Міністерство освіти і науки України
Рівненський державний гуманітарний університет
Кафедра інформаційних технологій та моделювання

Кваліфікаційна робота

за освітнім ступенем «магістр»

на тему:

ВЕБ-ЗАСТОСУНОК ДЛЯ СТВОРЕННЯ ТА ПЕРЕВІРКИ ДОМАШНІХ ЗАВДАНЬ

Виконала:
здобувачка 2 курсу
групи М-КН-21
спеціальності 122 «Комп'ютерні науки»
Шеремет Олена Петрівна

Науковий керівник:
ст. викл. Кирик Т. А.

Рівне – 2025

ЗМІСТ

Список використаних скорочень

AJAX	Asynchronous JavaScript And XML
CSS	Cascading Style Sheets (мова стилів)
HTML	HyperText Markup Language(мова розмітки)
JS	JavaScript
JSON	JavaScript Object Notation
JWT	JSON Web Token
MITM	«Man-In-The-Middle» (атака «людина посередині»)
XML	Extensible Markup Language (розширювана мова розмітки)
БД	База даних
ШІ	Штучний інтелект

ВСТУП

Актуальність теми. У сучасному світі технології займають важливе місце в освітньому процесі, зокрема в організації навчання та оцінювання знань учнів. Розвиток інформаційних технологій та їх впровадження в освітній процес змінили традиційні методи навчання, зробивши їх більш інтерактивними та доступними. Вебзастосунки стали невід'ємною частиною навчання, адже вони дозволяють автоматизувати процес створення та перевірки домашніх завдань, спрощують комунікацію між учнями та викладачами, а також забезпечують зручний доступ до навчальних матеріалів.

Однією з основних переваг використання вебзастосунків є їх здатність оптимізувати час, витрачений на виконання адміністративних завдань. Викладачі можуть легко створювати завдання, формувати критерії оцінювання, а також здійснювати автоматизовану перевірку робіт. Це звільняє час, який вони можуть витратити на більш важливі аспекти навчального процесу, такі як індивідуальна робота з учнями, надання консультацій та активна участь у навчальному процесі.

Крім того, вебзастосунки надають можливість створювати бази даних із завданнями, що дає змогу викладачам легко оновлювати матеріали, адаптувати їх до потреб конкретного класу чи учня. Учні, у свою чергу, мають доступ до завдань у будь-який час та з будь-якого пристрою, що дозволяє їм самостійно організовувати свій час і темп навчання. Цей підхід сприяє розвитку самостійності та відповідальності у учнів, оскільки вони можуть самостійно планувати свої заняття.

Вебзастосунки також мають великий потенціал для інтеграції новітніх технологій у навчальний процес. Вони можуть включати елементи гейміфікації, що підвищує мотивацію учнів. Ігрові механіки, такі як бали, нагороди за виконання завдань або лідерські таблиці, можуть стимулювати учнів до активної участі у навчальному процесі. Це, в свою чергу, сприяє покращенню результатів навчання.

У контексті глобалізації та дистанційного навчання, вебзастосунки стають ще більш актуальними. Уроки та завдання можуть бути доступні для учнів з різних регіонів та країн, що відкриває нові можливості для обміну досвідом та знаннями. Це особливо важливо в умовах, коли навчання часто проводиться в онлайн-форматі.

Таким чином, розробка вебзастосунку для створення та перевірки домашніх завдань є надзвичайно актуальною, оскільки він може суттєво підвищити ефективність навчального процесу та забезпечити індивідуальний підхід до кожного учня. Використання новітніх технологій у навчанні дозволяє покращити якість освіти, робить її більш доступною та зручною для всіх учасників процесу. У цьому контексті важливо не лише розробити функціональний вебзастосунок, а й забезпечити його відповідність сучасним вимогам, щоб він був зручним у використанні та відповідав потребам як викладачів, так і учнів.

Мета роботи полягає у розробці вебзастосунку, який дозволяє створювати, зберігати та перевіряти домашні завдання, а також аналізувати результати виконання завдань учнями. Враховуючи швидкий розвиток технологій та зміну освітніх парадигм, важливо створити інструмент, який спростить процес навчання як для викладачів, так і для учнів. Вебзастосунок має стати ефективним засобом для автоматизації рутинних завдань, покращення взаємодії між учасниками навчального процесу та надання можливості для аналізу результатів, що дозволить коригувати навчальний процес відповідно до потреб учнів.

Для досягнення сформульованої мети було поставлено такі завдання:

Аналіз сучасних методів створення та перевірки домашніх завдань. На першому етапі дослідження провести огляд існуючих вебзастосунків та інструментів, що використовуються в навчанні. Включити аналіз їх функціональних можливостей, зручності використання, а також переваг і недоліків. Визначення кращих практик дозволить створити основи для розробки нового вебзастосунку, який відповідатиме сучасним вимогам.

Визначення основних функціональних вимог до вебзастосунку. Наступний крок – сформулювати вимоги до функціоналу вебзастосунку, враховуючи потреби

користувачів. Визначити ключові функції, такі як можливість створення завдань, зберігання результатів, перевірка робіт, надання зворотного зв'язку та аналітичні інструменти для оцінки виконання завдань учнями, що дозволить створити чітку специфікацію, на основі якої буде розроблено вебзастосунок.

Розробка архітектури та дизайну інтерфейсу користувача. Після формулювання функціональних вимог розробити архітектуру вебзастосунку, яка включатиме структуру бази даних, а також алгоритми роботи основних функцій. Дизайн інтерфейсу користувача створити з урахуванням принципів зручності, доступності та естетичності, що забезпечить комфортне використання застосунку як для викладачів, так і для учнів. Важливо, щоб інтерфейс був інтуїтивно зрозумілим, що дозволить швидко освоїти його функціонал.

Реалізація бази даних та основного функціоналу застосунку. На цьому етапі потрібно реалізувати програмування основних компонентів вебзастосунку, включаючи інтеграцію з базою даних для зберігання інформації про домашні завдання, результати виконання та зворотний зв'язок. Усе це повинно дозволити створити функціональний продукт, який зможе виконувати поставлені завдання.

Тестування вебзастосунку та оцінка його ефективності. Останнім етапом потрібно протестувати розроблений вебзастосунок. Тестування включатиме перевірку всіх функцій на відповідність вимогам, а також оцінку зручності використання. Важливо також зібрати відгуки від потенційних користувачів, що дозволить внести необхідні корективи та покращити застосунок. Оцінку ефективності базувати на досягненні цілей, поставлених на початку дослідження, а також на аналізі отриманих результатів роботи вебзастосунку в реальному навчальному процесі.

Об’єктом дослідження є інтерактивний процес створення та перевірки домашніх завдань, що реалізується за допомогою сучасних веб-технологій, які сприяють оптимізації навчання.

Предметом дослідження є вебзастосунок, призначений для ефективного створення, зберігання та перевірки домашніх завдань, що забезпечує інтеграцію між учнями та викладачами в освітньому процесі.

Методи дослідження. Для вирішення визначених завдань та досягнення поставленої мети кваліфікаційної роботи використовувався комплекс методів дослідження, що дозволив всебічно охопити всі аспекти розробки вебзастосунку для створення та перевірки домашніх завдань.

Основними методами дослідження є:

Аналіз літератури. Проведено вивчення сучасних наукових і технічних публікацій, які присвячені проектуванню та розробці вебзастосунків, а також публікацій, які присвячені використанню інформаційних технологій в освітньому процесі. Це дозволило визначити актуальні підходи до побудови архітектури, вибору стеку технологій і засобів забезпечення безпеки даних.

Методи системного аналізу. Використано для моделювання структури вебзастосунку, формування вимог і визначення взаємозв'язків між його компонентами. Системний аналіз дав змогу розглянути систему як єдине ціле, визначити її підсистеми (фронтенд, бекенд, база даних, AI-модуль) та встановити логічні зв'язки між ними.

Методи проектування. Застосовано під час створення архітектури багаторівневого вебзастосунку, моделі бази даних, інтерфейсу користувача та REST-сервісів. Ці методи забезпечили структурованість, модульність і розширюваність програмного продукту.

Емпіричні методи. Використано на етапі практичної реалізації системи. До них належать експериментальна розробка програмних модулів, налагодження, тестування функціональних можливостей та оцінювання продуктивності системи під час роботи з реальними даними.

Методи статистичного аналізу. Методи статистичного аналізу застосовувалися для узагальнення результатів роботи системи, зокрема для перевірки коректності обробки даних, оцінювання стабільності виконання запитів і визначення середніх показників ефективності. Завдяки використанню цих методів здійснювалося порівняння результатів роботи різних модулів системи, аналіз їх продуктивності та перевірка відповідності отриманих даних очікуваним

значенням. Це дозволило об’єктивно оцінити якість функціонування вебзастосунку та підтвердити досягнення поставлених цілей.

Використання зазначених методів у комплексі забезпечило системний підхід до розробки вебзастосунку, сприяло досягненню поставлених цілей та реалізації запланованих завдань, що в підсумку призвело до створення ефективного та зручного інструменту для навчання.

Практичне значення дослідження полягає у створенні вебзастосунку, який автоматизує процес створення, подання та перевірки домашніх завдань, підвищуючи ефективність взаємодії між викладачами та студентами. Розроблена система може бути впроваджена в освітні заклади для організації дистанційного та змішаного навчання, зменшення навантаження на викладачів і покращення контролю успішності учнів. Отримані результати можуть бути використані для подальшої розробки подібних освітніх платформ або інтеграції з існуючими системами управління навчальним процесом.

Апробація і впровадження результатів дослідження. Основні результати роботи доповідалися на Міжнародній науково-практичній конференції «Розвиток сучасної науки: актуальні питання теорії та практики» (Запоріжжя, 20 червня 2025 р.) та були представлені у вигляді тез доповіді «Виклики та рішення при розробці веб-платформи для автоматизації створення та перевірки домашніх завдань» [32].

Додатково результати дослідження були представлені на XVIII Всеукраїнській науково-практичній конференції «Інформаційні технології в професійній діяльності» (Рівне, 10 листопада 2025 р.). у вигляді тез доповіді «Веб-застосунок для створення і перевірки домашніх завдань у цифровізації освітнього процесу».[29]

Структура роботи. Кваліфікаційна робота складається зі вступу, трьох розділів, висновків, переліку використаних джерел та додатків. Вступ містить обґрунтування актуальності теми, формулювання мети та завдань дослідження, а також опис методологічної основи, на якій ґрунтується робота.

Розділ 1. Теоретичні основи створення вебзастосунків складається з трьох підрозділів:

У першому підрозділі розглядаються поняття та особливості вебзастосунків, а також розглянуто сучасні вебзастосунки.

Другий підрозділ присвячений архітектурі вебзастосунків.

Третій підрозділ аналізує основні технології веб-розробки, що використовуються у сучасних вебзастосунках.

У четвертому підрозділі розглянуто теоретичні засади вибору технологічного стеку для розробки вебзастосунку.

Розділ 2. Проектування вебзастосунку для створення та перевірки домашніх завдань включає чотири підрозділи:

У першому підрозділі визначаються функціональні та нефункціональні вимоги до системи.

Другий підрозділ описує архітектуру вебзастосунку.

Третій підрозділ присвячений проектуванню бази даних для функціонування застосунку.

Четвертий підрозділ охоплює проектування інтерфейсу користувача.

Розділ 3. Реалізація вебзастосунку для створення та перевірки домашніх завдань містить три підрозділи:

Перший підрозділ описує структуру та основні модулі застосунку.

Другий підрозділ присвячений реалізації функціональних можливостей системи.

Третій підрозділ розглядає інтерфейс користувача та тестування системи.

Висновки підсумовують результати дослідження, формулюючи основні результати та рекомендації, які можуть бути корисними для подальшої роботи в цій галузі.

Список використаних джерел містить літературу та інші джерела, які були використані під час написання роботи.

Додатки містять додаткові матеріали, що деталізують технічні результати розробки та поглиблюють розуміння структури вебзастосунку, принципів взаємодії його компонентів і процесів тестування. Вони охоплюють схему даних, результати

модульних перевірок ключових сервісів та приклади реалізації механізмів генерації звітів.

Загальний обсяг роботи становить 74 сторінок. Вона містить 11 рисунків та 4 таблиці, а список використаних джерел включає 33 найменування. Обсяг додатків – 4 сторінки.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ СТВОРЕННЯ ВЕБЗАСТОСУНКІВ

1.1 Поняття та особливості вебзастосунків

1.1.1. Поняття вебзастосунку

Вебзастосунки стали невід'ємною частиною досвіду користувачів в інтернеті, при чому більшість людей навіть не усвідомлює, що використовують їх щоденно.

Вебдодаток – це програмне забезпечення, яке запускається у вікні браузера чи на телефоні. Усі дані, які отримують користувачі під час використання застосунку, зберігаються на сервері, тому немає потреби встановлювати додаток на локальний пристрій (комп'ютер чи ноутбук) і займати цінний ресурс – пам'ять пристрою.

Більшість користувачів навіть не здогадуються, що працюють із вебдодатком, адже його зовнішній вигляд дуже нагадує звичайний сайт. Проте ключова перевага вебдодатка – це його розширені можливості: обробка даних у реальному часі, редагування фото- та відеоконтенту, здійснення покупок чи продажів тощо. Головна умова для роботи – стабільний доступ до інтернету.

Коли мова йде про роботу вебдодатку, варто врахувати один важливий аспект: незалежно від сфери застосування – комунікації, електронна комерція чи фінансовий сектор – усі вебдодатки функціонують за однаковими принципами.

Коли ми говоримо про те, як працює вебдодаток, слід знати такий факт. Незалежно від того, в якій сфері використовуються вебдодатки, для комунікацій, eCommerce, фінансового сектору, – цей інструмент працює за єдиною системою. Продукт складається з трьох основних компонентів: клієнтської частини для взаємодії з користувачем, серверної частини, що забезпечує внутрішні процеси (реалізує логіку роботи додатку), та бази даних, в якій зберігаються всі файли (дані) [27].

1.1.2. Виклики та недоліки вебзастосунків

Безпека вебзастосунків

Веб-сайти є невід'ємною частиною сучасного світу, але вони також стикаються з багатьма загрозами безпеки, які можуть пошкодити їх репутації, функціональності та конфіденційності. Шкідливі атаки можуть призвести до крадіжки даних, поширення вірусів, злому або знищення вебсайтів. Тому важливо знати, як захистити свої вебсайти від таких загроз.

Загрози безпеці веб-сайтів можуть бути розрізнені на два типи: активні та пасивні.

Активні загрози – це цілеспрямовані дії, спрямовані на пошкодження або несанкціонований доступ до вебсайту. Основні види таких атак:

Ін'єкції коду – впровадження шкідливих скриптів у базу даних або код сайту для викрадення даних чи перенаправлення користувачів.

Перехоплення сесій – отримання ідентифікатора сесії користувача з метою доступу до його облікового запису.

Атаки на відмову у обслуговуванні (DoS) – перевантаження сервера великою кількістю запитів, що блокує роботу сайту.

Крос-сайтовий скриптинг (XSS) – вставлення шкідливого коду, який виконується у браузері користувача та може викрасти дані або підмінити контент.

Крос-сайтове підмінювання запитів (CSRF) – використання вразливостей сайту для надсилання підроблених запитів від імені користувача, що може змінити його налаштування чи дані. [25]

Пасивні загрози передбачають неактивне втручання, коли зловмисник намагається перехопити або переглянути дані користувачів. Основні типи:

Прослуховування – перехоплення мережевого трафіку між сайтом і користувачем з метою отримання конфіденційних даних (логінів, паролів, платіжної інформації)..

