектурну чистоту проєкту: розробникам вдалося досягти високої внутрішньої зв'язності (high cohesion) елементів кожного пакета при дотриманні принципу мінімального зачеплення (low coupling) між автономними модулями. Такий підхід забезпечує високу масштабованість: у разі зростання навантаження пакет Services та допоміжні компоненти можуть бути безболісно екстраговані в окремі мікросервіси. Для підтримки стабільності асинхронних інтеграційних процесів та запобігання блокуванню основного потоку виконання впроваджено механізм фонових задач (BackgroundTasks) [35].

2.4 Діаграма компонентів

Діаграма компонентів візуалізує топологію фізичного розгортання та взаємодії основних вузлів системи. Архітектура комунікації побудована на суворих принципах REST (Representational State Transfer), що відсікає необхідність збереження стану клієнта на сервері та робить систему надзвичайно гнучкою для інтеграції з мобільними застосунками в майбутньому [29, 30].



Бекенд-компонент платформи функціонує на базі сучасного мікрофреймворку FastAPI, який імплементує специфікацію ASGI (Asynchronous Server Gateway Interface), та тісно взаємодіє з реляційною системою управління базами даних PostgreSQL [36, 37]. Вибір асинхронного стеку продиктований необхідністю обслуговування великої кількості конкурентних з'єднань. Однак найскладнішим інженерним викликом на етапі розробки комунікаційного ядра стала інтеграція зовнішніх незалежних компонентів: API месенджера Telegram та хмарного середовища великої мовної моделі Google Gemini API.

Клієнтський рівень (Frontend) свідомо реалізовано як легковаговий вебзастосунок з використанням Vanilla JS та стандарту HTML5. Відмова від "важких" реактивних фреймворків на кшталт React чи Angular на даному етапі продиктована вимогою забезпечення максимальної швидкості завантаження інтерфейсу навіть при нестабільних мобільних з'єднаннях [38]. Взаємодія з сервером відбувається через асинхронні запити. Принцип відсутності стану (Statelessness) реалізовано через прикріплення JWT-токена (JSON Web Token) до заголовка кожного запиту. Серверу не потрібно звертатися до бази даних для перевірки сесії, що суттєво економить апаратні ресурси.

Специфіка роботи з цими сервісами полягає в тому, що звернення до нейромережі для семантичного аналізу тексту або відправка зворотного повідомлення у месенджер є типовими операціями вводу-виводу (I/O) через глобальну мережу Інтернет. Такі транзакції неминуче викликають непередбачувані часові затримки (latency), зумовлені швидкістю маршрутизації пакетів або навантаженням на сервери сторонніх хмарних провайдерів. В архітектурі асинхронних однопотокових серверів, де всі вхідні HTTP-запити обробляються єдиним циклом подій (Event Loop), це створює критичну загрозу.

Для усунення цієї вразливості та забезпечення високої доступності (High Availability) платформи було імплементовано нативний механізм фонових задач (BackgroundTasks) [35]. Архітектурно це працює наступним чином: коли

сервер отримує вхідний Webhook-запит від месенджера з новим повідомленням, він не очікує завершення його складного аналізу нейромережею. Замість цього система інкапсулює логіку обробки у незалежну фонову корутину (coroutine) і додає її до черги виконання фреймворку.

3 РОЗРОБКА ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Система управління інформаційною базою

Для реалізації серверної частини платформи було обрано Python у зв'язці з мікрофреймворком FastAPI [39]. Таке рішення продиктоване не лише зручністю розробки, а й практичною необхідністю забезпечити стабільну роботу системи під навантаженням. Проєктована система повинна безперебійно обробляти велику кількість одночасних запитів від студентів, викладачів та адміністративних користувачів, а FastAPI нативно підтримує асинхронну архітектуру (ASGI). Замість використання важких монолітних рішень, ця зв'язка дала змогу побудувати швидкий бекенд, який раціонально споживає апаратні ресурси та залишається зручним для подальшого масштабування.

Клієнтський інтерфейс розроблявся свідомо без використання популярних реактивних фреймворків на кшталт React чи Angular. Використано стандартні технології: HTML5, CSS3 та Vanilla JavaScript [40]. Мета такого підходу полягала у максимальній оптимізації, зменшенні кількості залежностей і спрощенні супроводу програмного продукту. Робочі кабінети завантажуються миттєво навіть за умов нестабільного мобільного зв'язку, що залишається критичним фактором для дистанційного навчання та доступу до освітніх матеріалів у різних умовах.

Комунікаційний контур спирається на два зовнішні інструменти. Обробку вхідних повідомлень у месенджері реалізовано через бібліотеку Aiogram 3 [41], яка забезпечує гнучку взаємодію з ботом та обробку користувацьких звернень. Семантичний аналіз тексту виконує хмарний модуль Google Gemini API [42], що дає змогу автоматизувати первинне

розпізнавання змісту повідомлень. Можна з певністю стверджувати, що саме ця програмна зв’язка взяла на себе основне операційне навантаження з первинної маршрутизації рутинних питань, розвантаживши викладачів і підвищивши швидкість реагування системи.

Як сховище даних задіяна об'єктно-реляційна СУБД PostgreSQL [43]. Освітній процес генерує щільну мережу залежностей між сутностями. Студенти закріплені за групами, групи мають свій розклад, а розклад безпосередньо пов'язаний з відвідуваністю та оцінками. Документоорієнтовані бази даних у таких умовах працювали б украй неефективно.

Структура розробленої бази даних складається з кількох ключових таблиць. Базовою сутністю виступає таблиця користувачів. Детальний опис її полів, типів даних та обмежень цілісності наведено у таблиці 3.1.

Таблиця 3.1

Специфікація таблиці користувачів (users)

Назва поля	**Тип даних**	**Обмеження**	**Опис**
id	Integer	Primary Key	Унікальний ідентифікатор користувача
email	Varchar(255)	Unique, Not Null	Електронна пошта (логін)
password_hash	Varchar(255)	Not Null	Захешований пароль (bcrypt)
full_name	Varchar(150)	Not Null	ПІБ користувача
role	Enum	Not Null	Роль (student, teacher, admin)
mes_id	BigInt	Unique, Nullable	ID акаунта у месенджер для бота
is_active	Boolean	Default True	Статус активності профілю

Представлена таблиця зберігає не лише авторизаційні дані, але й системні ролі (студент, викладач, адміністратор). Для інтеграції з

месенджером до неї додано специфічне поле ідентифікатора месенджер-акаунта.

Наступним логічним блоком стала таблиця збереження історії звернень. Її структуру деталізовано у таблиці 3.2.

Таблиця 3.2

Специфікація таблиці повідомлень ШІ-маршрутизатора (messages)

Назва поля	**Тип даних**	**Обмеження**	**Опис**
id	Integer	Primary Key	Унікальний ідентифікатор повідомлення
sender_id	Integer	Foreign Key	ID відправника (з таблиці users)
content	Text	Not Null	Текст оригінального запиту
reply	Text	Nullable	Текст відповіді (від ШІ або викладача)
status	Varchar(50)	Default 'pending'	Статус (pending, answered, resolved_ai)
is_escalated	Boolean	Default False	Маркер необхідності втручання викладача
timestamp	Timestamp	Default Now()	Час створення запиту

Вона фіксує текст запиту та системний маркер ескалації. Останній спрацьовує, якщо штучний інтелект не знайшов релевантної відповіді, після чого запит автоматично переходить до викладача. Декларативну модель сутності «Повідомлення» наведено на рис. 3.1.

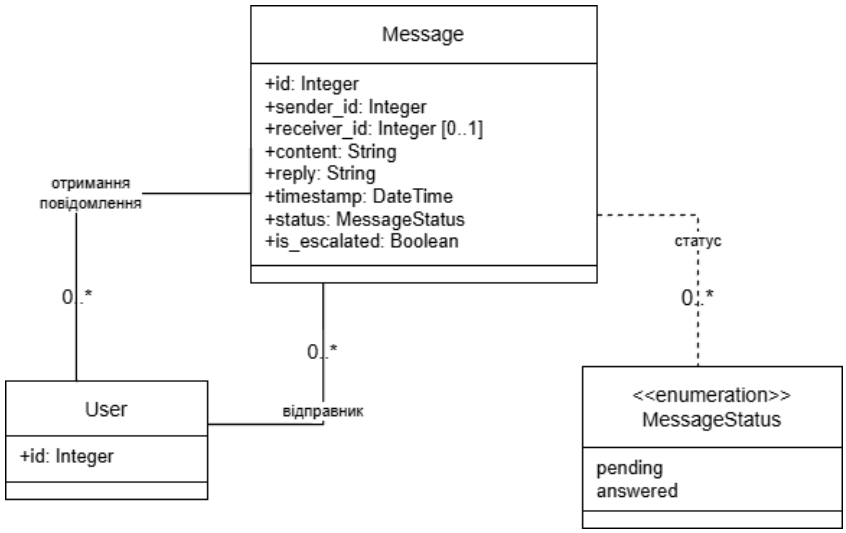


Рис. 3.1 – UML-діаграма класів декларативної моделі сутності «Повідомлення»

Облік присутності здобувачів на заняттях винесено в окрему таблицю, атрибутивний склад якої продемонстровано у таблиці 3.3.

Таблиця 3.3

Специфікація таблиці фіксації відвідуваності (attendance)

Назва поля	**Тип даних**	**Обмеження**	**Опис**
id	Integer	Primary Key	Унікальний ідентифікатор запису
student_id	Integer	Foreign Key	ID студента (з таблиці users)
schedule_id	Integer	Foreign Key	ID заняття (з таблиці schedule)
is_present	Boolean	Default True	Факт присутності на занятті
timestamp	Timestamp	Default Now()	Точний час фіксації

Зв'язок між студентами та навчальними групами реалізовано через проміжну асоціативну сутність. Оскільки один здобувач освіти може належати до кількох академічних груп одночасно, прямий зв'язок призвів би до дублювання записів. Взаємодію з базою даних на рівні програмного коду

реалізовано через ORM-бібліотеку SQLAlchemy [44]. Прямі SQL-запити в архітектурі проєкту практично відсутні, що знижує ризик синтаксичних помилок та вразливостей.

3.2 Розробка інформаційної бази

Фізична архітектура репозиторію відображає концепцію розділення відповідальності (Separation of Concerns). Для автоматизації розгортання середовища в кореневому каталозі розміщено пускові скрипти start (Windows).bat та start (Mac-Linux).sh, а також файл seed.py для первинного наповнення бази даних.

Серверна частина згрупована в директорії backend/app. Головним файлом, що ініціалізує ASGI-сервер Uvicorn, виступає main.py. Для уникнення монолітності коду всі HTTP-маршрути рознесені по окремих контролерах у папці backend/app/api/endpoints/.

Наприклад, логіка обробки повідомлень чату та прийому вебхуків від месенджера фізично ізольована у файлах chat.py та webhook.py. Реляційні моделі бази даних та схеми валідації вхідних даних (Pydantic) описані у файлах models.py та schemas.py відповідно. Водночас взаємодію із зовнішніми API, зокрема алгоритми звернення до штучного інтелекту, винесено в ізольований модуль backend/app/services/ai_service.py.

Клієнтська частина, розташована в директорії frontend/, реалізована без використання реактивних фреймворків. Увесь інтерфейс побудований на статичних HTML-сторінках (зокрема dashboard.html, admin-dashboard.html, teacher-dashboard.html), стилізованих за допомогою нативних CSS-модулів. Динаміка та мережеві запити забезпечуються скриптами api.js та script.js. Наявність спеціального файлу mockData.js дозволяє тестувати візуальну частину в ізольованому режимі без підключення до реального бекенду. Цей підхід дав змогу звести до мінімуму час до повної інтерактивності (Time to

Interactive). Графічний інтерфейс чітко розмежований відповідно до ролей користувачів, щоб не перевантажувати пристрої зайвими DOM-елементами.

Робочий простір студента орієнтований на швидкий доступ до навчального контенту. На головній панелі відображається лише критично важлива інформація: найближчі дедлайни, актуальний розклад занять та загальний прогрес. Важливим елементом інтерфейсу є плаваючий віджет чату. Він доступний з будь-якої сторінки платформи і слугує єдиним вікном для зв'язку зі штучним інтелектом.

Як виглядає інтерфейс робочого кабінету студента, показано на рис. Д.1 (див. Додаток Д).