Аналіз трафіку –Відстеження обсягів і частоти з'єднань для збору статистичних даних про користувачів або структуру запитів [25].

Залежність від інтернет-з'єднання

Проблеми з доступом: Веб-застосунки вимагають стабільного та швидкого інтернет-з'єднання, що може стати перешкодою для користувачів у районах з поганим доступом до інтернету.

- *Перебої в роботі*: Втрата з'єднання може призвести до втрати даних або неможливості виконати завдання, що вплине на досвід користувачів.

Технічні обмеження

- *Крос-платформна підтримка*: Хоча вебзастосунки повинні працювати на різних пристроях, їх функціональність може варіюватися залежно від типу браузера чи операційної системи, що може ускладнити досвід користувача.
- *Вимоги до ресурсів*: Окремі вебзастосунки споживають значні системні ресурси, що знижує продуктивність на застарілих або слабких пристроях.

Мінуси кросплатформної розробки застосунків

Обмежена гнучкість. Такі рішення мають нижчий рівень інтеграції з функціями конкретних платформ і пристроїв, ніж нативні додатки.

Зниження продуктивності. Через обмеження оптимізації кросплатформні застосунки працюють повільніше й менш стабільно порівняно з нативними аналогами.

Невідповідність дизайну. Можливі відмінності у вигляді та поведінці інтерфейсу на різних ОС (iOS, Android, Windows, macOS), що впливає на єдність UX.

Складнощі з публікацією. Вимоги App Store і Google Play відрізняються, тому проходження перевірок може бути тривалим і бюрократично складним.

Витрати на розробку та підтримку. Створення складного вебзастосунку потребує значних фінансових і часових ресурсів, а також постійного оновлення та забезпечення безпеки.

Ризики академічної недоброчесності. Використання освітніх вебзастосунків може спричинити зловживання, зокрема списування чи використання готових рішень, що знижує об’єктивність оцінювання[30].

1.1.3. Основні відмінності між вебдодатком і сайтом

Вебзастосунки поділяються на два основні типи – вебсайти та вебдодатки, які, попри певну схожість, виконують різні функції.

Вебсайти здебільшого орієнтовані на подання інформації у вигляді текстів, зображень чи відео, тоді як вебдодатки є інтерактивними програмами, що надають користувачам розширені можливості: авторизацію, обробку даних, персоналізацію та інші функції, які забезпечують індивідуальний досвід взаємодії.

Розглянемо детальніше основні відмінності вебдодатків і сайтів:

1. Функціональність. Вебсайти орієнтовані на подання інформації (тексти, зображення, відео), тоді як вебдодатки забезпечують інтерактивну взаємодію, підтримують обробку даних, створення акаунтів, платежі та персоналізацію.

2. Технології. Сайти створюють за допомогою HTML, CSS і JavaScript, а для вебдодатків використовують складніші фреймворки – React, Angular, Node.js чи Vue.js, що забезпечують динамічність і продуктивність.

3. Мобільність. Вебсайти універсальні для будь-яких пристроїв, тоді як вебдодатки часто оптимізують під смартфони чи планшети, адаптуючи інтерфейс до конкретного екрана..

4. Розробка та підтримка. Сайти потребують менше ресурсів і зусиль для створення та оновлення, тоді як вебдодатки мають складнішу архітектуру, інтеграцію з базами даних, системами безпеки та регулярне оновлення функцій.

Вибір між веб-сайтом та веб-додатком залежить від конкретних бізнес-цілей. Якщо основна мета – надання інформації про компанію, продукти чи послуги, веб-сайт буде достатнім рішенням. Проте, якщо бізнесу потрібна функціональність для активної взаємодії з користувачами, така як обробка замовлень, управління обліковими записами або персоналізовані налаштування, варто обрати веб-додаток, який зможе забезпечити всі ці можливості [33].

Ключові характеристики, що визначають вебзастосунок:

Доступність. Використовуються з будь-якого пристрою, підключеного до Інтернету, що забезпечує зручний доступ незалежно від місця та типу обладнання.

Адаптивність. Працюють у різних браузерах і на будь-яких операційних системах без потреби створювати окремі версії, що знижує витрати на розробку.

Відсутність інсталяції. Не потребують встановлення – користувач отримує доступ одразу через браузер, а всі оновлення застосовуються автоматично під час завантаження сторінки.

Безпека. Централізована система керування доступом підвищує рівень захисту та контроль над даними.

Збереження даних. Інформація користувачів зберігається у хмарному середовищі, що забезпечує її захист і запобігає втратам у разі пошкодження пристрою. [26]

1.1.4. Сучасні вебзастосунки для створення та перевірки домашніх завдань

Сучасні інформаційні технології значно змінили організацію навчального процесу, зокрема способи створення, подання та перевірки домашніх завдань. Для покращення взаємодії між викладачами та студентами використовуються вебзастосунки, які автоматизують ці процеси.

У сучасному освітньому середовищі існує значна кількість онлайн-платформ, які слугують допоміжним засобом у процесі створення, подання та перевірки навчальних завдань як у дистанційному, так і в змішаному форматі навчального процесу. Для того аби визначитися, які з рішень є найкращими, а які – менш ефективними, варто ознайомитись з прикладами існуючих вебзастосунків, їх можливостями, зручністю використання та ефективністю використання для організації навчання (навчального поцесу).

Google Classroom є одним із найпоширеніших інструментів, що застосовується у закладах освіти для організації дистанційного навчання. Щоб зрозуміти, чому обирають саме цей вебзастосунок розглянемо його сильні та слабкі сторони.

Переваги:

Простота використання та швидке налаштування. Викладач може створити клас, додати учнів за допомогою коду або запрошення, а студенти – швидко приєднатися до навчання через браузер або мобільний застосунок.

Інтеграція з сервісами Google. Підтримує інтеграцію з Google Drive, Docs, Sheets, Slides, що забезпечує ефективне управління матеріалами та обміном інформацією.

Зручна організація завдань. Дозволяє легко створювати, публікувати та контролювати виконання завдань, оскільки система дозволяє переглянути, хто здав роботу, а хто ще працює над нею.

Недоліки:

Відсутність автоматичних оновлень. Стрічка активності не оновлюється автоматично, тому аби учні не пропустили важливі оголошення, їм потрібно регулярно виконувати її оновлення.

Відсутність автоматизованих тестів. У системі не передбачено створення автоматизованих тестів, що дозволили б зменшити навантаження на викладачів за допомогою автоматичної перевірки такого типу завдань.

Залежність від екосистеми Google. Оскільки платформа потребує використання Google-сервісів, це може створити незручності для користувачів, які ніколи не користувалися цими інструментами [7].

Ще одна з найпоширеніших систем для управління навчанням, яка використовується в освітніх закладах є Moodle, яка як і Google Classroom має свої сильні та слабкі сторони.

Переваги:

Відкритий код. Платформа використовує відкритий код, що робить її економічно вигідною для навчальних закладів, оскільки базова версія доступна без оплати, а її функціонал можна гнучко підлаштувати під конкретні освітні вимоги.

Потужна система керування курсами. Moodle пропонує різноманітні функції, від створення курсів, завдань, тестів до створення групових проектів. Оскільки функції достатньо адаптивні, платформа може використовуватись у

освітніх установах різного рівня, від середньої школи до закладів вищої освіти та корпоративного навчання.

Гнучкість і розширюваність. Оскільки велика спільнота Moodle робить свій внесок у розробку плагінів та покращення системи, усунення недоліків та вразливостей відбувається швидко, а для підвищення ефективності навчального процесу існує достатньо інструментів для інтеграції додаткових функцій.

Недоліки:

Дизайн інтерфейсу користувача. Оскільки інтерфейс трохи застарілий, складний та перевантажений, платформа не така інтуїтивно зрозуміла у порівнянні з іншими.

Ресурсоємність. Moodle потребує власного хостингу, регулярних оновлень та технічного обслуговування, що робить платформу ресурсозатратною.

Надмірна універсальність. Хоча платформа надає широкий функціонал, вона буде надто громіздкою та малоефективною для вирішення конкретного завдання такого як для створення простої системи управління домашніми завданнями [12].

1.2 Архітектура вебзастосунків

1.2.1. Сутність та роль архітектури вебзастосунку

Архітектура вебзастосунків – це високорівнева структура, що визначає, принципи роботи, функціонування та масштабованості програмного продукту. Вона є основою проектування системи, поєднуючи фронтенд, бекенд і базу даних у єдину логічну модель.

Архітектура задає правила взаємодії між компонентами, забезпечує узгодженість їх роботи, продуктивність, безпеку та зручність підтримки. Її вибір є важливим етапом розробки, адже саме від нього залежать надійність і подальший розвиток вебзастосунку [24].

Важливість архітектури веб-додатків полягає в тому, що вона не тільки забезпечує правильне налаштування коду, але й визначає його продуктивність, безпеку, масштабованість та можливості для подальшого розвитку. При правильному проектуванні архітектури можливо знизити витрати на розробку та обслуговування, підвищити якість продукту та забезпечити гнучкість у разі змін або масштабування.

Архітектура вебдодатків включає кілька аспектів:

Модульність: розділення функціональних компонентів на окремі модулі полегшує підтримку та розробку системи.

Масштабованість: здатність системи працювати при різних навантаженнях і збільшувати число користувачів або обробляти дані без істотної зміни структури.

Безпеку: захист від потенційних атак і збереження конфіденційності даних.

Продуктивність: забезпечує швидку обробку запитів користувачів і мінімізує системні затримки. [24]

Завдяки правильній архітектурі вебдодатків можна домогтися високого рівня ефективності в роботі продукту, здатності адаптуватися до змін умов експлуатації і можливості легкого масштабування в майбутньому.

1.2.2. Основні компоненти вебзастосунків

У веброзробці створення інтерактивного користувацького досвіду ґрунтується на взаємодії трьох основних компонентів вебзастосунку: фронтенду (клієнтська частина), бекенду (серверна частина) та бази даних. Їх узгоджена робота визначає ефективність і стабільність системи.

Frontend – це клієнтська сторона, з якою користувач взаємодіє безпосередньо у браузері. Вона охоплює візуальні елементи інтерфейсу (UI), відповідає за відображення даних із сервера та передавання введеної користувачем інформації для подальшої обробки [20].

Основні технології, що використовуються для створення інтерфейсу користувача, включають:

HTML (HyperText Markup Language): базова мова розмітки, яка визначає структуру та зміст веб-сторінки.

CSS (Cascading Style Sheets): мова стилів, що додає візуальну привабливість, дозволяючи налаштувати кольори, шрифти, макети та інші аспекти дизайну.

JavaScript: мова програмування, що додає інтерактивність і динамічність, дозволяючи сторінкам реагувати на дії користувача.

Фреймворки та бібліотеки: такі як React, Angular, Vue.js і jQuery, які спрощують процес розробки інтерфейсу і додають багатий функціонал для покращення взаємодії користувача.

Технології, що зазвичай використовуються для розробки серверної частини:

— Мови програмування: Python, Java, Ruby, PHP та Node.js, які визначають логіку та обробку даних.

— Бази даних: MySQL, PostgreSQL, MongoDB та Redis, що зберігають і надають доступ до даних.

— Сервери: Apache, Nginx та інші для обробки запитів і передачі інформації.

— Фреймворки: такі як Django, Flask, Express, Spring і Rails, які спрощують розробку серверних функцій і допомагають підтримувати організовану структуру коду.

Ці технології разом забезпечують стабільність і надійність функціонування веб-додатків [8].

База даних вебдодатку – це система зберігання й керування інформацією, що використовується застосунком (користувачі, контент, налаштування тощо). Вона забезпечує швидкий доступ, цілісність і надійність даних.

БД для веб-додатків можуть бути створені на основі різних систем управління базами даних, включаючи:

— Реляційні БД (MySQL, PostgreSQL, Oracle) – структурують дані у таблицях і підтримують складні зв’язки.

— NoSQL БД (MongoDB, Cassandra, DynamoDB) – гнучко працюють із великими та нереляційними даними.
— Графові БД (Neo4j) – ефективні для зберігання взаємозв'язків, наприклад, у соціальних або рекомендаційних системах.

Вибір типу бази даних залежить від специфіки вебдодатку та характеру оброблюваних даних. Такі БД орієнтовані на масштабованість, багатокористувацький доступ і сумісність із фреймворками на зразок Django, Node.js чи Ruby on Rails, що забезпечує ефективне керування даними. [9]

Сьогодні у веб-розробці існують дві основні архітектури, що визначають спосіб взаємодії між серверними і зовнішніми компонентами.

1. *Серверні програми.* Браузер надсилає HTTP-запит до сервера, який формує HTML-сторінку у відповідь. Під час обробки запиту сервер може звертатися до бази даних і вставляти отримані дані у шаблон (ERB, Blade, EJS, Handlebars). HTML визначає структуру сторінки, CSS – оформлення, а JavaScript – інтерактивність..
2. *Спілкування за допомогою AJAX (Asynchronous JavaScript and XML).* Дозволяє надсилати запити без перезавантаження сторінки, зазвичай у форматі JSON. Такий підхід лежить в основі односторінкових застосунків (SPA), створених за допомогою фреймворків Angular, React, Ember тощо. Дані динамічно оновлюються на клієнті, що робить роботу системи швидшою й зручнішою.
3. *Ізоморфні (універсальні) програми.* Сучасні фреймворки (React, Next.js, Ember) дозволяють відображати сторінки як на сервері, так і на клієнті. Це забезпечує оптимальну продуктивність і швидке завантаження контенту завдяки комбінуванню серверного HTML і AJAX-запитів. [10]

1.2.3. Моделі архітектури та їх вплив на розробку вебзастосунків

Архітектура вебзастосунку визначає спосіб взаємодії між його основними компонентами – клієнтською частиною, сервером і базою даних. Вибір

архітектурної моделі впливає на масштабованість, безпеку, продуктивність і зручність підтримки системи [16].

Розглянемо кілька основних архітектурних підходів та їхній вплив на процес розробки.

1. *Клієнт-серверна архітектур*а. Ця модель передбачає розподіл завдань між клієнтськими та серверними компонентами, які взаємодіють через мережу. Клієнт надсилає запити, а сервер обробляє їх, звертаючись до бази даних і повертаючи результати у вигляді вебсторінки або даних. Такий підхід забезпечує централізоване зберігання інформації, єдине управління доступом та ефективну організацію взаємодії між користувачами й системою.

   До основних переваг клієнт-серверної моделі належать централізоване керування та оновлення даних, можливість масштабування та балансування навантаження, оптимальне використання ресурсів клієнтської і серверної частин, а також зниження витрат на обслуговування завдяки централізації. За наявності резервного копіювання система набуває підвищеної надійності. Водночас така архітектура має і недоліки: вона створює ризики безпеки (зокрема DoS-та MITM-атак), що потребує застосування методів шифрування, а також постійного моніторингу. Додатковим обмеженням є залежність від стабільної роботи сервера та початкові витрати на розгортання системи [17].

   Загалом клієнт-серверний підхід дає змогу чітко розділити фронтенд і бекенд, полегшуючи масштабування й підтримку, однак вимагає ретельного впровадження безпекових механізмів.Клієнт-серверна архітектура дозволяє розробникам розділяти фронтенд і бекенд, спрощуючи масштабування та підтримку, але потребує посилених заходів безпеки.

2. *Мікросервісна архітектура.* Мікросервісна модель передбачає поділ системи на незалежні сервіси, кожен з яких виконує окрему функцію та взаємодіє з іншими через REST або черги повідомлень. Така структура забезпечує можливість одночасної роботи кількох команд над різними компонентами, що прискорює розробку. Кожен сервіс може масштабуватися й оновлюватися

автономно, а відмова одного модуля зазвичай не призводить до повної зупинки системи. Крім того, архітектура дозволяє поступово додавати новий функціонал без ризику порушення цілісності всієї платформи. Серед недоліків варто виокремити складність управління міжсервісними взаємозв'язками, зростання вимог до безпеки та системи моніторингу, а також підвищені витрати на інфраструктуру і координацію роботи сервісів [30].

Мікросервісний підхід є доцільним для великих масштабованих платформ, проте потребує значного технічного досвіду у створенні розподілених систем.

3. *Монолітна архітектура*. Монолітний підхід передбачає створення всієї системи як єдиного коду з тісно пов'язаними компонентами. Такий тип зручний на ранніх етапах проекту або для невеликих систем із стабільними вимогами.

   До її переваг належать цілісна кодова база, що спрощує налагодження, тестування та розгортання, відсутність міжкомпонентної взаємодії, яка зменшує затримки, а також можливість масштабування шляхом дублювання застосунку. Моноліт спрощує керування конфігураціями та логуванням. Однак будь-які зміни у функціоналі вимагають повторного розгортання всього застосунку, що уповільнює CI/CD-процеси. Великі кодові бази важко підтримувати, збій одного модуля може порушити роботу всієї системи, а використання нових технологій є суттєво обмеженим [19].

   Монолітна архітектура залишається ефективною для невеликих проєктів, але з втратою гнучкості при розширенні системи.

4. *Трирівнева архітектура.* Цей підхід розділяє систему на три рівні: презентації (UI), застосунків (бізнес-логіка) і даних (база даних). Таке структурування забезпечує чітке розмежування відповідальностей, підвищує безпеку та спрощує масштабування.

   Серед її переваг можна виокремити можливість незалежного оновлення кожного рівня, контроль цілісності даних через проміжний рівень

застосунків, відсутність прямого доступу клієнта до бази даних, а також підтримку механізмів кешування й балансування навантаження. Архітектура забезпечує гнучке масштабування й покращує обслуговування системи.

До недоліків належать ускладнення структури застосунку, збільшення часу розробки, додаткове мережеve навантаження через використання проміжного рівня та потреба в окремому сервері для обробки запитів [3].

Загалом трирівнева модель є оптимальним рішенням для великих систем із високими вимогами до безпеки, стабільності та модульності.

1.2.4. Вибір архітектури для вебзастосунку

Вибір архітектури вебзастосунку є ключовим етапом розробки, оскільки визначає структуру системи, взаємодію її компонентів, ефективність, масштабованість і безпеку. Від архітектури залежить продуктивність, надійність роботи та можливість інтеграції з іншими сервісами.

Одним із головних критеріїв є адаптивність – здатність системи масштабувати окремі модулі без суттєвих змін у коді. Не менш важливою є безпека, що потребує впровадження механізмів шифрування, аутентифікації та контролю доступу, особливо в розподілених архітектурах.

Архітектура безпосередньо пов’язана з вибором технологій: мови програмування, фреймворки та бази даних мають відповідати вимогам продуктивності, надійності й обсягу даних. При роботі з великими навантаженнями пріоритет надається технологіям, що підтримують масштабування.

Також враховується ресурсна ефективність – складні архітектури потребують більше часу на розробку й обслуговування, тому важливо обрати рішення, що відповідає доступним ресурсам і термінам реалізації.

Отже, правильно обрана архітектура забезпечує ефективність, безпеку й стабільність системи, створюючи основу для її подальшого розвитку

1.3 Основні технології веброзробки

Створення сучасних вебзастосунків ґрунтується на використанні комплексу технологій, що забезпечують їхню функціональність, продуктивність і зручність. Вони охоплюють клієнтську та серверну частини, зберігання й обмін даними, а також інструменти для розробки, тестування та впровадження програмного забезпечення.

У цьому підрозділі розглянуто основні групи технологій веброзробки: клієнтські (Front-end), серверні (Back-end), сховища даних, технології обміну даними та інструменти розробки, що забезпечують ефективне створення й підтримку вебзастосунків.

1.3.1. Front-end

Сучасна веброзробка швидко еволюціонує, і якість користувацького досвіду значною мірою залежить від технологій Front-end. Це видима частина вебзастосунку, з якою безпосередньо взаємодіє користувач. Основу фронтенд-розробки становлять три базові технології:

HTML – мова розмітки, що визначає структуру та вміст сторінки. Вона дозволяє створювати елементи, як-от заголовки, абзаци, посилання, зображення й гіперпосилання, забезпечуючи логічну організацію контенту. HTML-документи є базою Інтернету та інтерпретуються браузерами для візуального відображення вебсторінок [15].

CSS – мова таблиць стилів, яка визначає зовнішній вигляд HTML-елементів. Вона дозволяє створювати адаптивні інтерфейси, оптимізовані для різних пристроїв, використовувати один файл стилів для кількох сторінок і зменшувати обсяг коду. CSS робить сторінки візуально привабливими та покращує користувацький досвід [1].

JavaScript(JS) – мова програмування, що забезпечує інтерактивність і динамічну поведінку сторінок. Вона працює на більшості пристроїв і підтримує як клієнтську, так і серверну розробку. Сучасні вебзастосунки використовують фреймворки React, Vue.js та Angular для створення односторінкових застосунків (SPA), які забезпечують швидку та зручну взаємодію з інтерфейсом [2, 11].

1.3.2. Back-end

Back-end (серверна частина) відповідає за реалізацію бізнес-логіки, збереження даних, обмін із клієнтською частиною, а також за безпеку, автентифікацію та керування сесіями. Сучасна бекенд-розробка базується на принципах модульності, масштабованості та надійності, використовуючи різні мови програмування.

Згідно з Stack Overflow Developer Survey 2023, основні мови Back-end (Python, Java, JavaScript, PHP, Go) залишаються найпопулярнішими серед розробників [13].

Python – універсальна мова, відома простотою синтаксису та широким застосуванням у веброзробці, науці про дані та ШІ. У бекенді часто використовується з фреймворками Django (повнофункціональний, архітектура MVC) і Flask (мікрофреймворк для менших проєктів). Переваги: простота, зріла екосистема. Недоліки: обмеження багатопотоковості (GIL).

Java – об'єктно-орієнтована, стабільна мова, що лежить в основі великих корпоративних систем. Переваги: багатопотоковість, розвинена екосистема, надійність. Недоліки: складний синтаксис, велике споживання пам'яті.

JavaScript (Node.js) – дозволяє використовувати одну мову для фронтенду й бекенду. Подійно-орієнтована архітектура забезпечує швидкість і масштабованість. Популярні фреймворки: Express.js, Next.js, Nuxt.js. Переваги: універсальність, швидка розробка. Недоліки: відсутність статичної типізації, однопотокова природа.

PHP – спеціалізована для веброзробки мова з великою спільнотою. Використовуються фреймворки Laravel (швидка розробка) і Symfony (модульність). Переваги: простота та поширеність. Недоліки: несумісність назв функцій, що ускладнює підтримку.

Go (Golang) – створена Google мова, орієнтована на продуктивність і паралельне програмування. Переваги: швидка компіляція, простий синтаксис. Недоліки: молода екосистема порівняно з Java чи JS. [13]

1.3.3. Бази даних

Бази даних (БД) – ключовий компонент динамічних вебзастосунків, який забезпечує зберігання, обробку та безпечний обмін даними між клієнтом і сервером. Вони дозволяють структурувати інформацію, формувати динамічний контент, масштабувати систему та підтримувати цілісність і безпеку даних.

Основні функції баз даних у веброзробці:

— структурування даних у таблицях, рядках і стовпцях;

— забезпечення динамічності контенту шляхом обробки запитів у реальному часі;

— масштабованість і підтримка високих навантажень;

— безпека та контроль доступу до конфіденційної інформації;

— цілісність і узгодженість даних через механізми обмежень і транзакцій.

Сучасні СУБД підтримують властивості ACID (атомарність, узгодженість, ізоляція, довговічність), роботу з SQL-запитами, контроль паралельності та зв'язки між таблицями. Це забезпечує стабільність і точність обробки даних у багатокористувацьких системах.

Найпоширеніші типи баз даних у веброзробці:

MySQL – реляційна СУБД із високою швидкодією та простотою інтеграції.

PostgreSQL – потужна реляційна БД з підтримкою складних транзакцій.

MongoDB – гнучка NoSQL БД для великих і неструктурованих даних.

SQLite – легка безсерверна реляційна БД для невеликих проектів.

Firebase – хмарна NoSQL платформа для застосунків реального часу.

Redis – сховище типу «ключ–значення», що використовується для кешування й керування сесіями [6].

Вибір бази даних визначається типом вебзастосунку, структурою та обсягом даних, вимогами до продуктивності й масштабованості, а також доступними ресурсами для розробки й підтримки системи.

1.3.4. Технології обміну даними

Обмін даними – це процес передачі інформації між системами, платформами чи користувачами, що забезпечує інтеграцію, автоматизацію та ефективне управління даними. Він лежить в основі більшості сучасних веб-застосунків, підтримуючи аналітику, штучний інтелект і роботу хмарних сервісів. Обмін здійснюється через API-інтерфейси, передачу файлів, потокові конвеєри або хмарні платформи, залежно від потреб системи [21].

Типи обміну даними

У режимі реального часу – миттєва передача даних у відповідь на події (використовується в IoT, моніторингу чи динамічному ціноутворенні).

Пакетний обмін – передавання великих обсягів інформації за розкладом (щогодини, щодня, щотижня).

Потоковий обмін – безперервна передача невеликих частин даних для аналітики в реальному часі. [21]

За архітектурою обмін даними поділяється на:

API-обмін – стандартизований доступ до даних, що спрощує інтеграцію мікросервісів і хмарних систем.

Обмін на основі подій – передача даних у момент виникнення події, що зменшує навантаження на мережу.

Черги повідомлень (Apache Kafka, RabbitMQ) – асинхронна модель "pub/sub" для масштабованих і розподілених систем. [21]

За моделлю доступу:

Приватний обмін – передача даних між довіреними сторонами з контролем доступу та аудитом.

Публічний обмін – відкриті API або державні репозиторії, що сприяють інноваціям, але потребують надійного захисту.

Одноранговий (peer-to-peer) – прямий зв'язок між системами без центрального брокера, що підвищує стійкість і автономність. [21]

Формати даних

1. Текстові формати:

JSON – гнучкий, незалежний від мови формат для обміну даними в реальному часі.

XML – структурований стандарт W3C, що підтримує складні ієрархії.

CSV – простий табличний формат для звітності та аналітики.

YAML – зручний для людини формат, сумісний із JSON.

2. Двійкові формати:

Protobuf – ефективна серіалізація структурованих даних у мікросервісах.

Avro – формат для екосистеми Hadoop, підтримує динамічні схеми та стиснення.

CORBA – ранній стандарт двійкового обміну між мовами, нині використовується рідко. [21]

Сучасні технології обміну даними сприяють інтеграції систем, розвитку ШІ та автоматизації, проте залишаються чутливими до ризиків безпеки, сумісності й якості даних, а також потребують значних витрат на інфраструктуру [21].

1.3.5. Інструменти розробки

Інструменти веброзробки – це програмні додатки, фреймворки, бібліотеки та утиліти, які допомагають розробникам проектувати, створювати, тестувати та підтримувати вебсайти та вебдодатки. Ці інструменти спрощують процес розробки, роблячи його швидшим, ефективнішим та організованішим.

Існують різні типи інструментів, які можна обрати для веброзробки. Вони часто включають фреймворки та бібліотеки, що забезпечують основу для створення веб-додатків.

Інструменти фронтенду: слід використовувати для фронтенду, наприклад, для клієнтських веб-сайтів. Наприклад, для інтерфейсу користувача.

Інструменти бекенду: можна використовувати для бекенду з логікою на стороні сервера – бази даних, API тощо.

Інтегровані середовища розробки (IDE): мають функції для редагування вихідного коду та тексту. Також пропонують різні плагіни та інструменти.

Інструменти управління проектами/співпраці: працюють як система контролю версій для вашого проекту.[14]

1.4 Теоретичні засади вибору технологічного стеку для вебзастосунку

На основі проведеного аналізу сучасних підходів до створення вебзастосунків, архітектурних моделей та технологічних рішень, розглянутих вище, доцільно застосовувати стек технологій, який буде забезпечувати ефективність, надійність, безпеку та масштабованість розроблюваної системи для створення та перевірки домашніх завдань.

Під час вибору технологічного стеку враховувалися такі критерії:

— підтримка багаторівневої клієнт–серверної архітектури;

— забезпечення інтерактивності та швидкодії користувацького інтерфейсу;

— простота інтеграції клієнтської та серверної частин через REST API;

— наявність перевірених засобів аутентифікації та авторизації;

— можливість розширення функціоналу без значних змін у кодовій базі.

Для реалізації клієнтської частини було обрано React.js – сучасну бібліотеку JavaScript, що базується на компонентному підході. React забезпечує створення односторінкового застосунку (SPA), який дозволить оновлювати контент без перезавантаження сторінки. Завдяки використанню Material UI та Tailwind CSS можна забезпечити адаптивний, доступний і естетично привабливий інтерфейс. Ці технології дозволяють реалізувати інтуїтивну взаємодію користувача з системою, що є критично важливим для освітнього застосунку.

Для реалізації серверної частина вебзастосунку вирішено використовувати фреймворк Spring Boot, який забезпечує побудову надійних REST-сервісів та підтримку принципів модульності. А використання підсистем Spring Security та JWT-аутентифікації дозволить гарантувати захист даних користувачів, керування користувачами та безпечний обмін інформацією між клієнтом і сервером.

Як систему керування базами даних доцільно використати PostgreSQL, яка характеризується високою продуктивністю, підтримкою транзакцій, розширеною системою прав доступу та сумісністю з Java через Spring Data JPA. Під час розробки й тестування можлива тимчасова заміна PostgreSQL на вбудовану H2 Database, що

забезпечує швидке локальне налаштування без зовнішнього серверного середовища.

Запропонований стек технологій (React.js, Spring Boot, PostgreSQL) поєднує переваги швидкодії, гнучкості, масштабованості та безпеки. Такий набір інструментів відповідає сучасним стандартам розробки освітніх платформ і забезпечує легку підтримку та розширюваність системи.

Підсумовуючи теоретичні засади вибору технологій, можна стверджувати, що використання такого стеку технологій дозволить забезпечити ефективну реалізацію функціональних можливостей майбутнього вебзастосунку, а також відповідатиме вимогам до продуктивності, надійності та захисту даних.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ВЕБЗАСТОСУНКУ ДЛЯ СТВОРЕННЯ ТА ПЕРЕВІРКИ ДОМАШНІХ ЗАВДАНЬ

2.1 Функціональні, нефункціональні вимоги до системи

Розробка якісного продукту завжди починається з визначення вимог, які сформують основу для подальшого проектування та реалізації. Тому варто визначити функціональні вимоги для майбутнього вебзастосунку для створення та перевірки домашніх завдань.

Для початку визначимо функціональні вимоги, які стануть критично важливими для реалізації вебзастосунку.