Інтерфейс викладача має принципово іншу структуру. Головним інструментом тут виступає панель вхідних запитів («Teacher Inbox»). Сюди система автоматично перенаправляє ті питання від здобувачів освіти, для яких алгоритм AI-маршрутизатора не зміг згенерувати автоматичну відповідь або які були класифіковані як такі, що потребують участі викладача чи адміністратора. Панель вхідних повідомлень викладача наведено на рис. Д.2 (див. Додаток Д).

Зі свого боку, адміністративна панель призначена виключно для моніторингу загальної статистики відвідуваності, а також для керування розкладом і користувачами системи (рис. Д.3, див. Додаток Д).

Взаємодія між фронтендом та сервером відбувається виключно через REST API. Для покриття потреб клієнтської частини розроблено набір відповідних ендпоінтів. Детальну специфікацію та маршрутизацію REST API наведено у таблиці 3.4. Зазначені кінцеві точки забезпечують виконання базових операцій для всіх ключових сутностей системи, зокрема користувачів, розкладу, повідомлень і навчальних даних. Такий підхід гарантує безпечний та структурований обмін інформацією, чітко розмежовуючи зони відповідальності між клієнтом і сервером.

Таблиця 3.4

Специфікація основних REST API маршрутів системи

HTTP Метод	**Маршрут (Endpoint)**	**Тіло запиту (Body)**	**Опис дії**	**Доступ**
POST	/api/auth/login	username, password	Автентифікація користувача, видача JWT токена	Всі
GET	/api/users/me	Немає	Отримання даних профілю поточного користувача	Авторизовані
POST	/api/chat/ask	message (String)	Відправка повідомлення до AI-асистента	Студенти
GET	/api/messages/inbox	Немає	Отримання списку ескальованих запитів	Викладачі, Адміни
POST	/api/messages/reply/{id}	reply (String)	Надання відповіді викладачем	Викладачі
GET	/api/schedule/today	Немає	Отримання розкладу занять на день	Студенти
POST	/api/attendance/checkin	schedule_id	Фіксація присутності студента на занятті	Студенти
GET	/api/admin/stats	Немає	Агрегована статистика системи для дашборду	Адміни

Форматом обміну даними обрано легковаговий JSON. Оскільки платформа функціонує за Stateless-архітектурою, сервер не зберігає стан сесії. Для ідентифікації застосовується JWT-токен, який генерується під час успішного логіну (маршрут auth.py) і потім додається клієнтом до заголовка кожного наступного HTTP-запиту.

Успішна авторизація завершується тим, що сервер повертає клієнту структуровану відповідь. Вона містить безпосередньо згенерований криптографічний маркер, його тип, а також базову інформацію про профіль користувача: унікальний ідентифікатор, електронну пошту, системну роль та повне ім'я. Ці дані зберігаються на стороні браузера і використовуються для локальної маршрутизації між екранами без необхідності виконання повторних запитів до бази даних.

Якщо криптографічний токен відсутній або термін його дії вичерпано, система відхиляє доступ ще на етапі перехоплення запиту. Контролери бізнес-логіки при цьому не навантажуються. Це створює додатковий рівень захисту бази даних від несанкціонованого сканування або витоку конфіденційної інформації.

3.3 Вибір інструментарію та розробка клієнтської частини

Для вирішення проблеми інформаційного перевантаження викладачів у систему інтегровано модуль семантичного аналізу на базі Google Gemini API. Його робота функціонально поєднана з месенджер-ботом, написаним за допомогою асинхронної бібліотеки Aiogram 3. Процес обробки звернень побудований за суворим, але гнучким алгоритмом.

Спочатку студент формує запит через месенджер або плаваючий вебвіджет. Бекенд перехоплює цей текст. Оскільки мережевий виклик до хмарної нейромережі може тривати кілька секунд, цю операцію свідомо загорнуто у фонові задачі (BackgroundTasks) мікрофреймворку FastAPI. Це критично важливо. Завдяки такому рішенню головний цикл подій сервера не блокується, і система продовжує безперебійно приймати запити від інших користувачів.

Сам процес класифікації намірів спирається на жорсткий промпт-інжиніринг. Щоб уникнути так званих інформаційних галюцинацій, алгоритму категорично заборонено генерувати вільний неструктурований текст. Модель зобов'язана повернути валідний JSON-об'єкт із визначеною категорією запиту

та згенерованою відповіддю. Якщо нейромережа розпізнає типове організаційне питання (наприклад, уточнення розкладу або дедлайну), вона самостійно формує релевантну відповідь. Бот миттєво ретранслює її здобувачу освіти. Інтерфейс взаємодії з месенджер-ботом наведено на рис. Д.4 (див. Додаток Д).

Однак логіка виконання змінюється, якщо запит має складний академічний характер або стосується адміністративних питань (скажімо, прохання перенести термін здачі роботи). У такому разі система примусово змінює системний статус повідомлення на очікування (pending) і активує прапорець ескалації. Це звернення миттєво з'являється у робочому кабінеті викладача для ручного опрацювання. Схему роботи цього інтелектуального маршрутизатора наведено на рис. 3.2.

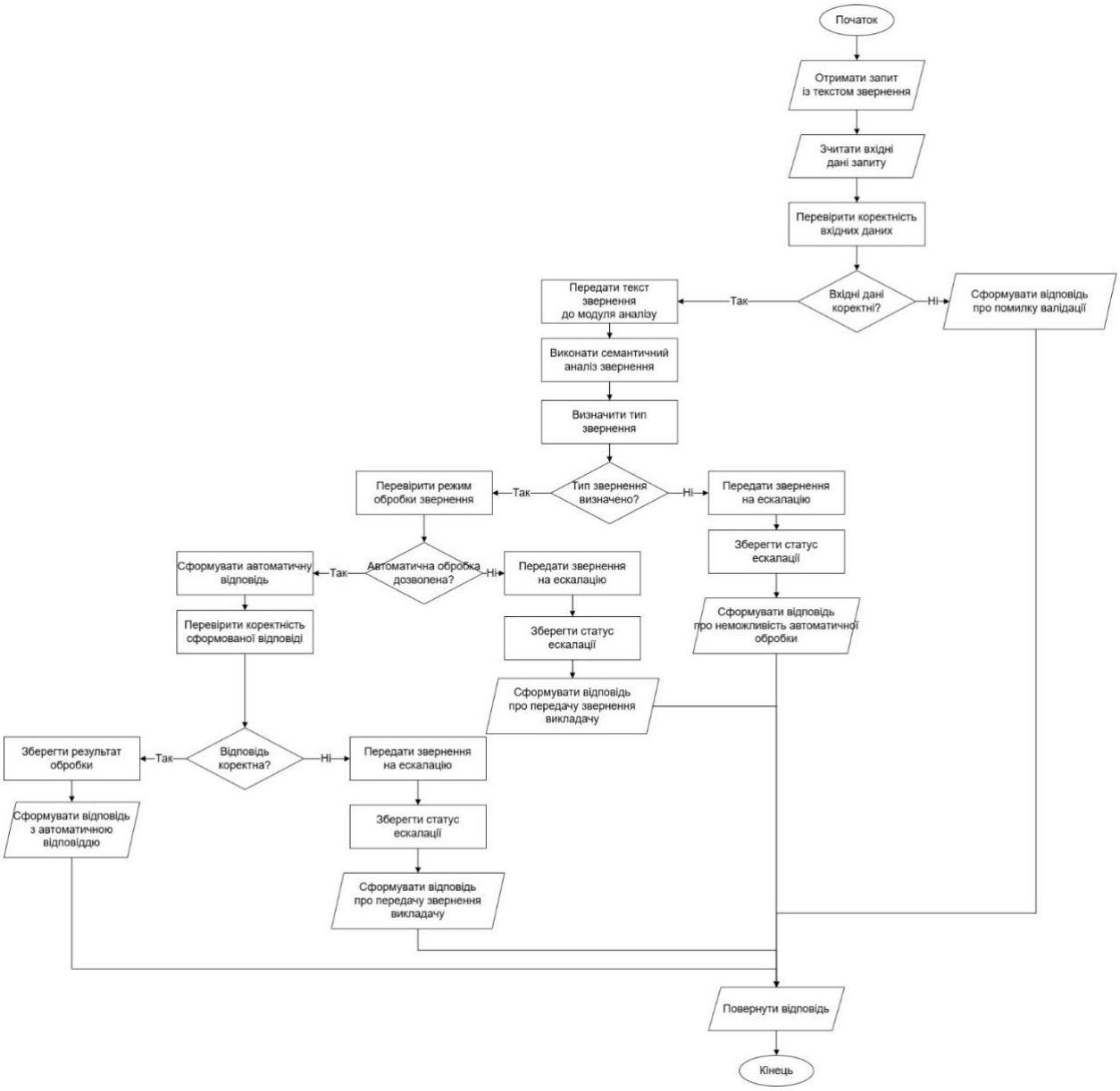


Рис. 3.2 – Схема алгоритму інтелектуальної обробки та маршрутизації

Окремим обчислювальним блоком виступає підсистема контролю відвідуваності через месенджер. Її архітектура вимагає багаторівневої валідації. Студент надсилає боту спеціальну команду з ідентифікатором заняття. Програмне ядро спочатку перевіряє криптографічну прив'язку месенджер-акаунта до профілю на вебплатформі (ця синхронізація виконується користувачем одноразово). Далі алгоритм зіставляє поточний серверний час із запланованим розкладом. Якщо заняття дійсно триває, ініціюється пошук дублікатів у базі даних. Цей крок запобігає проблемі множинних відміток від одного студента. Лише після успішного проходження

всього каскаду фільтрів факт присутності транзакційно фіксується у таблиці відвідуваності. Водночас покрокову логіку цього алгоритму у вигляді блок-схеми фіксації відвідуваності проілюстровано на рис. 3.4.

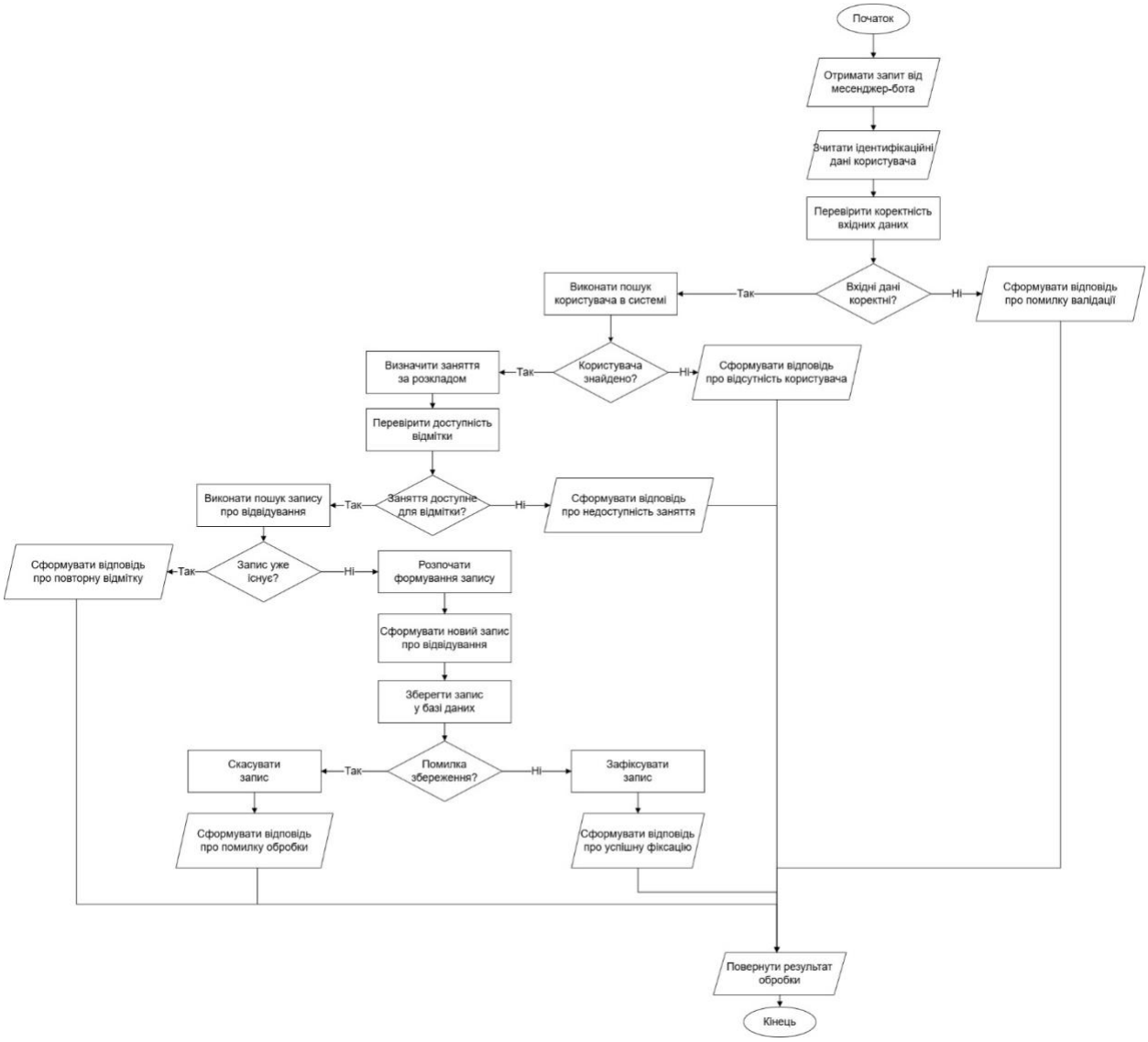


Рис. 3.4 – Блок-схема алгоритму фіксації відвідуваності занять

Можна з упевненістю констатувати, що такий підхід дозволив повністю виключити людський фактор з процесу збору операційної статистики. Зібрана телеметрія відвідуваності формує об'єктивну базу для подальшої агрегації даних та вивантаження підсумкових звітів адміністратором.