Майбутній вебзастосунок для створення та перевірки домашніх завдань повинен забезпечити:

1. Реєстрацію та автентифікацію користувачів
   - Система повинна забезпечувати можливість створення нового облікового запису користувача із зазначенням електронної пошти та пароля.
   - Підтримка авторизації за допомогою облікових даних (електронна пошта та пароль) для безпечного входу до системи.
   - Перевірка достовірності даних користувача та захист від несанкціонованого доступу.
   - Користувач отримує роль – *викладач*, *студент або адміністратор*, що визначає доступні функції системи.
   - Автоматичне розпізнавання ролі користувача та перенаправлення до відповідного інтерфейсу.
   - Підтримку JWS- автентифікації, що буде передбачати генерацію токена після успішного входу, перевірку токена для кожного наступного входу.
2. Функціонал для користувачів з роллю “Викладач”
   - Створення нових навчальних завдань, визначення їх змісту, термінів виконання та додавання допоміжних файлів чи посилань.

— Редагування чи видалення завдань.

— Перегляд списку студенських робіт, включно з статусом та часом здачі.

— Перевірка виконаних студентами завдань з можливістю оцінювання, коментування задля надання рекомендацій чи інших потрібних даних студенту по виконаному завданні.

— Перегляд результатів автоматичної перевірки робіт, яка включає в себе оцінювання та коментарі та перевірку на плагіат.

— Отримання статистичних даних про успішність студента.

— Формування звітів та аналітики .

3. Функціонал для користувачів з роллю “Студент”

— Доступ до списку доступних завдань з можливістю перегляду опису, дедлайнів та вкладених файлів чи посилань (за наявності).

— Відправка виконаних завдань у вигляді вкладеного файлу чи передбаченого системою формату.

— Можливість перегляду статусу завдання (“Нове”, “Надіслане”, “Перевірене ШІ”,“Оцінене”).

— Для оцінених завдань можливість перегляду оцінки та коментарів викладача.

— Автоматичне оновлення інформації без перезавантаження сторінок.

— Отримання сповіщень про нові завдання, дедлайни чи оцінки в особистому кабінеті.

4. Перевірку та оцінювання робіт

— Підтримка двох режимів перевірки завдань (ручна і автоматична (використання ШІ).

— При автоматичній перевірці оцінюється відповідність відповіді студента умовам завдання та пропонується попередня оцінка.

— Зберігання результатів перевірки в БД.

— Можливість формування коментарів та звітів про результат після перевірки.

5. Повідомлення та сповіщення
   — Надання сповіщень про наявність нових завдань, наближення термінів кінцевої здачі виконаних робіт та оцінки та коментарі для перевірених завдань.
   — Відображення повідомлень в інтерфейсі вебзастосунку.
   — Спливаючі повідомлення про результати операцій (збереження, відправка).
6. Аналітику та звітність. Система повинна забезпечити розширену аналітику навчального процесу, що дасть змогу викладачам оцінити результативність роботи студентів та відстежити динаміку їхнього прогресу. За допомогою аналітичного модуля система повинна охопити комплекс показників, які відображають результати студентів. На основі даних, зібраних у таблицях submission та task, система повинна обчислювати кількість зданих робіт та частку успішно виконаних завдань і забезпечити визначення середнього балу студента та формування узагальненої статистики за певний період або за конкретним завданням.
7. Адміністрування системи
   — Можливість контролю над користувачами системи (створення, редагування, блокування або видалення облікових записів).
   — Можливість призначення користувачам ролей (“Студент”, “Викладач”, “Адміністратор”)
   — Перегляд журналів активності, логів операцій та системних подій.

Аби наочно зобразити взаємодію користувачів із системою створення та перевірки домашніх завдань побудовано Use Case діаграму (Рисунок 2.1).

На діаграмі показано три основні ролі користувачів: Student, Teacher та Admin, а також допоміжний сервіс AI Service, що здійснює автоматичну перевірку завдань.

Студент може реєструватися, переглядати завдання, надсилати рішення та переглядати рейтинг.

Викладач створює й редагує завдання, перевіряє роботи, переглядає результати студентів і формує звіти.

Адміністратор керує користувачами системи.

Така діаграма добре відображає взаємозв’язки між ролями та основні функціональні сценарії вебзастосунку.

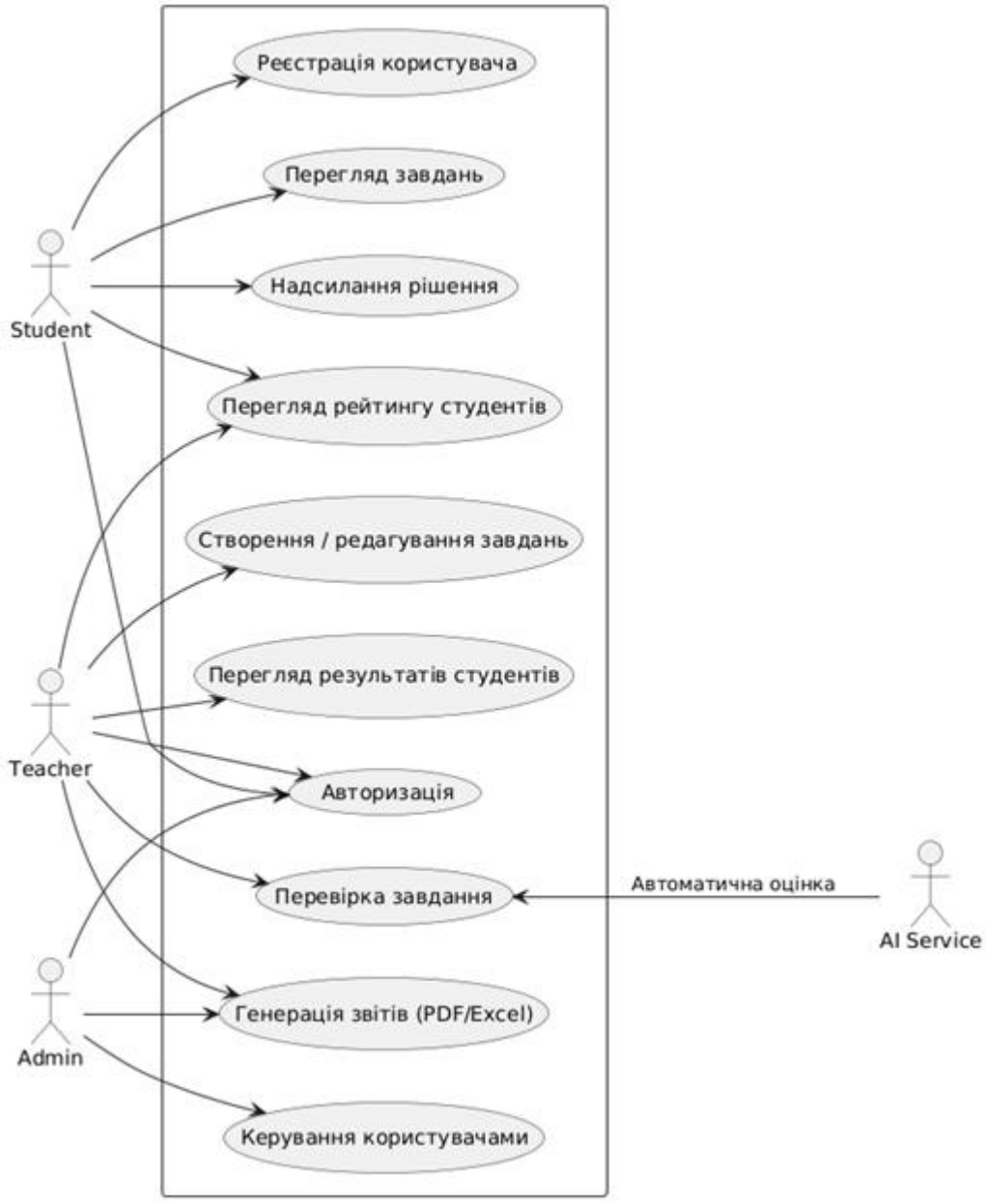

Рисунок 2.1. Use Case діаграма вебзастосунку

Також для забезпечення стабільної роботи системи її безпеки, зручності та ефективності варто визначити, які нефункціональні вимоги повинні бути забезпеченні у вебзастосунку.

Вебзастосунок для створення та перевірки домашніх завдань має забезпечувати швидку обробку запитів, формувати відповідь сервера не довше ніж за 2–3 секунди навіть під час активного використання. Архітектура повинна підтримувати одночасну роботу великої кількості користувачів без помітного зниження швидкодії.

Особливу увагу слід приділити безпеці, передбачивши шифрування даних, захист від SQL-ін'єкцій, XSS-атак і CSRF-запитів, використання JWS-токенів для аутентифікації та авторизації, а також чітке розмежування ролей користувачів із доступом до відповідних функцій.

Система повинна відзначатися надійністю, тобто мати механізми регулярного резервного копіювання бази даних, можливість відновлення після технічних збоїв без втрати критичних даних і ведення журналів помилок для моніторингу роботи, що дозволить гарантувати безперебійну роботу застосунку та збереження важливих даних.

Застосунок повинен мати можливість масштабування як вертикального (підвищення продуктивності серверного обладнання), так і горизонтального (додавання нових вузлів або серверів). Структура системи має передбачати можливість розширення функцій та обробки зростаючого обсягу даних без істотних змін у кодовій базі.

Важливо також забезпечити зручність використання – створити інтуїтивно зрозумілий інтерфейс, адаптований для користувачів із різним рівнем підготовки та придатний для коректного відображення на комп'ютерах, планшетах та смартфонах. Важливими у інтерфейсі є логічна навігація, читабельність, зручність взаємодії та мінімальна кількість дій для виконання основних операцій у системі.

Крім того, вебзастосунок має гарантувати сумісність із сучасними веб-браузерами(Chrome, Firefox, Edge, Safari) та підтримувати стандартні протоколи й

формати взаємодії (HTTP/HTTPS, JSON), що дозволить забезпечити доступність системи для користувачів.

2.2 Архітектура вебзастосунку

Щоразу, коли користувач відкриває веб-браузер, надсилає електронного листа чи переглядає онлайн-зміст, він фактично взаємодіє з розподіленою цифровою системою, яка забезпечує безперебійну передачу, обробку та збереження даних. Ці процеси реалізуються завдяки клієнт–серверній архітектурі, яка є основою функціонування більшості сучасних вебзастосунків.

Клієнт–серверна архітектура – це модель взаємодії, у якій клієнт (користувацький пристрій або веб-браузер) ініціює запит, а сервер обробляє цей запит і повертає відповідь. Взаємодія відбувається за принципом «запит – відповідь», що забезпечує стабільний і передбачуваний обмін даними. Такий підхід дозволяє ефективно організувати роботу системи, забезпечуючи одночасну взаємодію багатьох користувачів із сервером без порушення її роботи [18].

Для проектування вебзастосунку створення та перевірки домашніх завдань варто обрати саме клієнт–серверну архітектуру, оскільки вона забезпечує необхідну гнучкість, масштабованість, розподіл навантаження та безпеку. Ця архітектура дає змогу розмежувати функції між клієнтською та серверною частинами, що полегшує подальший розвиток і супровід системи [4].

У межах обраної архітектури клієнтська частина (front-end) відповідає за відображення інтерфейсу користувача, прийом даних та взаємодію з користувачем у реальному часі. Вона надсилатиме HTTP-запити до серверної частини через REST API, отримуючи у відповідь структуровані дані у форматі JSON. Така модель дає змогу швидко реагувати на дії користувача, оновлювати контент без повного перезавантаження сторінки та забезпечувати інтерактивність, що є важливим для навчальних застосунків.

Серверна частина (back-end) відповідає за функції обробки запитів, управління користувачами, створення й перевірки завдань, збереження результатів, формування аналітичних звітів тощо. На цьому рівні реалізуються бізнес-правила

системи, логіка оцінювання та контроль доступу. Сервер також відповідає за комунікацію з базою даних, у якій зберігаються відомості про користувачів, завдання, відповіді студентів і результати перевірок.

Важливим елементом клієнт–серверної архітектури є централізоване зберігання та обробка даних. Завдяки цьому всі користувачі працюють із єдиною узгодженою інформаційною базою, а адміністратор може контролювати доступ і вносити зміни без потреби у втручанні з боку клієнтів. Сервер забезпечує такі послуги, як збереження файлів, доступ до ресурсів та управління користувацькими сесіями.

Ще однією перевагою архітектури є масштабованість, тобто здатність системи розширюватися відповідно до кількості користувачів і складності завдань. При збільшенні навантаження можна розширити потужність серверів (вертикальне масштабування) або додати нові клієнтські робочі станції (горизонтальне масштабування), не змінюючи основну логіку системи [18].

Архітектура також передбачає реалізацію механізмів безпеки. Завдяки використанню токенів автентифікації (JWT) та системи ролей (Student, Teacher, Admin) забезпечується контроль доступу до ресурсів і захист персональних даних користувачів. Сервер може відхиляти несанкціоновані запити, що гарантує збереження конфіденційної інформації та стабільну роботу системи.

Загальну структуру взаємодії між компонентами представлено на рисунку 2.2, де схематично показано клієнтську частину (React-додаток у браузері), серверну частину (Spring Boot із REST API, модулями безпеки та штучного інтелекту) і базу даних (PostgreSQL), між якими здійснюється обмін інформацією через HTTP-запити.

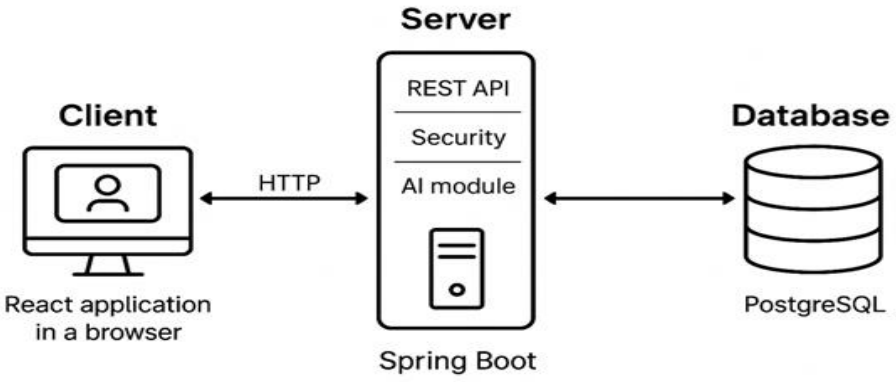

Рисунок 2.2. Архітектура вебзастосунку для створення та перевірки домашніх завдань

Таким чином, клієнт–серверна архітектура виступає оптимальним рішенням для розробки системи створення та перевірки домашніх завдань, оскільки забезпечує структуровану організацію процесів, гнучкість у розширенні та стабільну взаємодію між усіма компонентами вебзастосунку [18].

2.3 Проектування бази даних

БД є важливою складовою будь-якої системи, яка містить дані які повинні зберігатися. Саме БД у вебзастосунку для створення та перевірки домашніх завдань буде забезпечувати зберігання, обробку та доступ до навчальної інформації. Структура сховища даних повинна забезпечувати логічність та цілісність даних, а також забезпечувати можливість швидкого доступу для всіх користувачів відповідно до визначених ролей у системі.

Оскільки потрібно забезпечити надійність та можливість масштабування вебзастосунку, було обрано реляційну модель даних, реалізовану на основі системи управління базами даних PostgreSQL. Обрана СУБД забезпечує підтримку транзакцій, високу продуктивність, гнучкість у структурі даних і сумісність із

фреймворком Spring Data JPA, що спрощує інтеграцію із серверною частиною застосунку.

Проектування БД відбувалось спираючись на ключові принципи:

- Нормалізацію: зменшення надлишковості шляхом організації даних у пов'язані таблиці.
- Цілісність даних: забезпечення точності та узгодженості даних у всій базі даних.
- Масштабованість: проектування бази даних для ефективної обробки зростання обсягу даних.
- Оптимізація продуктивності: структурування даних для швидких запитів та пошуку.
- Гнучкість: можливість внесення змін або доповнень у майбутньому без критичних змін в коді[5].

Проектування бази даних здійснювалося з урахуванням функціональних вимог системи, що передбачає створення, виконання, перевірку та оцінювання домашніх завдань. Структура бази даних побудована таким чином, щоб відображати логіку взаємодії між трьома основними типами користувачів – студентом, викладачем та адміністратором, – а також підтримувати роботу модуля автоматичної перевірки завдань за допомогою технологій штучного інтелекту.

У моделі даних передбачено такі ключові сутності: користувачі системи (Users), навчальні завдання (Tasks), запитання тестового типу (Questions), подані студентами відповіді (Submissions), оцінки та відгуки (Grades, AI_Feedback).

Між цими таблицями встановлено зв’язки типу «один-до-багатьох» (1:N) та «один-до-одного» (1:1), що забезпечує логічну узгодженість даних.

Концептуальна схема взаємозв’язків між сутностями представлена на ER-діаграмі (див. *Додаток А*). На ній відображено основні таблиці бази даних та зв’язки між ними, які формують логічну основу майбутньої реалізації. Кожна таблиця має унікальний первинний ключ та визначені зовнішні ключі для забезпечення цілісності даних.

2.4 Проектування інтерфейсу користувача

Інтерфейс користувача є важливим елементом будь-якого вебзастосунку, оскільки саме через нього користувач взаємодіє із системою. Основною метою проектування інтерфейсу є створення зручного, інтуїтивно зрозумілого та естетично привабливого середовища [23], яке дозволить користувачам ефективно виконувати навчальні та організаційні завдання.

У процесі проектування інтерфейсу вебзастосунку для створення та перевірки домашніх завдань було застосовано низку базових принципів UI/UX-дизайну.

1. Принцип ієрархії.

Інтерфейс побудований із чітким візуальним упорядкуванням елементів: назви розділів і ключові кнопки («Надіслати завдання», «Перевірити») виділені збільшеним шрифтом і контрастним кольором. Це забезпечує швидке орієнтування користувача та акцентує увагу на основних діях.

2. Принцип послідовності.

Усі елементи мають єдиний стиль – кнопки, поля вводу та повідомлення оформлені однаково на всіх сторінках. Це створює відчуття стабільності та знижує когнітивне навантаження під час роботи.

3. Принцип контрасту.

Для акцентування важливих дій використано контрастні кольори: активні кнопки яскраві, допоміжні – нейтральні. Такий підхід підвищує зручність і запобігає помилковим натисканням.

4. Принцип вирівнювання.

Компоненти розміщено за сітковою структурою, що формує впорядкований і професійний вигляд інтерфейсу, підвищує читабельність і логічність сприйняття [22].

Застосування зазначених принципів забезпечує інтуїтивність, зручність та естетичну узгодженість інтерфейсу вебзастосунку.

Інтерфейс вебзастосунку побудований за принципами модульності, інтерактивності та адаптивності, що є характерними для сучасних веб-систем. Для

реалізації клієнтської частини використано бібліотеку React, яка дозволяє створити односторінковий застосунок (Single Page Application, SPA). Такий підхід забезпечує динамічне оновлення контенту без перезавантаження сторінок, що підвищує швидкодію системи та зручність користувачів.

Кожен елемент інтерфейсу має бути реалізований у вигляді окремого React-компонента, що відповідає за конкретну функцію: форму входу, список завдань, панель викладача, сторінку перевірки робіт тощо. Це спростить масштабування застосунку, полегшить повторне використання компонентів і сприятиме підтримці чистої архітектури фронтенду.

Інтерфейс користувача буде розділено на три основні частини відповідно до ролей у системі:

— Інтерфейс студента забезпечує перегляд доступних завдань, подання відповідей, перегляд результатів перевірки та отримання коментарів від викладача чи AI-модуля.

— Інтерфейс викладача міститить функції створення, редагування та перевірки завдань, а також генерацію звітів успішності студентів.

— Інтерфейс адміністратора надає засоби керування користувачами, моніторинг активності системи та контроль ролей.

Для побудови візуальної частини було обрано бібліотеки Material UI та Tailwind CSS, які забезпечують сучасний зовнішній вигляд, адаптивну верстку й уніфікований стиль. Дизайн повинен бути реалізований у спокійній академічній палітрі синіх і білих відтінків, що підкреслює професійність інтерфейсу.

Система навігації реалізована за допомогою React Router, дозволяє швидко перемикатися між основними сторінками застосунку: вхід, реєстрація, перегляд завдань, подані роботи, панель викладача чи адміністратора. Інтерактивні повідомлення про статус операцій (успіх, помилка, попередження) реалізовано через бібліотеку React-Toastify, що покращує зворотний зв’язок із користувачем.

Проектування інтерфейсу користувача здійснювалося з урахуванням типових сценаріїв взаємодії (user flow): реєстрація нового користувача, авторизація, перегляд завдань, подання відповідей, перевірка викладачем та формування

звітності, що дозволить забезпечити логічність і безперервність усіх робочих процесів у межах системи.

Для наочного відображення цих процесів на рисунку 2.3 подано схему навігації користувача, яка демонструє основні переходи між екранами інтерфейсу залежно від ролі користувача у системі – студента, викладача або адміністратора. Схема відображає послідовність дій користувача від моменту входу в систему до завершення роботи.

Рисунок 2.3. Схема навігації користувача у вебзастосунку для створення та перевірки домашніх завдань

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ВЕБ-ЗАСТОСУНКУ ДЛЯ СТВОРЕННЯ ТА ПЕРЕВІРКИ ДОМАШНІХ ЗАВДАНЬ

3.1 Структура та основні модулі застосунку

Розроблений вебзастосунок для створення та перевірки домашніх завдань реалізує комплексний проект системи керування навчальними завданнями, побудований за принципами багаторівневої клієнт-серверної архітектури (multi-tier architecture). Використання такого підходу у розробці програмного забезпечення для застосунку забезпечує логічне розмежування компонентів, покращує масштабованість і гнучкість системи, а також допомагає спростити підтримку системи.

Загальна структура системи побудована згідно з принципами багаторівневої архітектури, що означає, що застосунок поділено на окремі логічні рівні з чітким визначенням їх функцій та зон відповідальності.

Архітектура вебзастосунку складається з трьох основних логічних рівнів:

1. Frontend (React.js) – відповідальність за інтерфейс користувача, відображення даних, обробку подій, а також взаємодію із сервером через REST API.
2. Backend (Spring Boot) – відповідає за реалізацію бізнес-логіки, обробку запитів користувачів, здійснення перевірки доступу та взаємодії з БД.
3. Database (PostgreSQL/H2) – відповідає за зберігання усіх основних даних: користувачів, завдань, відповідей, оцінок, аналітичної інформації.

Міжрівневий зв’язок забезпечено з допомогою REST API, що дозволяє клієнтській частині бути незалежною від серверної, що дозволить в майбутньому (за потреби) масштабувати систему або ж змінити окремі модулі без втрати сумісності.

На рисунку 3.1 подано загальну архітектуру вебзастосунку,яка демонструє взаємодію основних компонентів системи: клієнтський інтерфейс, серверна логіка та БД.

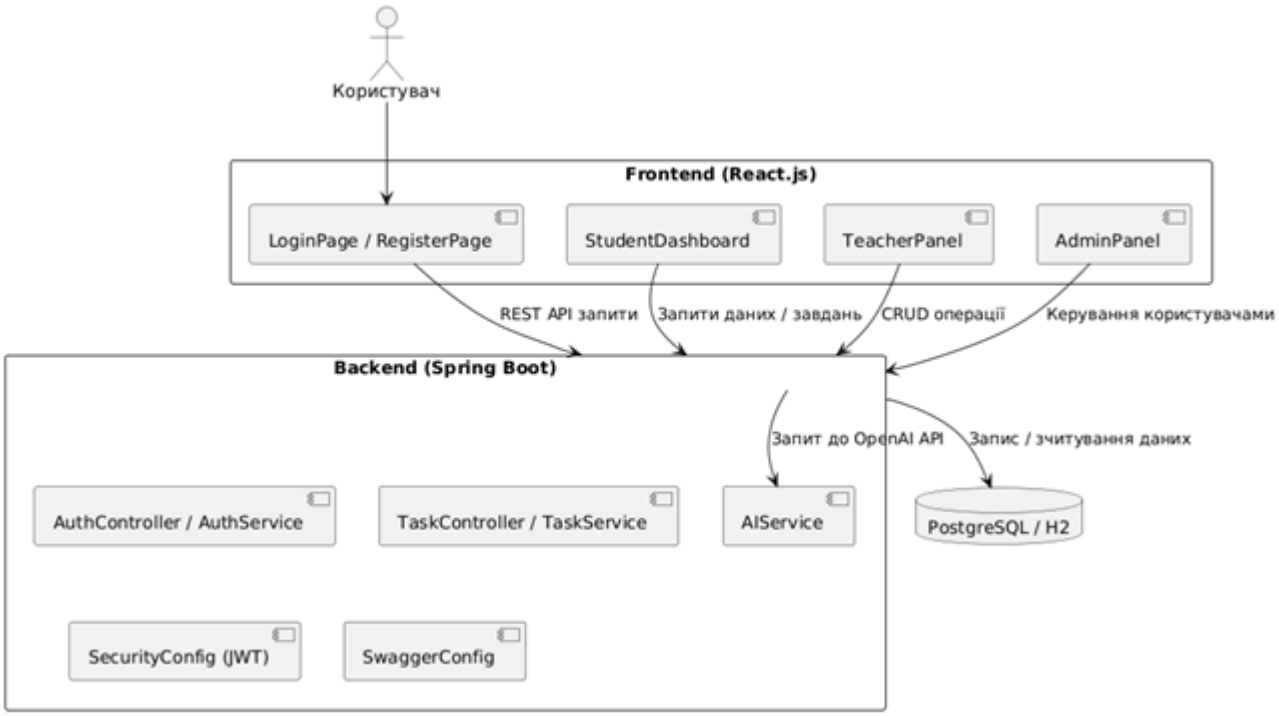

Рисунок 3.1. Архітектура вебзастосунку для створення та перевірки домашніх завдань

Після окреслення загальної структури системи доцільно докладніше розглянути впровадження кожного з рівнів. Почнемо з серверного рівня, оскільки саме він забезпечує основну бізнес-логіку, безпеку, обробку запитів користувачів та взаємодію з БД.

Серверна частина системи розроблена на платформі Spring Boot, що забезпечує швидку побудову надійних REST API-сервісів.

Структура проекту реалізована за принципом розділення відповідальності (Separation of Concerns), що виражається у використанні багатошарової архітектури. Це забезпечує логічне групування класів і конфігурацій за їх функціональним призначенням.

На рисунку 3.2 наведено структуру пакунків серверної частини вебзастосунку

На рисунку показано структуру основних директорій серверної частини проекту.

1. Пакунок *config* містить конфігураційні класи безпеки, інтеграції зі Swagger та OpenAI;
2. *controller* – REST API контролери;
3. *dto* – об’єкти передавання даних;
4. *entity* – сутності бази даних;
5. *projection* – об’єкти відображення даних у вигляді скорочених представлень;
6. *repository* – інтерфейси доступу до БД;
7. *security* – реалізація JWT-аутентифікації та сервісів користувачів;
8. *service* – бізнес-логіка застосунку.
9. Головний клас *HomeworkappApplication* ініціалізує роботу всієї серверної частини.

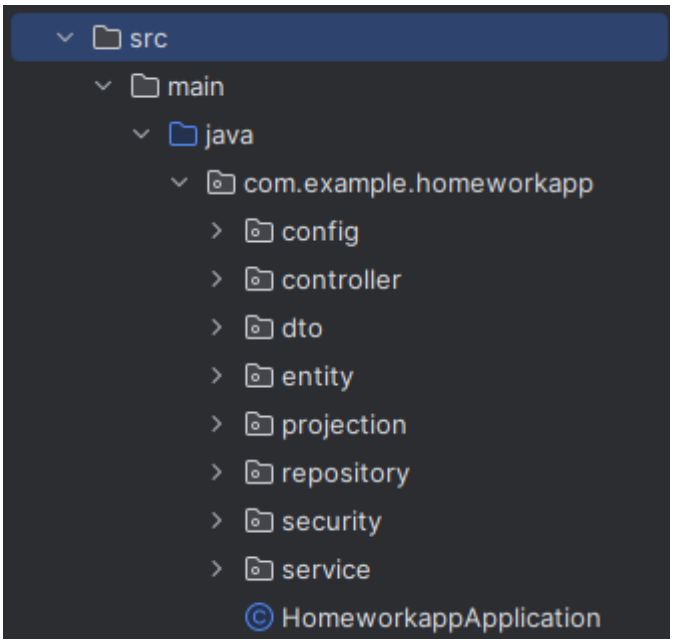

Рисунок 3.2. Структура пакунків серверної частини вебзастосунку для створення та перевірки домашніх завдань

Після деталізації структури пакунків варто зосередитись на взаємозв’язках між основними сутностями, сервісами та компонентами серверної частини застосунку. Для цього була побудована UML-діаграма класів, яка дозволяє візуально показати об'єктну модель, механізми взаємодії між сутностями та сервісними класами, які відповідають за авторизацію, перевірку та обробку завдань.

На Рисунку 3.3. зображено UML-діаграму класів бекенду у веб-застосунку для створення та перевірки домашніх завдань.