3.4 Алгоритмізація та програмування інтелектуальних модулів

Щоб захистити базу даних від некоректних записів під час здачі практичних робіт, застосовано підхід раннього повернення (Guard Clauses). Уся ця логіка інкапсульована в контролері assignments.py. Коли студент відправляє виконане завдання, сервер не намагається сліпо зберегти його. Спочатку виконується послідовна перевірка.

Алгоритм дивиться, чи існує запитуваний курс, і чи належить до нього цей конкретний користувач. Виявлення будь-якої невідповідності миттєво перериває виконання коду і повертає помилку доступу (HTTP 403). Найвищий пріоритет у цьому ланцюжку має перевірка наявності оцінки. Якщо викладач уже перевірив роботу та виставив бал, запис назавжди блокується для редагування. Лише за умови успішного проходження всіх цих фільтрів транзакція фіксується у базі даних (відповідну блок-схему наведено на рис. В.4).

Щоб допомогти здобувачам освіти орієнтуватися в академічному навантаженні, імплементовано алгоритм динамічного сортування. Завдання, до дедлайну яких залишається менше трьох діб, автоматично отримують на бекенді маркер найвищої терміновості. Фронтенд просто підхоплює цей статус і виводить такі роботи на початок списку. Це архітектурне рішення позбавляє браузер необхідності виконувати зайві обчислення з датами на стороні клієнта.

Окремого пояснення вимагає клієнтський прошарок для API-запитів. Усю логіку мережевої взаємодії винесено в єдиний файл api.js. Щоб мати змогу демонструвати інтерфейс платформи без повноцінного розгортання серверної інфраструктури, туди додано глобальний прапорець DEMO_MODE. Якщо його активовано, застосунок перехоплює HTTP-запити. Замість реального звернення до бекенду, він повертає тестовий масив даних із файлу mockData.js, імітуючи мережеву затримку. Це вкрай зручно для презентації проєкту. У бойовому ж режимі цей прошарок просто прикріплює JWT-токен до кожного запиту і транслює його на сервер.

Процес авторизації (модуль auth.py) також має строгий алгоритм. На етапі входу немає різниці між типами користувачів. Бекенд шукає збіг електронної пошти, потім звіряє хеш пароля, і лише після цього генерує маркер доступу з вшитою системною роллю. Подальша маршрутизація до відповідного робочого кабінету (студента, викладача чи адміністратора) відбувається вже на стороні клієнта після розшифровки токена. Логіку авторизації та інші базові процеси, такі як створення курсів викладачами, детально проілюстровано на блок-схемах у додатках (рис. В.5, В.6).

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

4.1 Тестування системи

Щоб переконатися, що розроблена платформа працює стабільно і не "впаде" від дій реальних користувачів навіть у моменти пікових навантажень, проведено комплексне функціональне тестування. Оскільки бекенд побудований як модульний моноліт, найзручнішим та найбільш репрезентативним виявився метод "чорного ящика".

Застосовано підхід абстрагування від внутрішньої структури коду та імітовано поведінку різних типів користувачів із застосуванням крайніх граничних значень (Edge Cases) для вхідних даних.

Усі експерименти проводилися в суворо ізольованому тестовому середовищі, яке повністю дублювало конфігурацію бойового сервера. Це було критично важливо для того, щоб безпечно генерувати штучні навантаження та помилки, не ризикуючи пошкодити основну базу даних або порушити цілісність існуючих записів.

Найбільше уваги приділено безпеці та стійкості до несанкціонованого втручання. В академічних системах витік персональних даних або навмисна зміна оцінок є критичною вразливістю, яка може нівелювати довіру до всієї організації навчального процесу.

Оскільки архітектура проєкту має єдину точку входу для всіх клієнтських запитів, головну увагу було зосереджено на жорсткій перевірці механізмів рольової ізоляції (RBAC), коректності розмежування прав доступу та надійності валідації криптографічних токенів. Це дало змогу перевірити, чи не може користувач отримати доступ до функцій, які не відповідають його ролі в системі.

Результати перевірки спроби нелегітимного доступу до

адміністративної панелі зі студентського акаунта (симуляція горизонтального підвищення привілеїв) зафіксовані у відповідному протоколі (таблиця 4.1).

Таблиця 4.1

Тестовий протокол: Перевірка рольової моделі доступу (RBAC)

Параметр	**Опис протоколу ТС-01**
Назва аудиту	Ізоляція адміністративних маршрутів від несанкціонованого доступу
Передумова	Суб'єкт авторизований у системі під роллю студента. Згенеровано валідний токен доступу.
Кроки відтворення	1. Ініціація прямого мережевого запиту до закритої кінцевої точки панелі адміністратора. 2. Фіксація коду відповіді серверного ядра.
Очікуваний стан	Сервер відхиляє транзакцію. Повертається статус-код HTTP 403 (Forbidden).
Фактичний стан	Запит відхилено на рівні middleware. Отримано статус 403. Виконано примусове перенаправлення суб'єкта.

Тест показав, що JWT-прошарок надійно відхиляє такі запити. Також перевірялась можливість штучно пошкодити маркер доступу в локальному сховищі браузера. Бекенд одразу розпізнав фальсифікацію криптографічного підпису, відхилив некоректний запит та примусово скинув сесію користувача.

Наступним кроком стала перевірка транзакційної цілісності бізнес-логіки. Метою перевірки було з’ясувати, чи зможе студент підмінити файл або змінити відповідь після того, як викладач уже перевірив роботу і виставив бал. Для цього було зімітовано повторний POST-запит на кінцеву точку збереження завдання з уже наявним результатом перевірки.

Як видно з наступного протоколу тестування (таблиця 4.2), база даних успішно заблокувала цю операцію, зберігши цілісність раніше зафіксованих навчальних результатів.

Таблиця 4.2

Тестовий протокол: Верифікація транзакційності процесу оцінювання

Параметр	**Опис протоколу ТС-02**
Назва аудиту	Блокування модифікації об'єкта після виставлення академічної оцінки
Передумова	Студент має історію подання роботи. Викладач зафіксував підсумковий бал (наявний запис у БД).
Кроки відтворення	1. Суб'єкт відкриває інтерфейс виконаного завдання. 2. Формується нове тіло відповіді з наступним відправленням POST-запиту на кінцеву точку збереження.
Очікуваний стан	Бекенд аналізує консистентність бази. Операція блокується. Повертається помилка HTTP 400 (Bad Request). База даних не оновлюється.
Фактичний стан	Мережева відповідь HTTP 400. Аномалій цілісності даних не зафіксовано. Зміни відхилено.
Статус виконання	Пройдено

Далі перевірялася робота комунікаційного ядра. Було важливо зрозуміти, чи дійсно мовна модель розрізняє семантику контексту. Спочатку боту надсилалися технічні питання щодо дисциплін, і алгоритм успішно видавав миттєві підказки. Потім тактику було змінено і надіслано організаційне питання про перенесення дедлайну. Нейромережа відпрацювала чітко. Алгоритм не став вигадувати відповідь, а привласнив запиту маркер ескалації та перенаправив його безпосередньо у робочий кабінет викладача.

Фінальним етапом стала перевірка месенджер-бота. Головний ризик тут полягає у несанкціонованих запитах та навмисному дублюванні відміток про присутність на лекціях. Під час надсилання команди реєстрації з невідомого акаунта, бот очікувано перервав виконання скрипта і висунув вимогу надати токен прив'язки. Результат цього тесту наведено у протоколі аудиту комунікаційного ядра (таблиця 4.3).

Таблиця 4.3

Тестовий протокол: Захист контуру месенджер-бота

Параметр	**Опис протоколу ТС-03**
Назва аудиту	Спроба реєстрації відвідуваності без синхронізації профілю користувача
Передумова	Суб'єкт взаємодіє з ботом, проте ідентифікатор месенджера відсутній у базі даних платформи.
Кроки відтворення	1. Користувач вводить команду ініціації фіксації присутності.
Очікуваний стан	Бот перериває виконання скрипта. Виводиться системна вимога щодо криптографічної прив'язки профілю. Запис до транзакційної таблиці не вноситься.
Фактичний стан	Командний запит відхилено перехоплювачем. Надано інструкцію з верифікації. Записів у БД не створено.
Статус виконання	Пройдено

Після успішної авторизації в месенджері було здійснено кілька спроб поспіль відправити команду присутності на одну й ту саму пару. Завдяки унікальним обмеженням, які раніше було закладено в структуру бази даних, система зафіксувала лише перший клік. На всі наступні спроби бекенд повернув повідомлення про наявність раніше зафіксованої відмітки, математично унеможлививши розмноження записів.

4.2 Вимоги до апаратного та програмного забезпечення

Для стабільної роботи розробленої платформи спроєктовано розподілену клієнт-серверну архітектуру. Щоб наочно показати, як саме розподіляються обчислювальні навантаження між пристроями користувачів, сервером і зовнішніми API, побудовано діаграму розгортання (рис. 4.1).

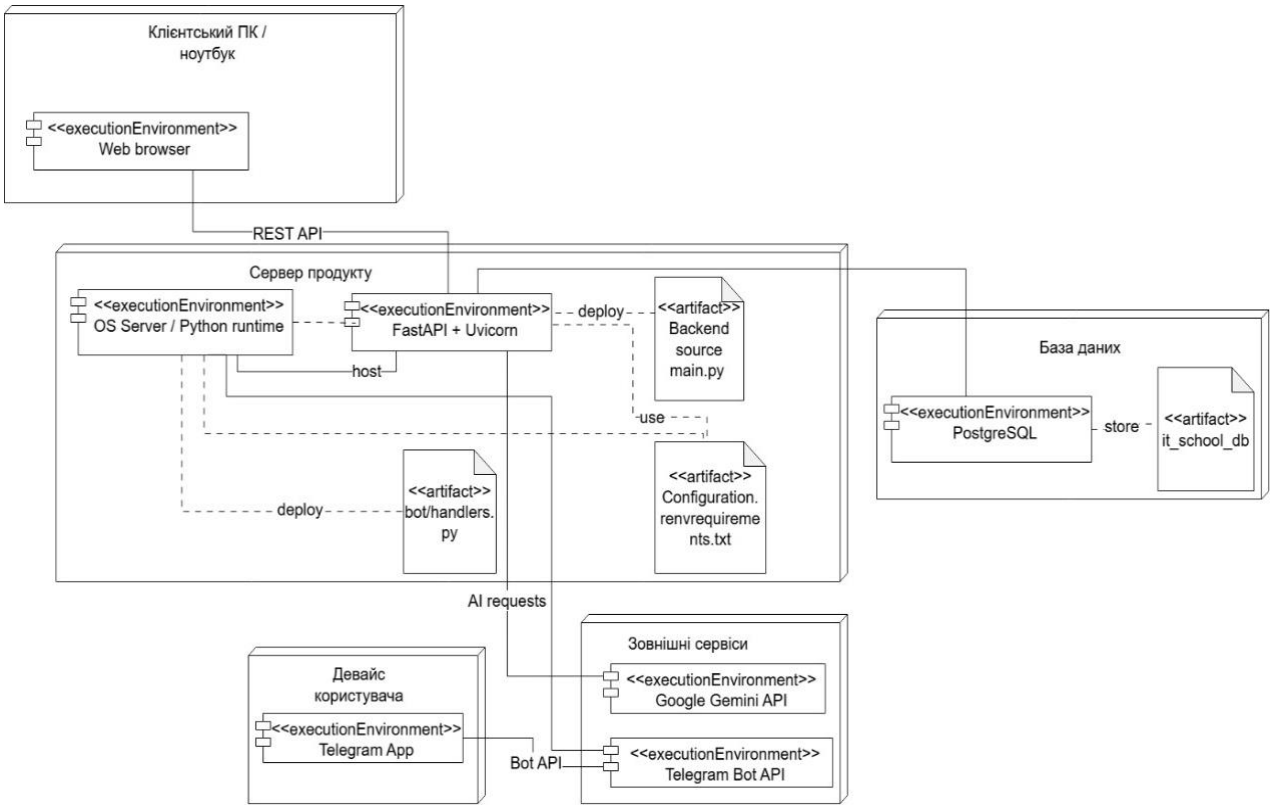


Рис. 4.1 – UML-діаграма розгортання (топологія) системи

Інфраструктура проєкту фізично розділена на кілька вузлів. Клієнтський рівень – це пристрої студентів та викладачів (смартфони, ноутбуки). Їхнє завдання зводиться виключно до рендерингу HTML/CSS та запуску клієнта месенджера.