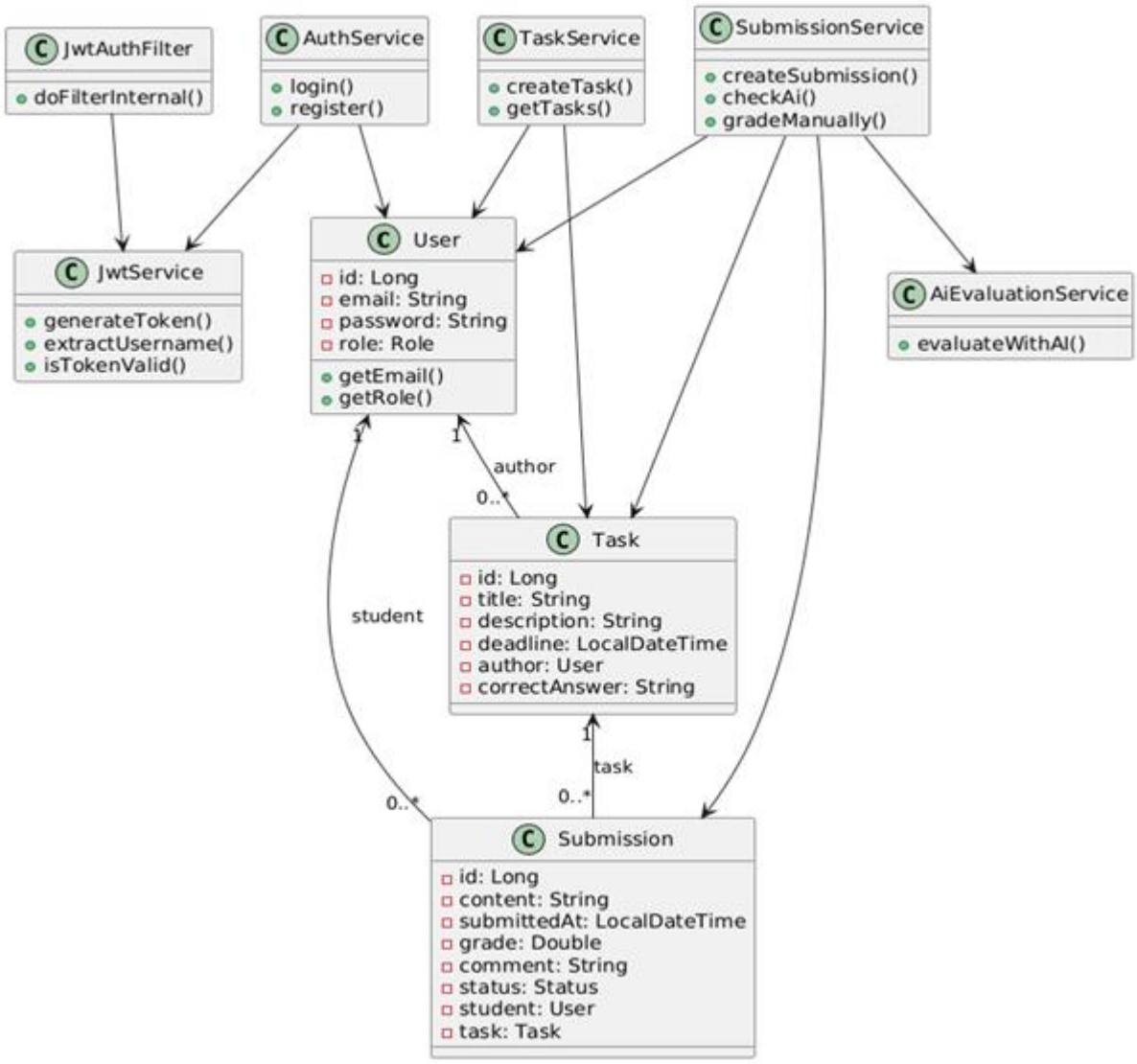

Рисунок 3.3. UML-діаграма класів вебзастосунку

В Таблиці 3.1 показано основні модулі бекенду та функції, які вони виконують.

Серверна частина підтримує JWT-аутентифікацію, хешування паролів (BCrypt), а також захист від CSRF і CORS, що гарантує безпечну взаємодію користувачів із системою.

Таблиця 3.1

Модуль	**Призначення**
AuthService	Реєстрація користувачів, авторизація, генерація JWT-токенів.
TaskService	CRUD-операції з навчальними завданнями.
SubmissionService	Обробка рішень студентів, збереження результатів перевірки.
AIService	Інтеграція з OpenAI API для автоматичної оцінки коротких відповідей.
TeacherService	Формування звітів і аналітики успішності студентів.
AdminController	Керування користувачами та ролями.
SecurityConfig	Налаштування Spring Security, ролей і доступів.
SwaggerConfig	Генерація документації API.

Аби забезпечити зручну та інтуїтивну взаємодію користувачів із системою, серверна частина інтегрована з клієнтським інтерфейсом, який реалізовано за допомогою сучасної бібліотеки React.js. Фронтенд відповідає за відображення інформації, обробку подій та передачу даних до бекенду через REST API.

Клієнтська частина вебзастосунку реалізована у вигляді односторінкового застосунку (SPA, Single Page Application), що забезпечує динамічне оновлення контенту без повного перезавантаження сторінки.

Основні компоненти і функції, які вони реалізують у системі відображені в таблиці 3.2.

Для стилізації застосовано **Material UI** та **Tailwind CSS**, що забезпечує адаптивний, сучасний і зручний дизайн. Комунікацію з бекендом реалізовано через бібліотеку **Axios**, яка виконує HTTP-запити до REST API.

Таблиця 3.2

Компонент	**Призначення**
LoginPage / RegisterPage	Авторизація та реєстрація користувачів через API /auth.
StudentDashboard	Перегляд завдань, подання відповідей, отримання результатів перевірки.
TeacherPanel	Створення та редагування завдань, перевірка робіт, аналітика успішності.
AdminPanel	Керування користувачами, створення нових викладачів.
AnalyticsPage	Візуалізація динаміки успішності за допомогою Chart.js.
AIEvaluationModal	Відображення результатів автоматичного оцінювання AI.
ProtectedRoute	Забезпечення доступу лише авторизованим користувачам (JWT-перевірка).

Для зберігання даних використовується реляційна база PostgreSQL, а під час локальної розробки – вбудована H2 Database.

База містить таблиці:

users – облікові записи користувачів (email, пароль, роль);

tasks – навчальні завдання з дедлайнами та описом;

submissions – відповіді студентів і результати перевірок;

ai_feedback – коментарі та оцінки від модуля AI;

grades – оцінки для аналітичних звітів.

3.2 Реалізація функціональних можливостей системи.

Розроблений вебзастосунок для створення та перевірки домашніх завдань реалізує повний цикл керування навчальними завданнями, починаючи від їх генерації користувачем з роллю “Викладач”, закінчуючи оцінюванням та аналітичним підрахунком результатів. Функціональні можливості системи побудовані за допомогою архітектури REST API, яка забезпечує чіткий розподіл ролей і стабільну взаємодію між клієнтською та серверною частинами.

Реєстрація та авторизація користувачів

Система підтримує створення облікових записів, автентифікацію та розмежування доступу до функцій системи на основі визначених ролей користувачів. Реалізація базується на фреймворку Spring Security з використанням JWT (JSON Web Token) для захищеної аутентифікації.

Реєстрація користувачів здійснюється через форму у React-інтерфейсі, де потрібно зазначити адресу електронної пошти та пароль. Для забезпечення безпеки системи пароль шифрується за допомогою алгоритму BCrypt. У системі визначено автоматичне призначення ролі STUDENT для нових користувачів. Створення користувачів із ролями TEACHER або ADMIN дозволено лише користувачу, який має права ADMIN.

Після успішного входу система генерує JWT-токен, який містить email, роль користувача та час дії. Токен передається у клієнтську частину й зберігається у локальному сховищі, що забезпечує безпечну авторизацію при наступних запитах.

На Рисунку 3.4 показано фрагмент класу *AuthService,* який реалізує створення користувача (*register()*) та генерацію JWT-токена після успішної авторизації в системі (*generateToken(user)*).

У випадку успішної авторизації система автоматично перенаправляє користувача до відповідного інтерфейсу відповідно до визначеної йому ролі:

— студента – на сторінку *StudentDashboard* (адреса */student*),

— викладача – на сторінку *TeacherPanel* (адреса */teacher*),

— адміністратора – на сторінку *AdminPanel* (адреса */admin*)

```
public AuthResponse register(RegisterRequest request) {  OlenaSheremet *
    if (userRepository.findByEmail(request.email()).isPresent()) {
        throw new RuntimeException("Користувач з такою електронною поштою вже існує");
    }

    var user = User.builder()
            .email(request.email())
            .password(passwordEncoder.encode(request.password()))
            .role(User.Role.STUDENT)
            .build();
    userRepository.save(user);        // створення і збереження користувача

    String token = jwtService.generateToken(user);
    return new AuthResponse(token);
}

public AuthResponse authenticate(AuthRequest request) {  OlenaSheremet *
    authenticationManager.authenticate(
            new UsernamePasswordAuthenticationToken(request.email(), request.password())
    );

    var user = userRepository.findByEmail(request.email())
            .orElseThrow(() -> new RuntimeException("Користувача не знайдено"));

    String token = jwtService.generateToken(user);  //генерація токена
    return new AuthResponse(token);
}
```

Рисунок 3.4. Фрагмент класу *AuthService*

Також варто зазначити, що для забезпечення постійної автентифікації під час виконання запитів у системі реалізовано спеціальний фільтр *JwtAuthenticationFilter* (Рисунок 3.5), який перевіряє правильність, строк дії та відповідність токена поточному користувачу.

Цей фільтр перехоплює кожен HTTP-запит, перевіряє наявність заголовка Authorization з токеном формату Bearer <token> і видобуває з нього ім'я користувача. Якщо токен дійсний, фільтр завантажує дані користувача, створює об’єкт автентифікації та встановлює його в контекст безпеки Spring, що дозволяє системі розпізнати авторизованого користувача під час виконання подальших

запитів. Після підтвердження автентифікації запит продовжує обробку в ланцюжку фільтрів.

```java
@Override  no usages  OlenaSheremet
protected void doFilterInternal(HttpServletRequest request,
                                HttpServletResponse response,
                                FilterChain filterChain)
        throws ServletException, IOException {

    final String authHeader = request.getHeader( s: "Authorization");

    if (authHeader == null || !authHeader.startsWith("Bearer ")) {
        filterChain.doFilter(request, response);
        return;
    }

    final String token = authHeader.substring( beginIndex: 7);
    final String username = jwtService.extractUsername(token);

    if (username != null && SecurityContextHolder.getContext().getAuthentication() == null) {
        var userDetails = userDetailsService.loadUserByUsername(username);

        var authToken = new UsernamePasswordAuthenticationToken(
                userDetails,  credentials: null, userDetails.getAuthorities()
        );
        authToken.setDetails(
                new WebAuthenticationDetailsSource().buildDetails(request)
        );

        SecurityContextHolder.getContext().setAuthentication(authToken);
    }

    filterChain.doFilter(request, response);
}
```

Рисунок 3.5. Фрагмент класу *JwtAuthenticationFilter* (метод *doFilterInternal()*)

Функціонал викладача

Викладач має доступ до функціоналу системи, який відповідає за створення, редагування та перевірки навчальних завдань.

У панелі TeacherPanel реалізовано наступні дії, які може виконувати викладач:

— створення нових завдань із зазначенням типу, опису та дедлайну;

— редагування та видалення завдань після завершення навчального періоду;

— перевірка поданих студентами робіт та виставлення оцінок вручну або з використанням AI-модуля.

Причому, кожне завдання зберігається у таблиці *tasks*, а відповіді студентів – у таблиці *submissions*. Для відображення результатів і статистики застосовується сервіс *TeacherService*, який формує аналітичні дані у форматі JSON.

Функціонал студента

Після авторизації користувача з роллю STUDENT перенаправляє до панелі *StudentDashboard*, де можна отримати перелік активних завдань із коротким описом, терміном виконання та корисними для виконання файлами чи посилання визначені і додані викладачем.

У системі передбачено подання відповіді у вигляді тексту або завантаженого файлу. Відповідь надсилається на сервер через POST-запит до */api/submissions*, після чого зберігається у базі даних.

Функціонал адміністратора

Адміністратор має доступ до панелі *AdminPanel*, яка дозволяє повноцінно управляти користувачами системи. До основних функцій належать створення нових облікових записів (у тому числі викладачів і адміністраторів), зміна ролей та видалення користувачів. Усі адміністративні операції реалізовано через REST-контролер *AdminUserController*, доступ до якого обмежено лише для ролі **ADMIN** за допомогою механізмів Spring Security.

У Таблиці 3.3 наведено запити, які може виконувати адміністратор та маршрути через які вони виконуються.

Таблиця 3.3

Маршрут	**Призначення**
POST */api/admin/create-user*	Створення нового користувача
GET */api/admin/users*	Отримання списку всіх користувачів
PUT */api/admin/change-role/{id}*	Зміна ролі користувача

DELETE */api/admin/delete/{id}*	Видалення користувача з системи

Аналітика та звітність

Система реалізує аналітичний модуль для узагальнення та систематизації результатів навчальної діяльності студентів на основі даних, зібраних у таблицях *SUBMISSION, TASK* та *USERS*. Робота цього модулю реалізована у сервісах *TeacherService* та *ReportController,* які формують статистичні показники в режимі реального часу.

До основних показників, які входять до аналітики належать:

— кількість зданих і перевірених робіт;

— середній бал студента;

— динаміка успішності у вигляді послідовності отриманих оцінок.

Аналітика реалізована на основі сервісів *TeacherService* та *ReportController*.

Сервіс *TeacherService* обчислює основні статистичні показники, визначаючи загальну кількість зданих і перевірених робіт, середній бал та рейтинг студентів. На основі даних про виконання завдань сервіс також формує динаміку успішності, що відображає зміну результатів студента у часі.

Формування звітів здійснюється в ReportController, який надає аналітичні дані у форматах JSON, PDF та Excel. Система дозволяє отримувати індивідуальні звіти студентів та зведені рейтинги групи, забезпечуючи викладача структурованою інформацією для оцінювання результатів та моніторингу навчального процесу.

Система сповіщень та допоміжні функції

Для покращення взаємодії з користувачем реалізовано систему сповіщень та повідомлень, які виконують наступне:

— нагадування про наближення дедлайну;

— сповіщення про нові завдання або результати перевірки;

— підтвердження успішного надсилання відповіді.

Повідомлення відображаються через бібліотеку *React-Toastify*, а їхнє оновлення відбувається у режимі реального часу.

3.3 Інтерфейс користувача та тестування системи

Інтерфейс користувача

Інтерфейс вебзастосунку для створення та перевірки домашніх завдань розроблено з використанням технологій React.js, Material UI та CSS3, які дали змогу забезпечити сучасний, мінімалістичний і адаптивний дизайн. Основна мета інтерфейсу – забезпечити інтуїтивну взаємодію користувачів із системою, зручність у навігації та узгодженість візуального вигляду для всіх ролей користувачів.

Інтерфейс побудований за принципом односторінкового застосунку: перед завантаженням будь-яких сторінок компонент *ProtectedRoute* з бібліотеки React перевіряє отриманий від серверу JWT-токен. У випадку його відсутності або завершення строку дії токена користувача перенаправляє на сторінку входу.

Сторінка входу (Login Page)

Форма для авторизації користувачів розташована по центру екрана та містить поля для введення електронної пошти й пароля. Також наявні кнопки «Увійти» і «Зареєструватися», які власне і реалізують навігацію між сторінками.

На сторінці авторизації користувач надсилає запит до */api/auth/login*, після чого у разі успіху отримує JWT-токен та роль, яка визначає подальший маршрут. У випадку невірних даних інтерфейс відображає повідомлення з помилкою, а сервер повертає статус 403, як реалізовано у *AuthService*.

Сторінка реєстрації (Register Page)

Містить поля Email, Пароль та Підтвердити пароль із клієнтською валідацією мінімальної довжини.

Під час реєстрації запит надсилається на */api/auth/register*, а всі нові облікові записи автоматично отримують роль STUDENT. Інтерфейс містить клієнтську валідацію полів і перевіряє збіг пароля та підтвердження пароля. Серверна частина

створює користувача через AuthService, застосовуючи хешування пароля за допомогою BCrypt.

Після успішної реєстрації користувач автоматично отримує роль *STUDENT* і перенаправляється на сторінку авторизації.

Кабінет студента (Student Dashboard)

У верхній частині сторінки, яка відображається після авторизації користувача з роллю *STUDENT* міститься панель навігації з ім'ям користувача та кнопкою «Вийти».

Основна частина відображає список завдань у вигляді карток, що містять назву, дедлайн, а також статус виконання. Дедлайни позначаються кольорами: зелений – активне завдання, жовтий – термін наближається, червоний – прострочене.

Після вибору завдання відкривається форма подання відповіді, де студент може ввести текст або прикріпити файл.

Після перевірки результат відображається у вигляді оцінки та коментаря від викладача чи AI-модуля.

Панель викладача (Teacher Panel)

Для авторизованого користувача з роллю *TEACHER* доступна сторінка де у лівій частині інтерфейсу міститься вертикальне меню з розділами: *Завдання, Результати*, *Аналітика* та *Профіль*.

У розділі «Завдання» доступна кнопка «Додати завдання», що відкриває модальне вікно створення нового завдання.

Для користувача доступна можливість переглядати, редагувати чи видаляти завдання, а також бачити кількість поданих студентами відповідей.

Розділ «Результати» містить таблицю із поданими роботами, датами та оцінками.

У вкладці «Аналітика» дані візуалізуються у вигляді графіків і діаграм, створених за допомогою бібліотеки Chart.js.

Панель адміністратора (Admin Panel)

Панель адміністратора, яка відображається після авторизації відповідного користувача, надає інструменти для керування користувачами системи.

Основний елемент – таблиця з фільтрацією за ролями (*STUDENT*, *TEACHER*, *ADMIN*).

Для адміністраторів передбачено наступні можливі дії в системі:

— створення нового викладача через форму введення email і пароля;

— зміна ролей користувачів;

— видалення облікових записів;

— формування звітів.

Тестування системи

Тестування є завершальним етапом розробки, який забезпечує стабільність, безпеку та правильну роботу всіх її модулів. Воно дозволяє перевірити, як працює серверна та клієнтська частина, виявити помилки перед реалізацією системи в навчальному середовищі та підтвердити, що реалізований функціонал відповідає технічним вимогам. Особливу увагу було приділено тестуванню важливих сервісів – аутентифікації, обробки завдань та модуля автоматичної перевірки.

Для того аби забезпечити стабільність роботи основних модулів системи, коректного функціонування REST API, а також забезпечити зручність використання інтерфейсу вебзастосунку для створення та перевірки домашніх завдань було проведено тестування системи по 4 типах тестів.

У Таблиці 3.4 наведено типи тестування, їх основне призначення, а також інструменти, які були використані для проведення тестувань.

Таблиця 3.4

Тип тестування	**Призначення**	**Використані інструменти**
Unit-тести	Перевірка роботи окремих сервісів (AuthService, AIService, TaskService).	JUnit 5

Інтеграційні тести	Перевірка взаємодії REST API, бази даних і безпеки.	Spring Boot Test, MockMvc
API-тести	Тестування запитів і відповідей REST API.	Postman, Swagger UI
UI-тести	Перевірка відображення інтерфейсу та сценаріїв користувача.	Selenium, Cypress

Тестування вебзастосунку відбувалося за кількома сценаріями, наведеними нижче.

Реєстрації користувача

У випадку успішної реєстрації нового користувача – код відповіді 201 Created. Для повторної спроби реєстрації за допомогою використання того свмого email – 409 Conflict.

Авторизація та перевірка JWT

У разі введення правильних облікових даних користувач отримує доступ до системи з кодом відповіді 200 OK. Якщо пароль введено неправильно – система повертає 403 Forbidden, а при спробі використати прострочений токен – 401 Unauthorized.

Особливу увагу приділено тестуванню контролю доступу (Рисунок 3.6). Модульні тести показали, що користувачі із роллю STUDENT та TEACHER не мають доступу до адміністративних маршрутів /api/admin/create-user, тоді як ADMIN успішно виконує відповідні операції, що підтверджує коректність роботи механізму *@PreAuthorize*.

CRUD-операцій над завданнями

Створення, редагування та видалення завдань дозволено лише користувачам із ролями TEACHER або ADMIN. Роль STUDENT має обмежений доступ і не може виконувати ці дії.

Подання відповідей студентом

Після завантаження файлу або тексту система успішно зберігає спробу виконання в таблиці *submission* з кодом відповіді 200 OK, що підтверджує коректну обробку даних.

AI-перевірка завдань

Під час звернення до OpenAI API запит має завершуватися успішно, після чого система отримує оцінку та коментар, які зберігаються у базі даних для подальшого аналізу.

```
public class AdminUserControllerAccessTest {

    private MockMvc mockMvc;

    @BeforeEach
    void setup() {
        UserRepository userRepository = Mockito.mock(UserRepository.class);
        PasswordEncoder encoder = Mockito.mock(PasswordEncoder.class);

        AdminUserController controller = new AdminUserController(userRepository, encoder);

        mockMvc = standaloneSetup(controller)
                .build();
    }

    @Test
    @WithMockUser(roles = "STUDENT")
    void studentCannotAccessAdminEndpoints() throws Exception {
        mockMvc.perform(post( uriTemplate: "/api/admin/create-user"))
                .andExpect(status().isForbidden());
    }

    @Test
    @WithMockUser(roles = "TEACHER")
    void teacherCannotAccessAdminEndpoints() throws Exception {
        mockMvc.perform(post( uriTemplate: "/api/admin/create-user"))
                .andExpect(status().isForbidden());
    }

    @Test
    @WithMockUser(roles = "ADMIN")
    void adminCanAccess() throws Exception {
        mockMvc.perform(post( uriTemplate: "/api/admin/create-user")
                        .content("{\"email\":\"t@test.com\",\"password\":\"123\",\"role\":\"TEACHER\"}")
                        .contentType("application/json"))
                .andExpect(status().isOk());
    }
}
```

Рисунок 3.6. Unit-тест контролю доступу (AdminUserControllerAccessTest)

Аналітика викладача

Система повинна коректно обчислювати середні бали, рейтинг студентів і рівень узгодженості оцінок між AI та викладачем, забезпечуючи достовірність аналітичних звітів.

Метою тесту є перевірка механізму контролю доступу, реалізованого на основі Spring Security. Тест імітує виконання HTTP-запитів від користувачів із різними ролями (STUDENT, TEACHER, ADMIN) та перевіряє, чи коректно застосовуються обмеження, визначені анотацією *@PreAuthorize("hasRole('ADMIN')")*.

У методі *setup()* створюється автономне тестове середовище на базі MockMvc, де контролер *AdminUserController* запускається із замоканими залежностями: *UserRepository* та *PasswordEncoder*. Це дозволяє протестувати логіку контролера без запуску повного Spring-контексту.

Тест *studentCannotAccessAdminEndpoints()* виконується з імітацією користувача ролі STUDENT *(@WithMockUser(roles = "STUDENT")*). При спробі виконати POST-запит на */api/admin/create-user* очікується відповідь **403 Forbidden**, що підтверджує заборону доступу.

Аналогічно тест *teacherCannotAccessAdminEndpoints()* перевіряє, що користувач з роллю TEACHER також отримує **403 Forbidden** при зверненні до адміністративного маршруту.

Тест *adminCanAccess()* моделює користувача з роллю ADMIN. У цьому випадку POST-запит успішно виконується, а відповідь сервера має статус 200 OK, що доводить коректність налаштування прав доступу для адміністратора.

Виконання тесту підтверджує:

1. STUDENT і TEACHER не мають доступу до адміністративних функцій.
2. ADMIN може успішно створювати нового користувача.
3. Налаштування безпеки застосовані коректно, а анотація @PreAuthorize працює відповідно до логіки системи ролей.

Для підтвердження стабільності аналітичного модуля проведено окремі тести генерації PDF- та Excel-звітів. Перевірено MIME-типи відповідей, правильність заголовків Content-Disposition, структуру створених документів та обробку випадків, коли студент не має жодної поданої роботи (Рисунок 3.7 та Рисунок 3.8).

Тест *PdfReportServiceTest* перевіряє коректність роботи сервісу *PdfReportService*, який генерує PDF-звіти для студентів на основі статистичних показників. Мета – переконатися, що сервіс формує непорожній PDF-документ із валідним вмістом без запуску всього застосунку.

```
public class PdfReportServiceTest {  new *

    private final PdfReportService pdfReportService = new PdfReportService();  1 usage

    @Test  new *
    void testGeneratePdf() {
        byte[] pdf = pdfReportService.generateStudentReport(
                email: "student@mail.com",
                avg: 85.0,
                total: 10,
                checked: 8
        );

        assertNotNull(pdf);
        assertTrue( condition: pdf.length > 100); // PDF має бути не порожній
    }
}
```

Рисунок 3.7. Тестування PDF-звітів *(PdfReportServiceTest)*

У тесті створюється екземпляр сервісу *PdfReportService* без додаткових залежностей. Далі метод *generateStudentReport()* викликається з тестовими даними: email студента, середній бал, кількість поданих робіт і кількість перевірених викладачем.

Отриманий PDF передається у змінну pdf, після чого виконуються два ключові твердження, а саме *assertNotNull(pdf)* підтверджує, що сервіс повернув валідний масив байтів, а *assertTrue(pdf.length > 100)* підтверджує, що PDF має зміст і не є порожнім файлом.

Результат виконання тесту *PdfReportServiceTest* підтверджує наступне:

1. Генерація PDF працює без помилок.
2. Сервіс створює структурований документ.

3. Вихідний файл містить достатню кількість інформації.

Сервіс може працювати ізольовано, без взаємодії з БД чи іншими

```
public class ExcelReportTest {

    static class RatingImpl implements RatingProjection {
        private final String email;
        private final Double avgGrade;
        private final Long countSub;

        RatingImpl(String email, Double avgGrade, Long countSub) {
            this.email = email;
            this.avgGrade = avgGrade;
            this.countSub = countSub;
        }

        @Override public String getEmail() { return email; }
        @Override public Double getAvgGrade() { return avgGrade; }
        @Override public Long getCountSub() { return countSub; }
    }

    @Test
    void testExcelGeneration() throws Exception {
        List<RatingProjection> ratings =
                List.of(new RatingImpl( email: "s@mail.com",  avgGrade: 90.0,  countSub: 5L));

        Workbook workbook = new XSSFWorkbook();
        Sheet sheet = workbook.createSheet( s: "Ratings");

        Row header = sheet.createRow( i: 0);
        header.createCell( i: 0).setCellValue("Email");
        header.createCell( i: 1).setCellValue("Середній бал");
        header.createCell( i: 2).setCellValue("Кількість завдань");

        Row row = sheet.createRow( i: 1);
        row.createCell( i: 0).setCellValue("s@mail.com");
        row.createCell( i: 1).setCellValue(90.0);
        row.createCell( i: 2).setCellValue(5);

        ByteArrayInputStream input = new ByteArrayInputStream(
                workbookToBytes(workbook)
        );

        Workbook result = new XSSFWorkbook(input);
        Sheet resultSheet = result.getSheetAt( i: 0);

        assertEquals( expected: "Email", resultSheet.getRow( i: 0).getCell( i: 0).getStringCellValue());
        assertEquals( expected: "s@mail.com", resultSheet.getRow( i: 1).getCell( i: 0).getStringCellValue());
    }

    private byte[] workbookToBytes(Workbook workbook) throws Exception {
        var out = new java.io.ByteArrayOutputStream();
        workbook.write(out);
        return out.toByteArray();
    }
```

Рисунок 3.8. Тестування Excel-звітів *(ExcelReportTest)*

компонентами.

Метою тесту *ExcelReportTest* є перевірка коректності формування Excel-файлу зі студентськими рейтингами. Такий файл експортується викладачами в форматі XLSX, тому важливо, щоб структура таблиці відповідала бізнес-логіці та була правильною для реального використання.

Спочатку у тесті після запуску створюється тестова імплементація інтерфейсу RatingProjection, що містить email студента, середній бал та кількість завершених завдань. Ці дані імітують результат запиту до бази даних.

Далі на основі бібліотеки Apache POI створюється робоча книга XSSFWorkbook і робочий лист Ratings. На перший рядок записуються заголовки таблиці: Email, Середній бал, Кількість завдань. У другий рядок записуються тестові дані студента.

Створена таблиця перетворюється у масив байтів (імітація передачі Excel-файлу на клієнт).

Вміст таблиці повторно читається назад у новий Workbook, після чого виконуються твердження:

- відповідність заголовків початковим;
- коректність email;
- збіг середнього бала;
- збіг кількості виконаних завдань.

Виконання цього тесту підтверджує:

1. Excel-файл формується правильно.
2. Структура документа відповідає вимогам звітності.
3. Дані зберігаються у клітинках без втрати значень.
4. Формат XLSX підтримується й коректно читається.

Результати тестування підтвердили, що всі функції системи – від аутентифікації до аналітики – працюють коректно, а реалізовані модулі стійкі до помилок, некоректних даних та мережевих збоїв.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було досягнуто поставленої мети, а саме – розроблено вебзастосунок для створення та перевірки домашніх завдань, який реалізує сучасний підхід до автоматизації освітнього процесу. Система поєднує можливості керування навчальними завданнями, інтерактивної взаємодії між викладачем і студентом, аналітики успішності та автоматичного оцінювання за допомогою технологій штучного інтелекту. В ході роботи було вирішено поставлені завдання з аналізу предметної області, визначення вимог, проектування архітектури, реалізації основних компонентів і перевірки коректності функціонування системи.

На основі проведеного теоретичного аналізу досліджено сучасні тенденцій розробки вебзастосунків призначених для використання в освітньому процесі. Розглянуто основні архітектурні моделі (клієнт–серверну, трирівневу, мікросервісну) та обґрунтовано вибір багаторівневої клієнт–серверної архітектури (multi-tier architecture) як найбільш ефективної для реалізації освітніх систем із централізованим зберіганням даних. Такий підхід забезпечив гнучкість, масштабованість, логічне розмежування компонентів і спростив подальшу підтримку та розвиток програмного продукту.

У процесі проектування розроблено структуру системи, що включає UML-діаграми, модель бази даних, схеми взаємодії основних компонентів. Також на цьому етапі було визначено функціональні та нефункціональні вимоги до вебзастосунку. Особливу увагу приділено базі даних, реалізованій у системі управління PostgreSQL, яка забезпечує надійне зберігання користувацьких, навчальних і аналітичних даних. Для локального тестування було визначено використання вбудованої бази H2. Розроблена структура бази даних підтримує логічні зв’язки між користувачами, завданнями, поданими роботами, оцінками, коментарями та аналітичними даними. Для візуалізації структури була розроблена ER-модель, яка охопила основні сутності та зв'язки між ними.

У практичній частині роботи було реалізовано вебзастосунок, який побудований із застосуванням сучасного стеку технологій, таких як Java 21, Spring Boot, Spring Security, JWT, React.js, Material UI, Tailwind CSS та PostgreSQL. Серверна частина побудована на Spring Boot і реалізує повний набір REST-сервісів, включаючи модулі реєстрації та авторизації, керування завданнями, обробку поданих робіт, формування звітів та адміністративного керування користувачами. Налаштування Spring Security та механізму JWT дозволило забезпечити захищений доступ до ресурсів системи та чітке розмежування користувачів за ролями (STUDENT, TEACHER, ADMIN).

Окремим компонентом системи є аналітичний модуль, який здійснює розрахунок кількості зданих та перевірених робіт, середнього балу та формує PDF та Excel-звіти. Використання цього модулю дає змогу оперативно оцінювати успішність студентів, відстежувати динаміку результатів та забезпечити викладача об'єктивною інформацією для подальшого планування навчального процесу.