Усі складні обчислення винесено на сервер застосунку. Саме там працює середовище Python з ASGI-сервером Uvicorn, який паралельно обслуговує FastAPI та скрипти месенджер-бота. Для зберігання даних передбачено окремий сервер із базою даних PostgreSQL. Комунікація між бекендом і хмарним модулем Google Gemini API відбувається виключно через захищені HTTPS-з'єднання, що забезпечує безпечну передачу запитів і відповідей.

Спираючись на цю топологію та результати тестування, сформовано вимоги до апаратного забезпечення. На етапі запуску (MVP) бекенд і базу даних можна тримати на одній віртуальній машині. Специфікації для сервера застосунку наведено у таблиці 4.4.

Таблиця 4.4

Мінімальні та рекомендовані апаратні вимоги до сервера застосунку (FastAPI)

Характеристика	**Мінімальні вимоги**	**Рекомендовані вимоги (Production)**
Процесор (CPU)	1 vCPU (від 2.0 GHz)	2-4 vCPU (від 2.5 GHz)
Оперативна пам'ять	1 GB	2-4 GB
Накопичувач (ROM)	10 GB SSD	20 GB SSD
Мережа	100 Mbps, статична IP	1 Gbps, статична IP

Мінімальні параметри обґрунтовані просто: хоча асинхронний фреймворк добре працює навіть на одному ядрі процесора, для паралельного виконання фонових задач (зокрема, очікування відповіді від штучного інтелекту) потрібні додаткові потоки.

Оперативна пам'ять на рівні 2-4 ГБ необхідна для кешування ORM-моделей та обробки вхідних JSON-файлів. Окремо варто наголосити, що для нормальної роботи вебхуків месенджер-бота серверу обов'язково потрібна статична IP-адреса.

Вимоги до сервера бази даних виділено у таблиці 4.5.

Таблиця 4.5

Вимоги до апаратного забезпечення сервера бази даних (PostgreSQL)

Характеристика	**Рекомендовані параметри**
Процесор (CPU)	2 vCPU
Оперативна пам'ять	2-4 GB
Накопичувач (ROM)	20+ GB NVMe SSD

Якщо контингент студентів значно зросте, PostgreSQL краще винести на окрему машину. Для бази потрібен швидкий твердотільний накопичувач

(NVMe SSD), щоб уникнути затримок під час масової фіксації відвідуваності на початку пари.

Програмний фундамент системи розроблявся з орієнтацією на кросплатформність. Вимоги до серверної інфраструктури вказано у таблиці 4.6.

Таблиця 4.6

Програмні вимоги до серверної інфраструктури

Компонент	**Вимоги до ПЗ та версій**
Операційна система	Linux (Ubuntu 22.04 LTS / Debian 11)
Середовище виконання	Python 3.10 або вище
СУБД	PostgreSQL 14.x або вище
Вебсервер (ASGI)	Uvicorn

Як базову ОС рекомендується використовувати Linux (Ubuntu або Debian), оскільки це індустріальний стандарт для розгортання Python-застосунків.

Натомість програмний стек клієнтського рівня свідомо зведено до мінімуму. Завдяки концепції Thin Client ("Тонкий клієнт") і відмові від важких фронтенд-фреймворків, платформа не висуває жодних специфічних вимог до пристроїв користувачів, окрім наявності сучасного браузера та клієнта Месенджеру. Системні вимоги для клієнта наведено у таблиці 4.7.

Таблиця 4.7

Програмні вимоги до клієнтського рівня

Компонент	**Вимоги**
Веббраузер	Google Chrome 90+, Mozilla Firefox 88+, Safari 14+
Месенджер	Клієнтський застосунок обраного месенджера
Мережевий доступ	Протоколи HTTP/HTTPS, відкритий порт 443

4.3 Склад інсталяційного пакета та керівництво користувача

4.3.1 Структура файлів інсталяційного пакета

Для зручного перенесення проєкту з локального середовища розробки на бойовий сервер сформовано цілісний інсталяційний пакет. Він фізично розділений на дві незалежні директорії: серверну (backend/) та клієнтську (frontend/). Такий розподіл гарантує ізоляцію бізнес-логіки від візуального представлення та дозволяє масштабувати ці компоненти незалежно один від одного. У серверній частині містяться всі необхідні конфігураційні артефакти, включаючи специфікацію залежностей requirements.txt та шаблон змінних оточення .env.example. Важливо зазначити, що конфігураційні файли не містять реальних секретних ключів з міркувань безпеки, а лише демонструють необхідну структуру для подальшого налаштування. Клієнтський пакет містить набір статичних HTML-сторінок та оптимізованих JS/CSS-файлів, повністю готових до роздачі будь-яким вебсервером без потреби у додатковій компіляції чи складній збірці. Повний вихідний код розробленої навчальної платформи розміщено у відкритому репозиторії, посилання на який наведено у Додатку А.

4.3.2 Інструкція з розгортання та ініціалізації системи

Процес розгортання системи максимально автоматизовано, щоб уникнути помилок ручного конфігурування та звести до мінімуму людський фактор на етапі впровадження. Системному адміністратору достатньо скопіювати файл .env.example, перейменувати його на .env та вписати туди реальні криптографічні ключі, токени доступу до API та облікові дані для підключення до бази даних.

Після цього потрібно запустити скрипт start (Windows).bat або start (Mac-Linux).sh, залежно від операційної системи хоста. Скрипт послідовно виконує

всі етапи ініціалізації: самостійно створює ізольоване віртуальне середовище, встановлює необхідні Python-бібліотеки з requirements.txt, застосовує міграції до бази даних через утиліту Alembic для формування правильної реляційної структури і, зрештою, запускає ASGI-сервер.

Щоб не тестувати систему на порожній базі та одразу перевірити коректність її роботи, також створено допоміжний скрипт seed.py. Його одноразовий запуск автоматично наповнює реляційні таблиці тестовим контингентом користувачів, навчальними дисциплінами, групами та розкладом.

4.3.3 Керівництво користувача та сценарії взаємодії

Сценарії взаємодії користувачів із платформою побудовані за принципом мінімізації когнітивного навантаження та інтуїтивної зрозумілості інтерфейсу. Після авторизації студент одразу потрапляє у свій персональний дашборд, де агрегується вся актуальна інформація щодо поточного прогресу та найближчих дедлайнів. Якщо у здобувача освіти виникає організаційне або академічне питання, він може звернутися до системи через плаваючий віджет чату на сайті або безпосередньо у месенджер-бот. Нейромережа виконує семантичний аналіз тексту вхідного повідомлення. Якщо питання складне або стосується специфічних академічних нюансів, алгоритм не намагається згенерувати випадкову відповідь. Натомість студент отримує сповіщення про ескалацію запиту. Викладач, авторизувавшись у системі, бачить цей запит у спеціалізованому розділі «Teacher Inbox» і відповідає на нього напряму через панель керування, після чого відповідь автоматично транслюється студенту.

Для повноцінної роботи з месенджер-ботом (зокрема для обліку відвідуваності) студентові потрібно лише один раз відправити команду синхронізації профілю, яка генерується в його особистому кабінеті. Після успішного зв'язування акаунтів здобувач отримує змогу фіксувати свою присутність на лекціях відправкою короткої команди. Цей механізм повністю

позбавляє викладача необхідності проводити ручну перекличку, економлячи навчальний час. Всі ключові інтерфейси взаємодії, включаючи мобільну версію бота, продемонстровані на скріншотах (див. Додаток Д).

ВИСНОВКИ

У кваліфікаційній роботі успішно розв'язано комплексне завдання розробки навчальної вебплатформи з інтегрованим інтелектуальним комунікаційним ядром. У ході виконання дослідження було проведено ґрунтовний аналіз предметної області, обрано оптимальний технологічний стек та реалізовано повноцінний програмний продукт, готовий до практичної експлуатації в умовах реального освітнього процесу. За результатами проєктування, програмування та тестування сформульовано такі висновки:

1. Спроєктовано та розгорнуто нормалізовану реляційну базу даних на основі СУБД PostgreSQL. На рівні фізичної схеми закладено жорсткі унікальні обмеження та каскадні правила, які гарантують транзакційну цілісність даних і математично унеможливлюють появу аномалій. Така структура створює надійний фундамент для зберігання великих масивів академічної інформації.
2. Серверну частину системи реалізовано за архітектурним патерном «модульний моноліт» на базі мікрофреймворку FastAPI. Замість класичного збереження стану сесії на сервері, що неминуче перевантажує апаратні ресурси при зростанні контингенту, імплементовано Stateless-автентифікацію через механізм JWT-токенів. Це концептуальне рішення значно підвищило стійкість платформи до високонавантажених запитів та забезпечило її готовність до безболісного горизонтального масштабування.
3. Комунікаційне ядро платформи успішно інтегровано з великою мовною моделлю Google Gemini API та обраним месенджером. Завдяки розробленим жорстким інструкціям алгоритм працює як високоточний розумний маршрутизатор: він самостійно генерує релевантні відповіді на типові організаційні питання здобувачів освіти, а нестандартні запити миттєво ескалює до ізольованого робочого кабінету викладача. Мережеві запити до хмарного ШІ інкапсульовано у фонових задачах, що

гарантує відсутність блокування основного потоку сервера. Впровадження цього модуля дозволяє суттєво знизити рутинне інформаційне навантаження на науково-педагогічних працівників.

4. Клієнтський рівень розроблено за концепцією Thin Client з усвідомленою відмовою від використання масивних реактивних фреймворків, натомість застосовано оптимізовані інструменти Vanilla JS та HTML5. Це архітектурне рішення забезпечило практично миттєве завантаження графічного інтерфейсу та високу чуйність платформи навіть за умов нестабільного інтернет-з'єднання, що є критично важливим фактором доступності для користувачів мобільних пристроїв.
5. Проведене комплексне функціональне тестування підтвердило стабільність обробки транзакцій та стійкість системи до непередбачуваних дій користувачів. Механізми рольової моделі доступу (RBAC) надійно блокують будь-які спроби несанкціонованого доступу до адміністративних маршрутів, а алгоритми перевірки даних на рівні бекенду запобігають нелегітимній модифікації академічних результатів чи підміні файлів.

Підсумовуючи, можна стверджувати, що розроблена інформаційна система повністю відповідає поставленим на початку роботи технічним вимогам. Її впровадження дозволяє автоматизувати ключові освітні процеси, підвищити прозорість обліку відвідуваності та створити сучасне інтерактивне середовище для взаємодії між викладачами та студентами. Подальший розвиток проєкту може бути спрямований на розширення функціоналу адміністративної панелі шляхом додавання модуля предиктивної аналітики на основі зібраних даних. Мета кваліфікаційної роботи досягнута в повному обсязі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Вакула Н. О. Цифровізація та її вплив на освітній простір у контексті формування ключових компетентностей [Електронний ресурс] / Н. О. Вакула // На Урок : освітній журнал. – Режим доступу: https://naurok.com.ua/ (дата звернення: 22.03.2026).