Клієнтська частина базується на технологіях React.js, Material UI та Tailwind CSS, що забезпечує адаптивний, інтуїтивно зрозумілий та структурований дизайн. Інтерактивну комфортну взаємодію користувача з системою без перезавантаження сторінок (SPA-підхід) зі значно підвищеною швидкодією забезпечено за допомогою реалізації окремих робочих панелей для студентів, викладачів та адміністратора. Інтерфейс підтримує валідацію форм, інтерактивне оновлення даних, відображення результатів перевірки та повідомлення про важливі події через систему сповіщень.

Тестування вебзастосунку охоплювало модульні, інтеграційні, API- та UI-тести, що підтвердили коректність реалізованої функціональності, стабільність роботи окремих модулів, правильність розмежування доступу та стійкість системи до некоректних даних і помилок зовнішніх сервісів. Окремо також було протестовано механізм контролю доступу та правильність генерації PDF- та Excel-звітів. Отримані результати підтвердили відповідність реалізації проектним вимогам і логічну завершеність системи.

Практичне значення розробленого вебзастосунку полягає в тому, що він може бути використаний як основа для створення повноцінної системи підтримки навчального процесу у вищих навчальних закладах, школах або онлайн-платформах, яка дозволить зменшити навантаження на викладачів і забезпечити студентів зручним інструментом виконання завдань. Вебзастосунок може бути також використаний як прототип для створення більш масштабних освітніх платформ або інтегрований у вже існуючі системи електронного навчання.

Перспективи подальшого розвитку системи передбачають розширення її функціональності шляхом інтеграції з іншими освітніми платформами такими як Google Classroom або Moodle, впровадження системи push-сповіщень через електронну пошту чи месенджери, додавання системи рекомендацій на основі штучного інтелекту та розроблення мобільної версії застосунку. Також можливим напрямом розвитку є створення адаптивної системи оцінювання, яка враховуватиме індивідуальні особливості навчання кожного користувача.

Отже, проведене дослідження та реалізований вебзастосунок повністю відповідають сформульованій меті та завданням роботи, демонструють практичну цінність запропонованих технічних рішень і підтверджують високу ефективність застосування сучасних вебтехнологій в освітньому процесі. Система є завершеним, функціональним та перспективним рішенням, що має значний потенціал для подальшого розвитку та інтеграції в цифрове освітнє середовище.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. A Beginner's Guide to HTML and CSS for Web Development. *Ironhack*. URL: https://www.ironhack.com/gb/blog/a-beginner-s-guide-to-html-and-css-for-web-development.
2. A Complete Guide to JavaScript Application Development. *Leanware*. URL: https://www.leanware.co/insights/javascript-application-development.
3. Advantages and Disadvantages of Three-Tier Architecture in DBMS. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/dbms/advantages-and-disadvantages-of-three-tier-architecture-in-dbms/.
4. Client-Server Architecture - System Design. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/system-design/client-server-architecture-system-design/.
5. Database Design Basics, Structure, and Principles. *Tadabase*. URL: https://tadabase.io/blog/database-design.
6. Databases in Web Development. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/web-tech/databases-in-web-development/.
7. Google Classroom Review: Pros And Cons Of Using Google Classroom In eLearning. *eLearning Industry*. URL: https://elearningindustry.com/google-classroom-review-pros-and-cons-of-using-google-classroom-in-elearning.
8. How backend and frontend connect and work together in web development. *Medium*. URL: https://nikhilsomansahu.medium.com/how-backend-and-frontend-connect-and-work-together-in-web-development-2c852bd1e9e6.
9. How to build a web app database?. *Acho*. URL: https://acho.io/blogs/how-to-build-web-app-database.
10. In simple terms: code on the backend, frontend and how they interact. *Code with Hugo*. URL: https://codewithhugo.com/in-simple-terms-code-on-the-backend-frontend-and-how-they-interact/#the-state-of-backend-frontend-interaction.
11. Key Technologies Behind Modern Web Applications and Websites. *DEV Community*. URL: https://dev.to/purvituvar/key-technologies-behind-modern-web-applications-and-websites-aen.
12. Pros and Cons of Moodle as an LMS Platform. *Course Orbit*. URL: https://courseorbit.com/blog/pros-and-cons-of-moodle-as-an-lms-platform/ .
13. The 5 Best Backend Development Languages to Master (2025). *Roadmaps*. URL: https://roadmap.sh/backend/languages.
14. Top Web Development Tools in 2025. *BrowserStack*. URL: https://www.browserstack.com/guide/web-development-tools.
15. Understanding HTML: The Foundation of Web Development. *ONELINE*. URL: https://oneline.ch/en/blog/understanding-html/.
16. Web application architecture: how to choose the best for your product. *MobiDev*. URL: https://mobidev.biz/blog/web-application-architecture-types.
17. What is client-server architecture? Everything you should know. *Simplilearn*. URL: https://www.simplilearn.com/what-is-client-server-architecture-article.

18. What Is Client-Server Architecture?. *Coursera*. URL: https://www.coursera.org/articles/client-server-architecture.
19. What is monolithic architecture in software?. *informa techtarget*. URL: https://www.techtarget.com/whatis/definition/monolithic-architecture.
20. What is the difference between front-end and back-end development?. *Concepta*. URL: https://www.conceptatech.com/blog/difference-front-end-back-end-development.
21. What Is Data Exchange?. *IBM*. URL: https://www.ibm.com/think/topics/data-exchange.
22. 7 Key UI Design Principles + How To Use Them. *Figma*. URL: https://www.figma.com/resource-library/ui-design-principles/.
23. 12 iconic web app design examples for 2024. *Softr*. URL: https://www.softr.io/blog/web-app-design-examples.
24. Архітектура вебзастосунків 2024: ультимативний гайд для розробників. *robot_dreams*. URL: https://robotdreams.cc/uk/blog/567-arhitektura-vebzastosunkiv.
25. Безпека веб-сайтів. загрози безпеці веб-сайтів та поради щодо захисту від шкідливих атак. *OniStudio*. URL: https://www.onistudio.com.ua/bezpeka-veb-sajtiv/.
26. Веб-додаток і його характеристики. *Centum-D*. URL: https://centum-d.ua/veb-dodatok-yogo-harakteristiki/.
27. Веб-додаток: що це таке, типи, переваги, принцип роботи. *Megasite*. URL: https://megasite.ua/ua/veb-prilogenie-chto-eto-takoe-tipi-preimushchestva-printsip-raboti.
28. Новицька В. Педагогічні читання «Використання Інтернет-Сервісів в освітньому процесі» : Доповідь. Чернівці, 2022.
29. Олена Шеремет, Тетяна Кирик. Веб-застосунок для створення і перевірки домашніх завдань у цифровізації освітнього процесу. Інформаційні технології в професійній діяльності : матеріали XVIII Всеукраїнської науково-практичної конференції (Рівне, 10 листопада 2025 року). Рівне : РВВ РДГУ. 2025. С. 163-166.
30. Переваги та недоліки кросплатформної та нативної розробки мобільних додатків. *Merehead*. URL: https://merehead.com/ua/blog/cross-platform-native-mobile-development/.
31. Про мікросервісну архітектуру. *FoxmindEd*. URL: https://foxminded.ua/mikroservisna-arkhitektura/.
32. Шеремет О. П. Виклики та рішення при розробці веб-платформи для автоматизації створення та перевірки домашніх завдань. *Розвиток сучасної науки: актуальні питання теорії та практики: матеріали VIII Всеукраїнської студентської наукової конференції*: матеріали конф., м. Запоріжжя, 20 черв. 2025 р. Вінниця, 2025. С. 383–385.
33. Що таке веб-додаток: чим сайт відрізняється від веб-додатку?. *Outsourcing Team*. URL: https://outsourcing.team/ua/blog/development/shho-take-veb-dodatok-chim-sajt-vidriznyayetsya-vid-veb-dodatku/

ДОДАТКИ
ДОДАТОК А
ER-діаграма бази даних вебзастосунку

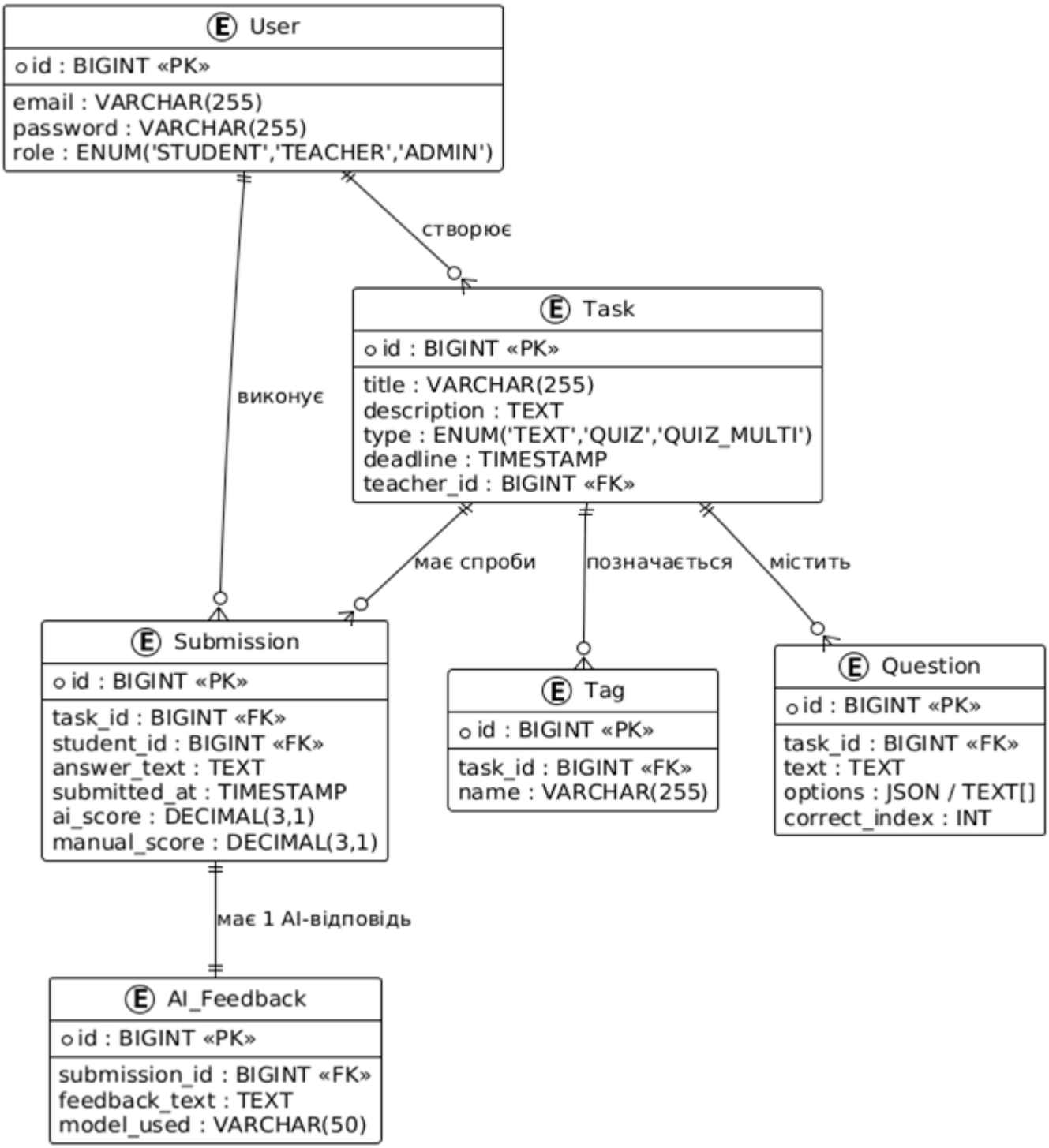

Рисунок А.1. ER-діаграма бази даних вебзастосунку для створення та перевірки домашніх завдань

На рисунку А.1 представлено ER-діаграму (Entity–Relationship Diagram) бази даних вебзастосунку призначеного для створення, виконання та перевірки домашніх завдань у навчальному середовищі.

Діаграма відображає логічну структуру системи, основні сутності, їх атрибути та зв'язки між ними.

1. *Сутність* ***User***

Сутність *User* (користувач) зберігає інформацію про всіх учасників системи – студентів, викладачів та адміністраторів.

Кожен користувач має унікальний ідентифікатор (*id*), електронну пошту (*email*), зашифрований пароль (*password*) та роль (*role*), що визначає рівень його доступу до функцій застосунку.

Роль задається типом *ENUM* і може приймати три значення: *STUDENT*, *TEACHER* або *ADMIN*.

Зв'язки:

- один користувач-«викладач» може створювати кілька завдань (*Task*);
- один користувач-«студент» може виконувати кілька завдань, формуючи записи у таблиці *Submission*.

2. *Сутність* ***Task***

Сутність *Task* (завдання) містить дані про навчальні завдання, створені викладачами.

Кожне завдання має унікальний ідентифікатор (*id*), назву (*title*), опис (*description*), тип (*type*) та кінцевий термін виконання (*deadline*).

Тип завдання може бути текстовим (*TEXT*), тестовим (*QUIZ*) або комбінованим (*QUIZ_MULTI*).

Поле *teacher_id* виступає зовнішнім ключем, що посилається на автора завдання (таблиця *User*).

Зв'язки:

- одне завдання може містити кілька питань (*Question*);
- кожне завдання може мати кілька поданих рішень (*Submission*);
- завдання може позначатися тегами (*Tag*).

3. *Сутність* ***Question***

Сутність *Question* (питання) використовується для завдань тестового типу. Вона зберігає текст питання (*text*), варіанти відповідей (*options*) у форматі JSON або масиву, а також індекс правильної відповіді (*correct_index*).

Поле *task_id* є зовнішнім ключем, який пов'язує питання з відповідним завданням.

Зв'язок:

- один запис *Task* може мати декілька пов'язаних питань у таблиці *Question*.

4. *Сутність* ***Submission***

Сутність *Submission* (відповідь студента) призначена для зберігання результатів виконання завдань студентами.

Вона містить текст або посилання на завантажений файл (*answer_text*), дату подачі (*submitted_at*), а також оцінки – автоматичну від AI (*ai_score*) і ручну від викладача (*manual_score*).

Кожна відповідь прив'язується до конкретного завдання (*task_id*) і студента (*student_id*).

Зв'язки:

- кожна відповідь має один запис у таблиці *AI_Feedback*, що містить текстову оцінку, сформовану штучним інтелектом;
- відповіді студентів пов'язані із сутністю *User* (через *student_id*) і належать певному завданню (*task_id*).

5. *Сутність* ***AI_Feedback***

Сутність *AI_Feedback* (відгук штучного інтелекту) зберігає результати автоматичного аналізу студентських відповідей.

Кожен запис містить текст коментаря (*feedback_text*) і назву моделі AI (*model_used*), яка була використана для перевірки.

Поле *submission_id* виступає зовнішнім ключем, що вказує на конкретну роботу студента (*Submission*).

Зв'язок:

- між сутностями *Submission* та *AI_Feedback* існує зв’язок 1:1, тобто кожна відповідь має лише один AI-відгук.

6. *Сутність* ***Tag***

Сутність *Tag* (мітка) використовується для класифікації завдань за темами або категоріями, що полегшує пошук і фільтрацію.

Вона містить унікальний ідентифікатор (*id*), назву мітки (*name*) та зовнішній ключ (*task_id*), що вказує на відповідне завдання.

Зв’язок:

- один запис *Task* може мати кілька міток, що реалізує зв’язок 1:N між *Task* і *Tag*.