2. Підвищення якості вищої освіти за допомогою цифрових технологій та дистанційного навчання для здобувачів вищої освіти в Україні [Електронний ресурс] // Академічні візії : електронне наукове фахове видання. – Режим доступу: https://academy-vision.org/ (дата звернення: 22.03.2026).

3. Perception of Learning Management System (LMS) on the Academic Performance of Undergraduate Students [Electronic resource] // ERIC : Institute of Education Sciences. – Mode of access: https://eric.ed.gov/ (date of access: 22.03.2026).

4. Exploring the Impact of Learning Management System (LMS) [Electronic resource] // IEEE Xplore Digital Library. – Mode of access: https://ieeexplore.ieee.org/ (date of access: 22.03.2026).

5. Understanding the Impact of a Learning Management System Using a Novel Modified DeLone and McLean Model [Electronic resource] // MDPI : Open Access Journals. – Mode of access: https://www.mdpi.com/ (date of access: 22.03.2026).

6. Securing Academic Social Platforms: Implementing Role-Based Access Control (RBAC) in University-Based Digital Systems [Electronic resource] // IJRTI. – Mode of access: https://www.ijrti.org/ (date of access: 22.03.2026).

7. Role-based Access Control in Educational Administration System [Electronic resource] // MATEC Web of Conferences. – Mode of access: https://www.matec-conferences.org/ (date of access: 22.03.2026).

8. Role Based Access Control (RBAC) [Electronic resource] // NIST Computer Security Resource Center. – Mode of access: https://csrc.nist.gov/ (date of access: 22.03.2026).

9. Modern LMS vs Traditional LMS: The Guide [Electronic resource] // Disco Learning Platform. – Mode of access: https://www.disco.co/ (date of access: 22.03.2026).

10. Використання месенджерів як засобу підтримки навчального процесу [Електронний ресурс] // Електронна бібліотека ІІТЗН НАПН України. – Режим доступу: https://lib.iitta.gov.ua/ (дата звернення: 22.03.2026).

11. The Problem of Content Overload in Higher Education-and How AI Solves It [Electronic resource] // Tribe AI. – Mode of access: https://www.tribe.ai/ (date of access: 22.03.2026).

12. Causal Modeling of Academic Activity and Study Process Management [Electronic resource] // MDPI. – Mode of access: https://www.mdpi.com/ (date of access: 22.03.2026).

13. Optimization of Business Processes Through BPM Methodology [Electronic resource] // MDPI. – Mode of access: https://www.mdpi.com/ (date of access: 22.03.2026).

14. Ontological analysis of business process modeling of higher education institutions [Electronic resource] // ResearchGate. – Mode of access: https://www.researchgate.net/ (date of access: 22.03.2026).

15. AI-Powered Automatic Question Generation for Teachers [Electronic resource] // ResearchGate. – Mode of access: https://www.researchgate.net/ (date of access: 22.03.2026).

16. AI in the Classroom: Insights from Educators on Usage, Challenges [Electronic resource] // MDPI. – Mode of access: https://www.mdpi.com/ (date of access: 22.03.2026).

17. Dealing with information overload: a comprehensive review [Electronic resource] // National Institutes of Health (NIH). – Mode of access: https://pubmed.ncbi.nlm.nih.gov/ (date of access: 22.03.2026).

18. Performance Comparison of Monolithic and Microservices Architectures in Handling High-Volume Transactions [Electronic resource] // ResearchGate. – Mode of access: https://www.researchgate.net/ (date of access: 22.03.2026).
19. Monolithic vs Microservices - Difference Between Software Development Architectures [Electronic resource] // Amazon Web Services (AWS) Documentation. – Mode of access: https://aws.amazon.com/ (date of access: 22.03.2026).
20. Custom LMS vs. Off-the-Shelf: Choose the Best [Electronic resource] // Enfin Technologies. – Mode of access: https://www.enfintechnologies.com/ (date of access: 22.03.2026).
21. Principles of Database Systems [Електронний ресурс] / J. D. Ullman. – Режим доступу: https://lsv.ens-paris-saclay.fr/~goubault/BD/ullman-principles-of-database-and-knowledge-base-systems-volume-1.pdf (дата звернення: 23.03.2026).
22. Design Theory for Relational Databases: Functional Dependencies and Normalization [Електронний ресурс]. – Режим доступу: https://www.eng.utah.edu/~cs5530/Lectures/fds.pdf (дата звернення: 23.03.2026).
23. Normalization - Chapter 7: Relational Database Design [Електронний ресурс]. – Режим доступу: https://www.db-book.com/slides-dir/PDF-dir/ch7.pdf (дата звернення: 23.03.2026).
24. What is Data Integrity | Issues & Types Explained [Електронний ресурс] // Imperva. – Режим доступу: https://www.imperva.com/learn/data-security/data-integrity/ (дата звернення: 23.03.2026).
25. Effective analysis and classification of UML class diagrams [Електронний ресурс]. – Режим доступу: https://pubs.aip.org/aip/acp/article-abstract/3393/1/060011/3381619/Effective-analysis-and-classification-of-UML-class?redirectedFrom=fulltext (дата звернення: 23.03.2026).
26. Chaudron M. R. V., Heijstek W., Nugroho A. How effective is UML modeling? An empirical perspective on costs and benefits [Електронний

ресурс] // Software and Systems Modeling. – 2012. – Vol. 11, Is. 4. – P. 571–580. – Режим доступу: https://doi.org/10.1007/s10270-012-0278-4 (дата звернення: 27.03.2026).

27. Design Patterns [Електронний ресурс] // Refactoring.Guru. – Режим доступу: https://refactoring.guru/design-patterns (дата звернення: 23.03.2026).

28. Design Patterns for Modern Backend Development [Електронний ресурс] // GeeksforGeeks. – Режим доступу: https://www.geeksforgeeks.org/system-design/design-patterns-for-modern-backend-development/ (дата звернення: 23.03.2026).

29. Understanding REST API Architectural Styles and Design Patterns [Електронний ресурс]. – Режим доступу: https://xapihub.io/resources/blog/understanding-rest-api-architectural-styles-and-design-patterns/ (дата звернення: 23.03.2026).

30. RESTful API design principles: best practices [Електронний ресурс] // Upsun. – Режим доступу: https://upsun.com/blog/restful-api-design-principles/ (дата звернення: 23.03.2026).

31. Why We Chose a Modular Monolith Over Microservices [Електронний ресурс] // Medium. – Режим доступу: https://medium.com/@afandy.lamusu/why-we-chose-a-modular-monolith-over-microservices-and-you-should-too-d46b8921eea9 (дата звернення: 23.03.2026).

32. Monolithic vs Microservices - Difference Between Software Development Architectures [Електронний ресурс] // AWS. – Режим доступу: https://aws.amazon.com/compare/the-difference-between-monolithic-and-microservices-architecture/ (дата звернення: 23.03.2026).

33. UML Sequence Diagram Tutorial [Електронний ресурс] // Lucidchart. – Режим доступу: https://www.lucidchart.com/pages/uml-sequence-diagram (дата звернення: 23.03.2026).

34. LLM-Based Routing in Mixture of Experts: A Novel Framework [Електронний ресурс] // arXiv. – Режим доступу: https://arxiv.org/html/2501.09636v2 (дата звернення: 23.03.2026).
35. Asynchronous Tasks With Django and Celery [Електронний ресурс] // Real Python. – Режим доступу: https://realpython.com/asynchronous-tasks-with-django-and-celery/ (дата звернення: 23.03.2026).
36. Benchmarking the performance of Python web frameworks [Електронний ресурс]. – Режим доступу: https://ph.pollub.pl/index.php/jcsi/article/download/7738/5148/36459 (дата звернення: 23.03.2026).
37. Flask vs. FastAPI: A Deep Dive into Synchronous and Asynchronous Web Frameworks [Електронний ресурс] // Medium. – Режим доступу: https://medium.com/@gsaidheeraj/flask-vs-fastapi-a-deep-dive-into-synchronous-and-asynchronous-python-web-frameworks-9cf4b62caff8 (дата звернення: 23.03.2026).
38. Why I use Vanilla JavaScript instead of a framework [Електронний ресурс] // Medium. – Режим доступу: https://medium.com/gitconnected/why-i-use-vanilla-javascript-instead-of-a-framework-6fb1a6a166f2 (дата звернення: 23.03.2026).
39. FastAPI: High performance, easy to learn, fast to code, ready for production [Електронний ресурс] // Official Documentation. – Режим доступу: https://fastapi.tiangolo.com/ (дата звернення: 23.03.2026).
40. MDN Web Docs: Web technology for developers [Електронний ресурс] // Mozilla Foundation. – Режим доступу: https://developer.mozilla.org/ (дата звернення: 23.03.2026).
41. Aiogram 3.x Official Documentation [Електронний ресурс]. – Режим доступу: https://docs.aiogram.dev/ (дата звернення: 23.03.2026).
42. Google Gemini API Documentation: Build with AI [Електронний ресурс] // Google for Developers. – Режим доступу: https://ai.google.dev/docs/ (дата звернення: 23.03.2026).

43. PostgreSQL: The World's Most Advanced Open Source Relational Database [Електронний ресурс] // Official Documentation. – Режим доступу: https://www.postgresql.org/docs/ (дата звернення: 23.03.2026).
44. SQLAlchemy Documentation: The Database Toolkit for Python [Електронний ресурс]. – Режим доступу: https://docs.sqlalchemy.org/ (дата звернення: 23.03.2026).

ДОДАТОК А

Посилання на вихідний код проєкту

Повний вихідний код розробленої навчальної платформи (backend та frontend частини) розміщено у системі контролю версій на платформі GitHub.

Доступ до репозиторію за посиланням:

https://github.com/VeronicaKondratenko/dp-2026

ДОДАТОК Б

Фрагменти вихідного коду

Файл 1: models.py (Опис структури бази даних та ORM-моделей)

```
import enum

from sqlalchemy import (
    Column, Integer, String, Boolean, ForeignKey, Table, Enum,
    DateTime, Time, Date, func, BigInteger, UniqueConstraint
)
from sqlalchemy.orm import relationship, validates
from .database import Base



class UserRole(str, enum.Enum):
    student = "student"
    teacher = "teacher"
    admin = "admin"



student_group_association = Table(
    "student_group_association",
    Base.metadata,
    Column("student_id", Integer, ForeignKey("users.id",
ondelete="CASCADE")),
    Column("group_id", Integer, ForeignKey("study_groups.id",
ondelete="CASCADE"))
)
```

```
class User(Base):
    __tablename__ = "users"

    id = Column(Integer, primary_key=True)
    email = Column(String, unique=True, index=True)
    password_hash = Column(String)
    full_name = Column(String)
    role = Column(Enum(UserRole), default=UserRole.student)

    groups = relationship("StudyGroup", secondary=student_group_association,
back_populates="students")
    submissions = relationship("AssignmentSubmission",
back_populates="student")


class Course(Base):
    __tablename__ = "courses"

    id = Column(Integer, primary_key=True)
    title = Column(String)
    description = Column(String)

    disciplines = relationship("Discipline", back_populates="course")


class AssignmentSubmission(Base):
    __tablename__ = "assignment_submissions"

    id = Column(Integer, primary_key=True)
    student_id = Column(Integer, ForeignKey("users.id",
```

```
ondelete="CASCADE"),  nullable=False)
    assignment_id = Column(Integer, ForeignKey("assignments.id",
ondelete="CASCADE"), nullable=False)
    content = Column(String, nullable=True)
    file_name = Column(String, nullable=True)
    is_locked = Column(Boolean, default=False, nullable=False)
    grade_score = Column(Integer, nullable=True)

    student = relationship("User", back_populates="submissions")
    assignment = relationship("Assignment", back_populates="submissions")



class Schedule(Base):
    __tablename___ = "schedules"

    id = Column(Integer, primary_key=True)
    date = Column(Date)
    time = Column(Time)
    end_time = Column(Time, nullable=True)
    group_id = Column(Integer, ForeignKey("study_groups.id"))
    teacher_id = Column(Integer, ForeignKey("users.id"))

    group = relationship("StudyGroup")
    teacher = relationship("User")
```

Файл 2: assignments.py (Бізнес-логіка перевірки доступу та оцінювання)

```
from fastapi import APIRouter, Depends, HTTPException
from sqlalchemy import select
from sqlalchemy.ext.asyncio import AsyncSession
from sqlalchemy.orm import selectinload
```

```
from ...database import get_db
from ...models import User, Assignment, StudyGroup, UserRole
from ...auth import get_current_user


router = APIRouter()


@router.get("/")
async def get_assignments(
    db: AsyncSession = Depends(get_db),
    current_user: User = Depends(get_current_user)
):
    """Role-based assignment retrieval logic"""

    if current_user.role == UserRole.student:
        student_result = await db.execute(
            select(User)
            .where(User.id == current_user.id)
            .options(selectinload(User.groups).selectinload(StudyGroup.courses))
        )
        student = student_result.scalar_one()

        course_ids = set()
        for group in student.groups:
            for course in group.courses:
                course_ids.add(course.id)

        if not course_ids:
```

```
            return []

        # Запити до бази даних для отримання дисциплін, тем та завдань
    студента
        return await fetch_student_assignments(db, course_ids, current_user.id)

    elif current_user.role == UserRole.teacher:
        groups_result = await db.execute(
            select(StudyGroup)
            .where(StudyGroup.teacher_id == current_user.id)
            .options(selectinload(StudyGroup.courses))
        )
        groups = groups_result.scalars().all()

        # Запити для отримання всіх завдань для викладача
        return await fetch_teacher_assignments(db, groups)
```

Файл 3: schedule.py (Алгоритм виявлення накладок у розкладі)

```
import datetime

from fastapi import APIRouter, Depends, HTTPException
from sqlalchemy.ext.asyncio import AsyncSession
from sqlalchemy.future import select

from ...database import get_db
from ...models import Schedule


router = APIRouter()
```

```
def _overlaps(start_a, end_a, start_b, end_b) -> bool:
    return start_a < end_b and start_b < end_a



@router.post("/")
async def create_schedule(schedule, db: AsyncSession = Depends(get_db)):
    """Create schedule with comprehensive conflict detection"""

    requested_start_dt = datetime.datetime.combine(schedule.date, schedule.time)
    requested_end_dt = datetime.datetime.combine(schedule.date,
schedule.end_time)

    existing_result = await db.execute(
        select(Schedule)
        .where(
            Schedule.date == schedule.date,
            (Schedule.group_id == schedule.group_id) | (Schedule.teacher_id ==
schedule.teacher_id),
        )
    )
    existing_schedules = existing_result.scalars().all()

    for existing in existing_schedules:
        existing_start_dt = datetime.datetime.combine(existing.date, existing.time)
        existing_end_dt = datetime.datetime.combine(existing.date,
existing.end_time)

        if _overlaps(requested_start_dt, requested_end_dt, existing_start_dt,
existing_end_dt):
```

```
            conflict_parts = []
            if existing.group_id == schedule.group_id:
                conflict_parts.append("same group")
            if existing.teacher_id == schedule.teacher_id:
                conflict_parts.append("same teacher")

            detail = f"Schedule overlap ({', '.join(conflict_parts)}) on
{existing.date.isoformat()}"
            raise HTTPException(status_code=400, detail=detail)

    db_schedule = Schedule(**schedule.dict())
    db.add(db_schedule)
    await db.commit()

    return db_schedule
```

ДОДАТОК В

Архітектурні UML-діаграми системи

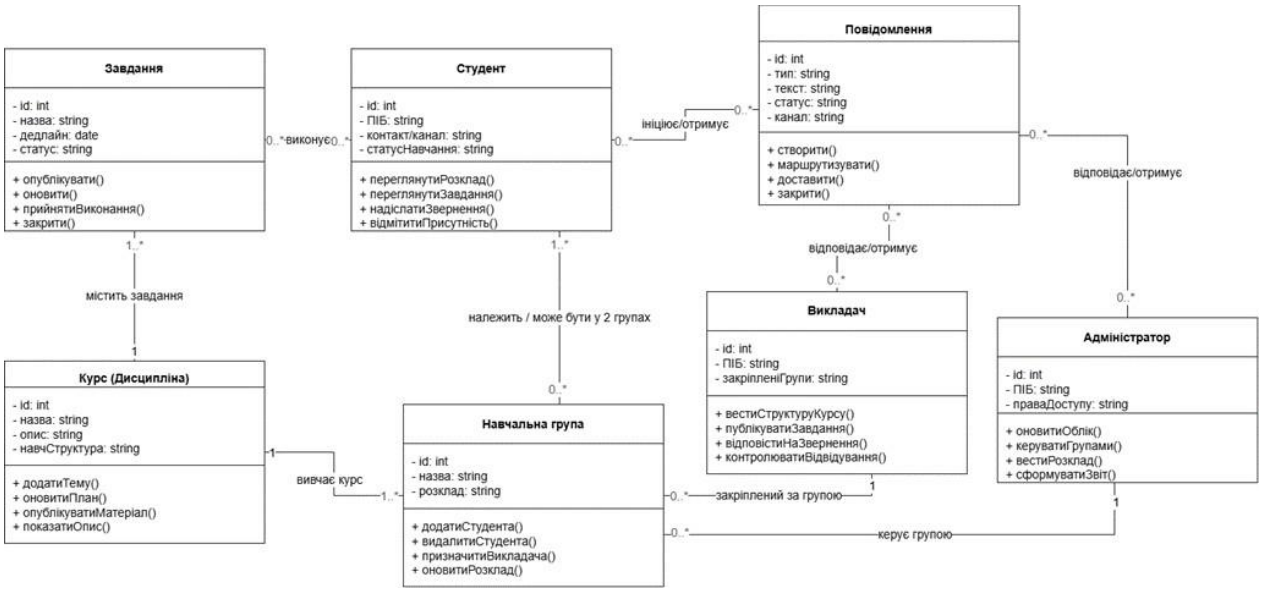


Рис. В.1 – Діаграма класів

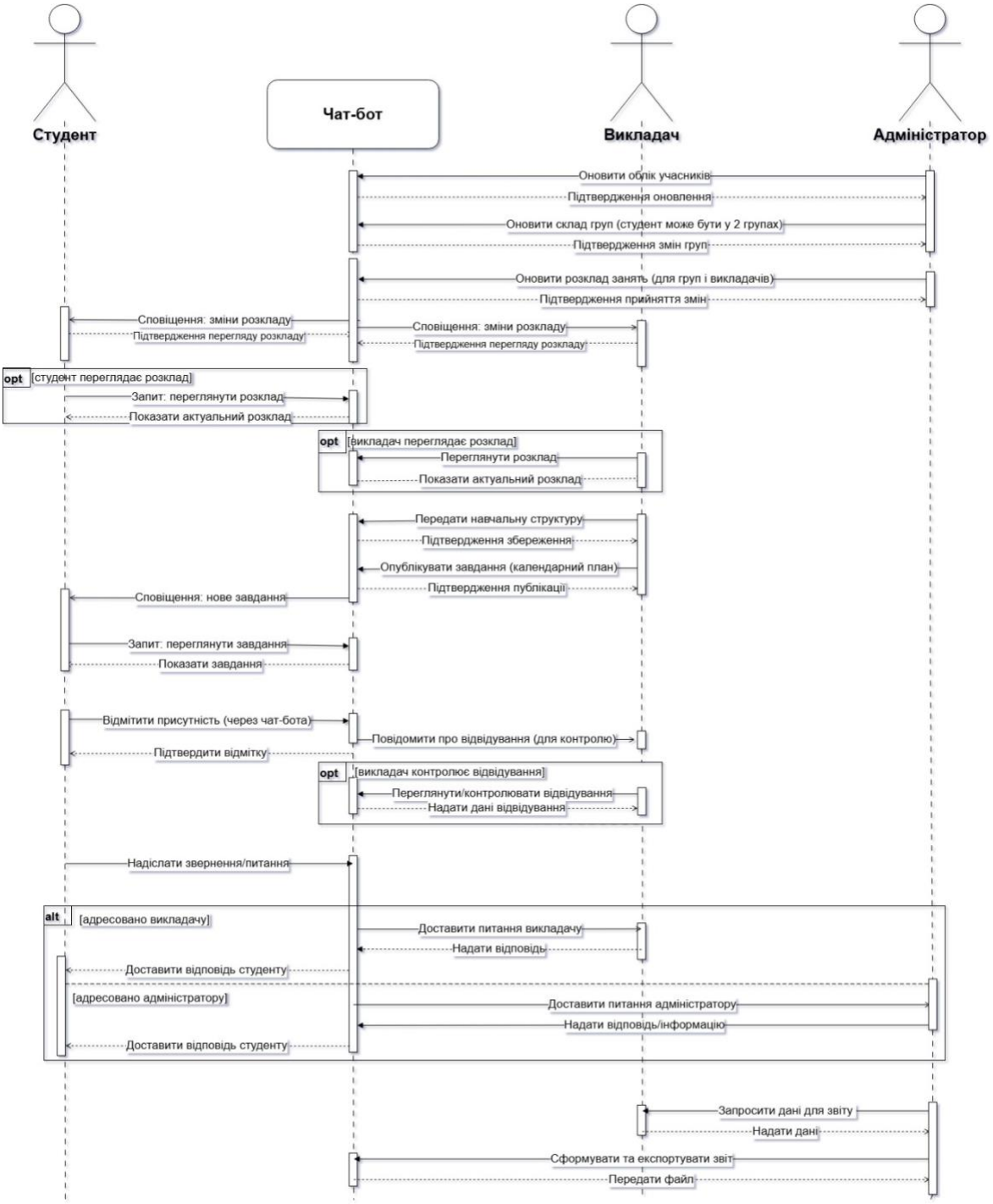
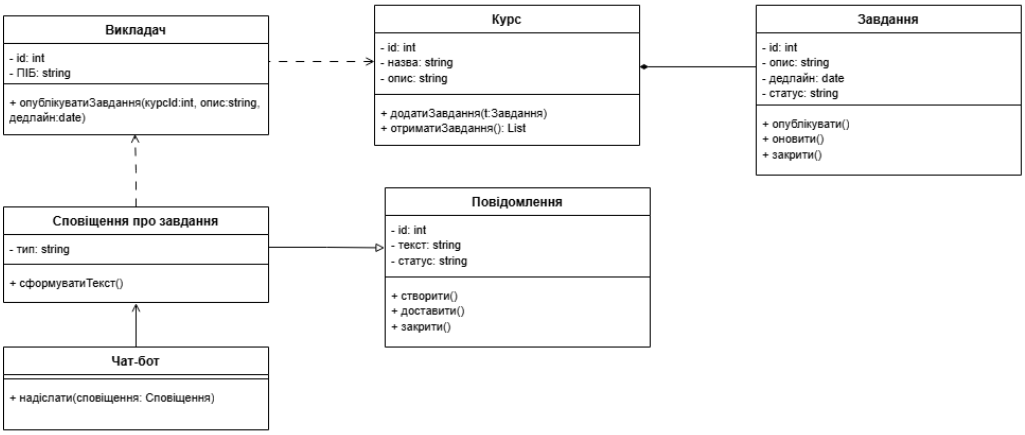
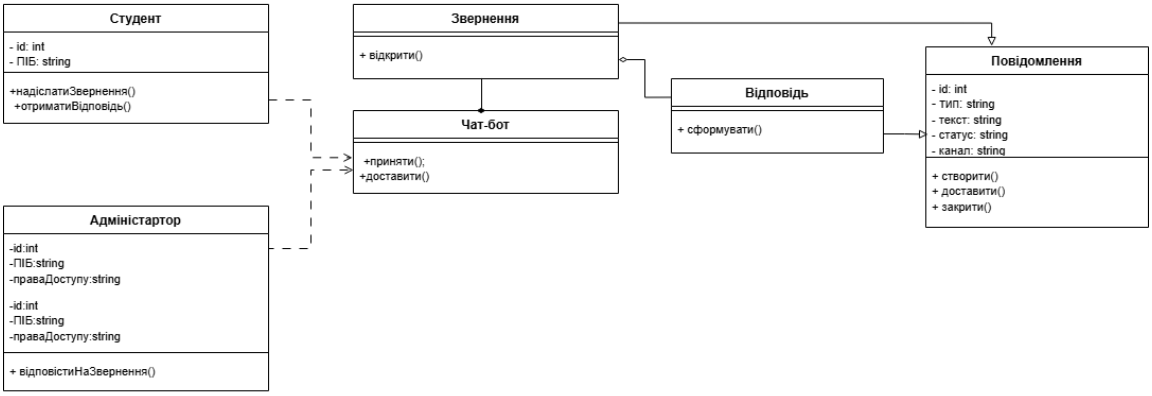


Рис. В.2 – Діаграма послідовності

Кооперація «Публікація завдання та сповіщення»



Кооперація «Адміністратор відповідає на звернення»



Кооперація «Фіксація відвідування через чат-бот»

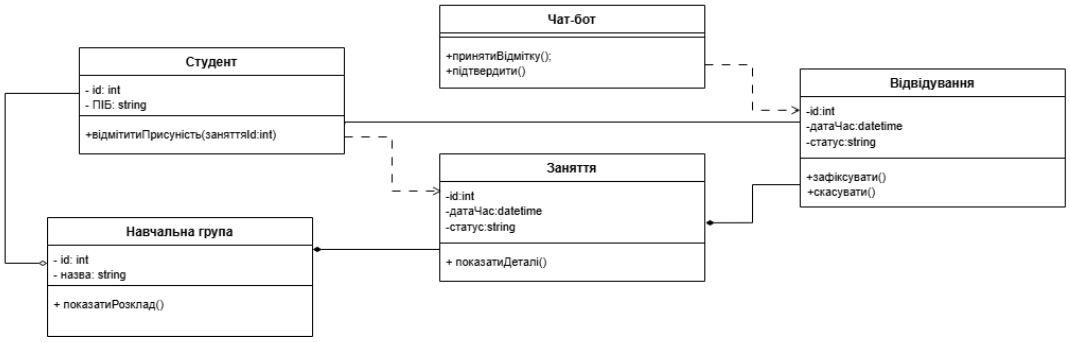


Рис. В.3 – Моделі кооперацій класів для ключових бізнес-процесів платформи

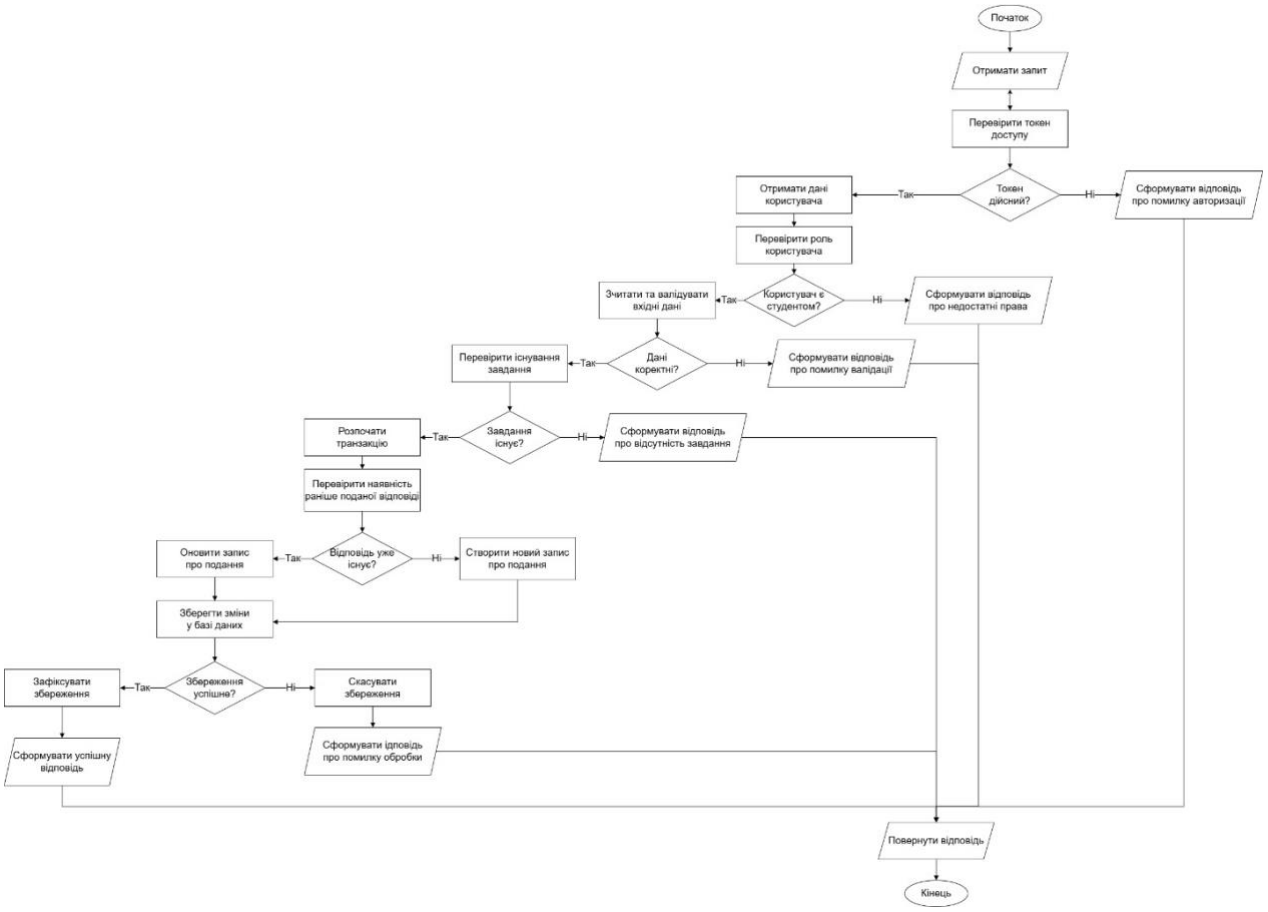


Рис. В.4 - Блок-схема алгоритму здачі завдання

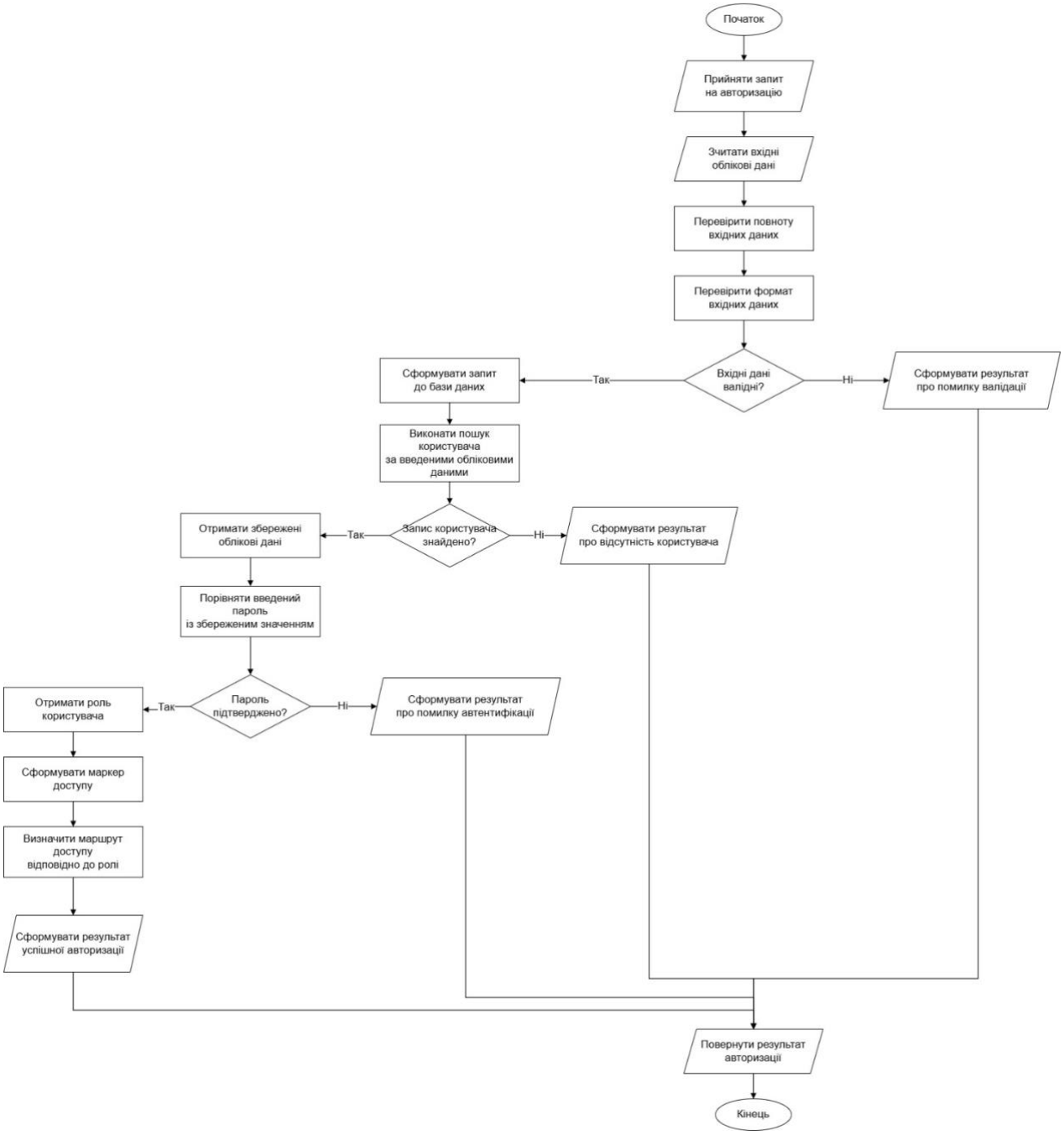


Рис. В.5 – Схема алгоритму авторизації користувача

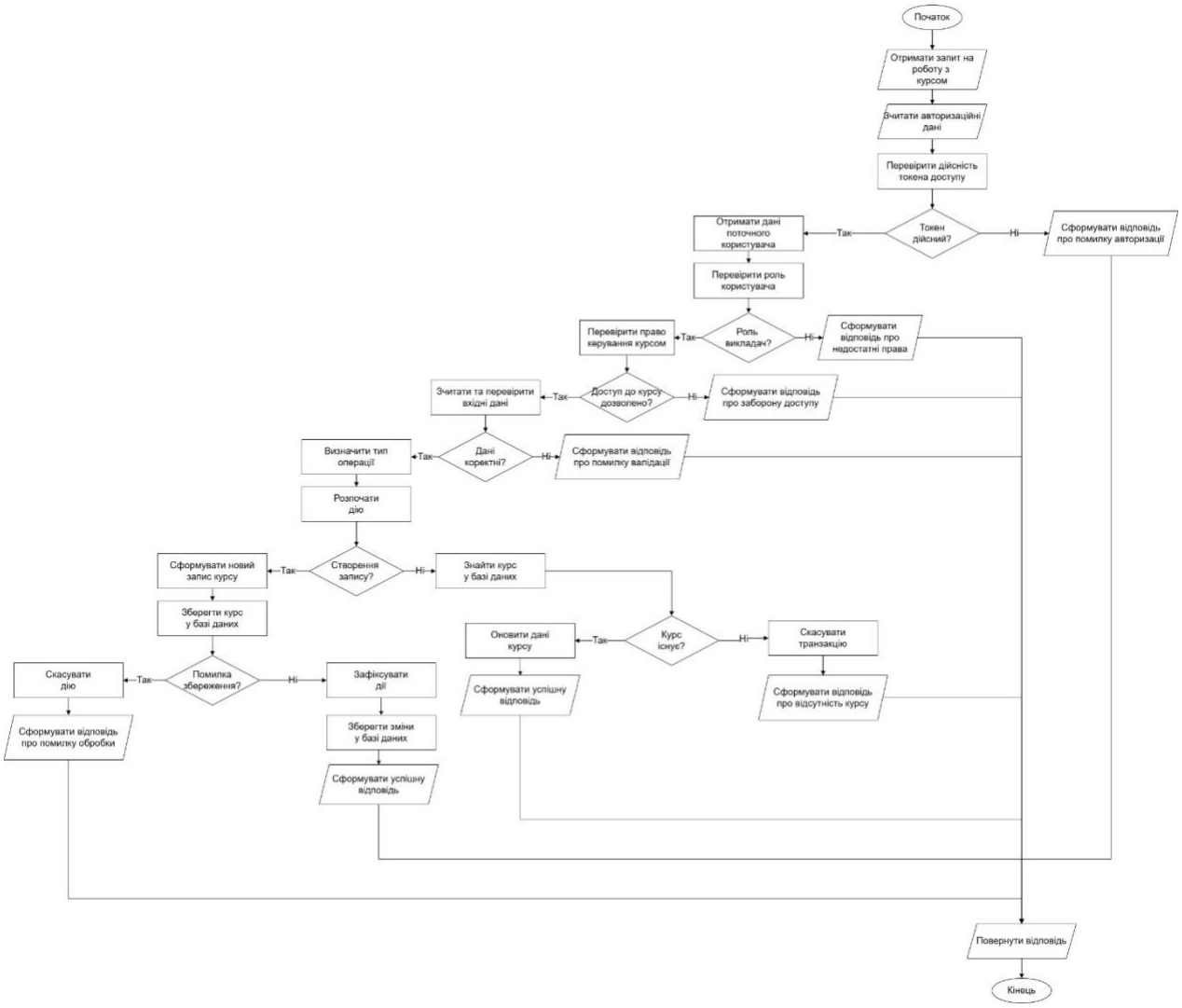


Рис. В.6 – Блок-схема алгоритму керування курсом викладачем

ДОДАТОК Д

Інтерфейси користувача та месенджер-бота

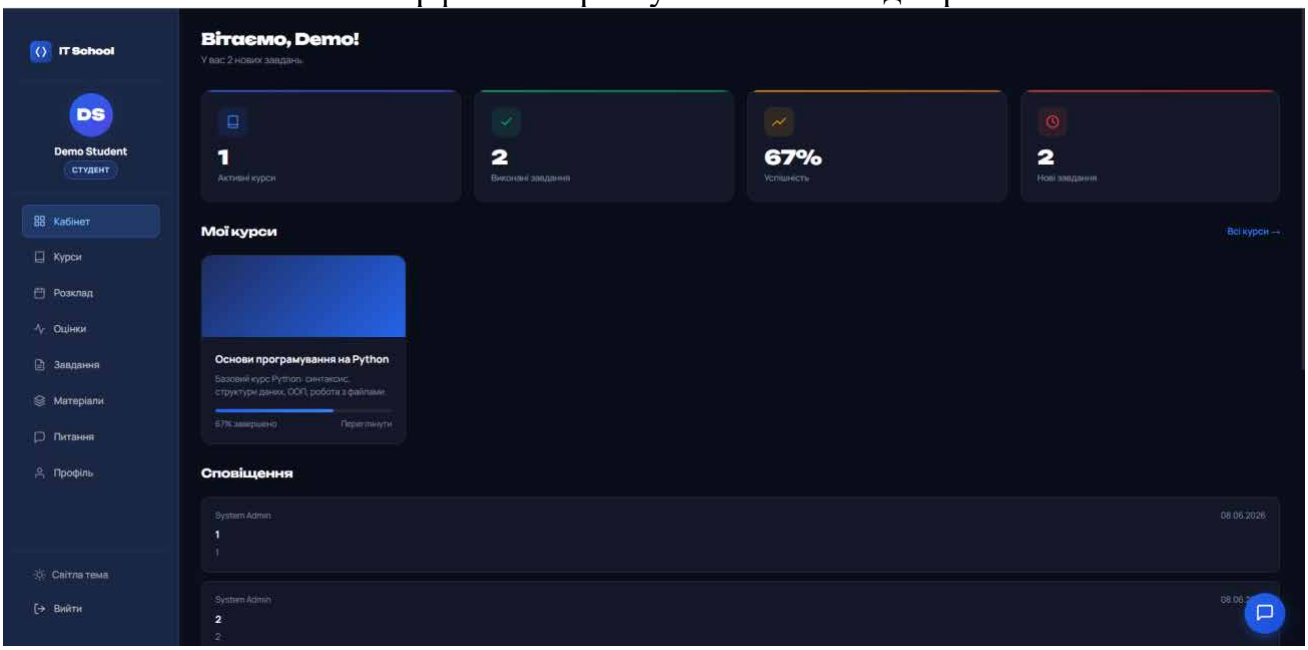


Рис. Д.1 – Інтерфейс робочого кабінету студента

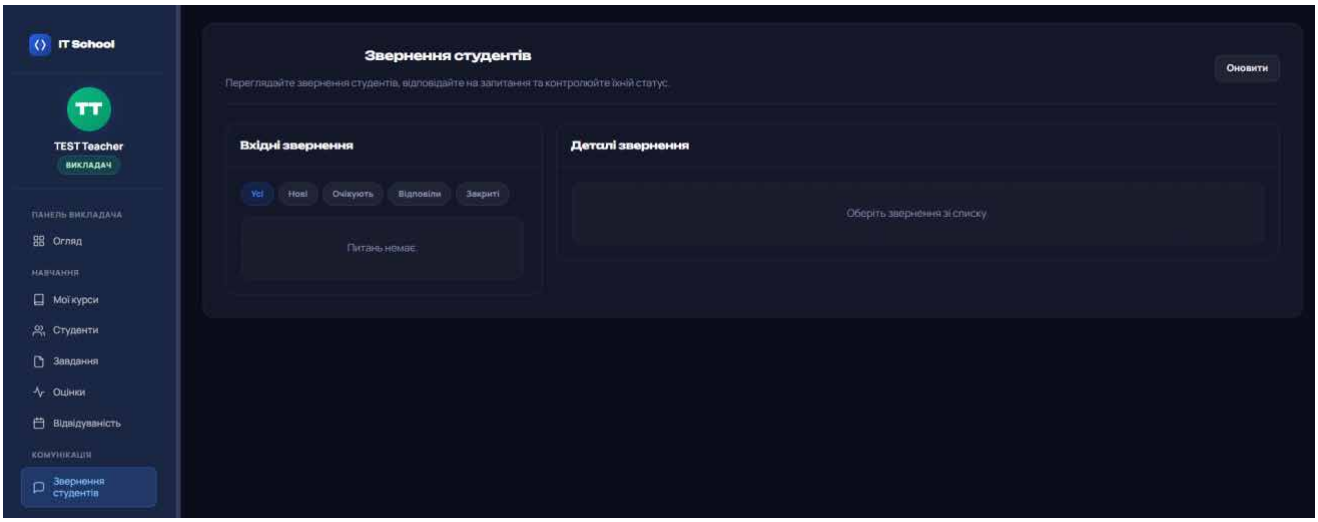


Рис. Д.2 – Панель вхідних повідомлень викладача

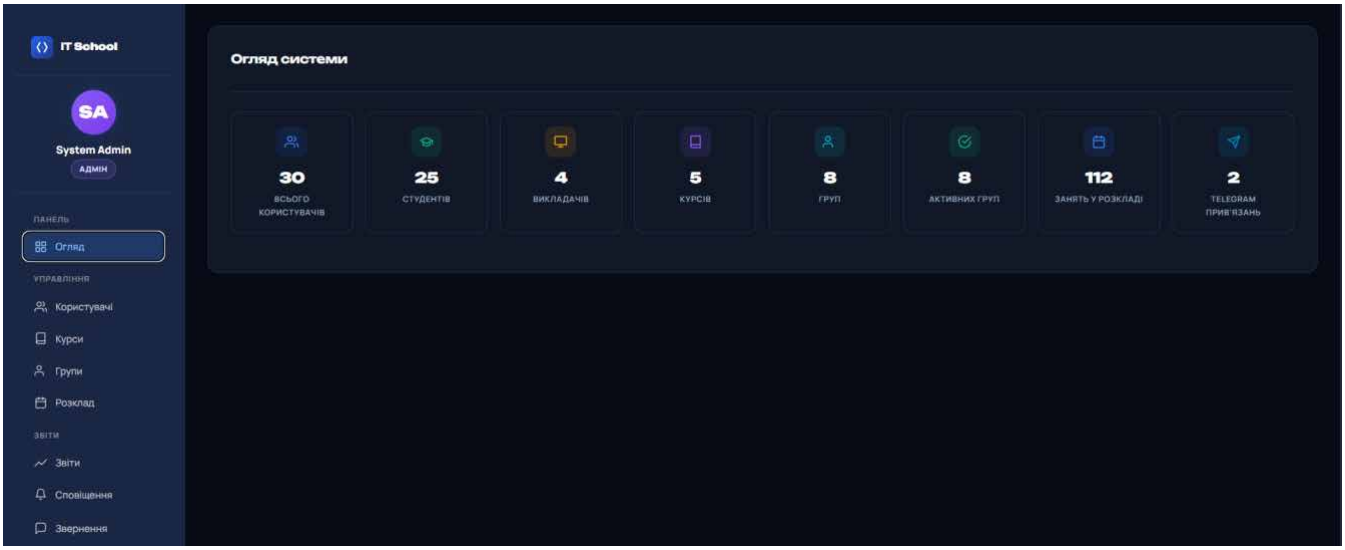


Рис. Д.3 – Інтерфейс адміністративної панелі

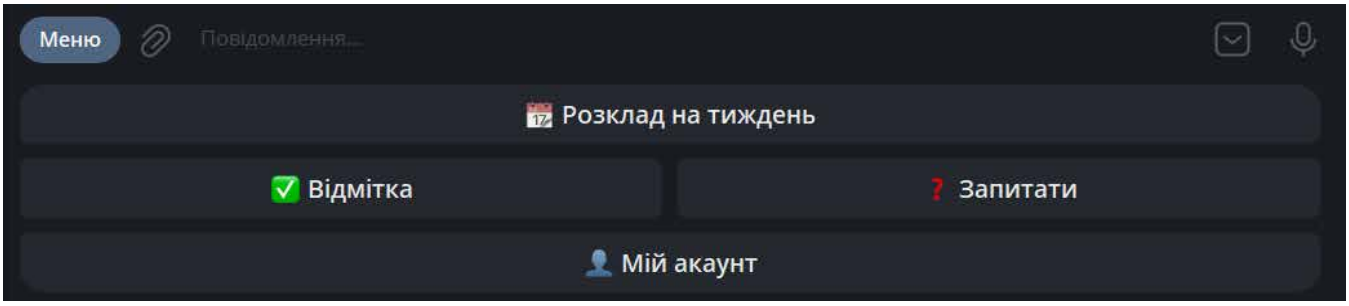


Рис. Д.4 – Інтерфейс взаємодії з месенджер-ботом

ДОДАТОК Е

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій
Декларація про академічну доброчесність та використання ШІ

Я, Кондратенко Вероніка Дмитрівна, здобувачка НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

1. Моя бакалаврська кваліфікаційна робота на тему «Програмне забезпечення системи керування процесом розповсюдження інформації серед учасників навчального процесу» є результатом моєї особистої праці.

2. Усі запозичення з текстів інших авторів мають відповідні посилання згідно з чинними стандартами.

3. Щодо використання інструментів ШІ зазначаю, що (обрати необхідне):

o [ ] інструменти генеративного ШІ не використовувалися.

o [ ] інструменти ШІ (вказати назву: ChatGPT, **Gemini**, Claude, DeepL, _____інші (вказати назву)) були використані для:

- [✓] перекладу іншомовних джерел;
- [ ] стилістичного редагування та перевірки граматики;
- [ ] допомоги у написанні/відлагодженні програмного коду;
- [ ] інше: _____.

Я розумію, що надання недостовірної інформації може призвести до скасування результатів захисту та відрахування з числа здобувачів НУБіП України.

«___»_____2026 р. _____ **Вероніка**

(підпис) **КОНДРАТЕНКО**

Календарний план

№ з/п	Назва етапів виконання бакалаврської кваліфікаційної роботи	Строк виконання етапів бакалаврської кваліфікаційної роботи	Примітка
1	Отримання та уточнення теми бакалаврської кваліфікаційної роботи	19.12.2025 – 26.12.2025	Виконано
2	Аналіз предметної області та вимог до навчальної вебплатформи	27.12.2025 – 26.01.2026	Виконано
3	Огляд існуючих навчальних систем та формування постановки завдання	27.01.2026 – 09.02.2026	Виконано
4	Проєктування логічної моделі бази даних та UML-моделей системи	10.02.2026 – 12.03.2026	Виконано
5	Вибір технологічного стеку та архітектури програмного забезпечення	13.03.2026 – 24.03.2026	Виконано
6	Розробка серверної частини, клієнтського рівня, чат-бота та AI-модуля	25.03.2026 – 03.05.2026	Виконано
7	Функціональне тестування, перевірка доступу та захисту даних	04.05.2026 – 12.05.2026	Виконано
8	Оформлення пояснювальної записки, додатків і презентаційних матеріалів	13.05.2026 – 22.05.2026	Виконано

Керівник бакалаврської кваліфікаційної роботи _____ (підпис) **Віктор КИРИЧЕНКО**

Студентка _____ (підпис) **Вероніка КОНДРАТЕНКО**

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

КОНДРАТЕНКО ВЕРОНІКИ ДМИТРІВНИ

Наказ НУБіП України №3204 «С» від 19.12.2025