МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

СНИТКО АНАСТАСІЯ РУСЛАНІВНА

Допускається до захисту:
в.о. завідувача кафедри
інформаційних технологій
канд. техн. наук, доцент
_____ О. В. Зелінська
«____» _____ 20__ р.

ВЕБДОДАТОК ДЛЯ ТЕСТУВАННЯ ЗНАНЬ ТА ПРОВЕДЕННЯ ОПИТУВАНЬ

Спеціальність 122 «Комп’ютерні науки»

Кваліфікаційна (бакалаврська) робота

Керівник:
Антонов Ю.С., кан. фіз.-мат. наук, доцент,
доцент кафедри інформаційних
технологій

_____
(Підпис)

Оцінка _____/_____/_____
(бали/за шкалою ЄКТС/за національною шкалою)
Голова ЕК: _____
(Підпис)

Вінниця 2025

АНОТАЦІЯ

Снитко А. Р. Вебдодаток для тестування знань та проведення опитувань. Спеціальність 122 «Комп'ютерні науки». Донецький національний університет імені Василя Стуса, Вінниця, 2025.

Кваліфікаційна робота присвячена актуальній проблемі проєктування та розробки вебсистеми для проведення опитувань і тестування знань. Робота спрямована на створення функціонального, зручного та надійного інструменту оцінювання, який усуває обмеження існуючих платформ, особливо в контексті зростання популярності дистанційного навчання та онлайн-взаємодії у бізнесі.

Робота складається зі вступу, трьох розділів, висновків, списку використаних джерел із 64 найменувань та одного додатку. Робота містить 16 рисунків. Загальний обсяг роботи становить 97 сторінки.

У вступі обґрунтовано актуальність теми дослідження, визначено об'єкт, предмет, мету та завдання роботи, а також описано практичну цінність розробки.

У першому розділі проведено аналіз історії, еволюції та класифікації систем тестування знань і опитувань. Розглянуто існуючі платформи, їхні переваги та недоліки, а також проаналізовано комп'ютерно-математичні моделі, що використовуються в автоматизованих системах контролю знань.

Другий розділ присвячено проєктуванню вебсистеми. Сформульовано детальні функціональні та нефункціональні вимоги до програмного продукту. Розроблено реляційну модель даних у PostgreSQL, що підтримує питання з кількома правильними відповідями та нормалізована для забезпечення цілісності даних. Описано трирівневу архітектуру системи (клієнт – сервер додатків – база даних), де клієнтська частина реалізується як SPA на React, а серверна – на Node.js та Express.

У третьому розділі детально описано процес практичної реалізації розробленого вебдодатку. Реалізовано front-end частину з використанням React, Redux, React Router та Ant Design, та back-end частину на Node.js, Express.js з Drizzle ORM для взаємодії з PostgreSQL. Впроваджено механізми JWT-

автентифікації, логування дій користувачів та базову аналітику. Представлено результати роботи системи та проведено аналіз користувацького досвіду (UX).

У висновках підведено підсумки виконаної роботи, зазначено, що поставлену мету досягнуто. Створений вебдодаток є готовим програмним продуктом. Окреслено наукову новизну, що полягає в інтеграції механізмів логування та аналітики для ітеративного вдосконалення системи, та перспективи подальшого розвитку.

Ключові слова: вебсистема, тестування знань, онлайн-опитування, JavaScript, Node.js, Express, React, PostgreSQL, Drizzle ORM, JWT, REST API, SPA, логування, аналітика.

Snitko A. R. Web application for knowledge testing and conducting surveys. Specialty 122 "Computer Science". Vasyl Stus Donetsk National University, Vinnytsia, 2025.

The qualification thesis is dedicated to the urgent problem of designing and developing a web system for conducting surveys and knowledge testing. The work aims to create a functional, convenient, and reliable assessment tool that eliminates the limitations of existing platforms, especially in the context of the growing popularity of distance learning and online interaction in business.

The work consists of an introduction, three chapters, conclusions, a list of references with 64 titles, and one appendix. The work contains 16 figures. The total volume of the work is 97 pages.

The introduction substantiates the relevance of the research topic, defines the object, subject, purpose, and tasks of the work, and also describes the practical value of the development.

The first chapter analyzes the history, evolution, and classification of knowledge testing and survey systems. Existing platforms, their advantages, and disadvantages are considered, as well as computer-mathematical models used in automated knowledge control systems are analyzed.

The second chapter is devoted to the design of the web system. Detailed functional and non-functional requirements for the software product are formulated. A relational data model in PostgreSQL has been developed, which supports questions with multiple correct answers and is normalized to ensure data integrity. A three-tier system architecture (client – application server – database) is described, where the client-side is implemented as an SPA in React, and the server-side is in Node.js and Express.

The third chapter describes in detail the process of practical implementation of the developed web application. The front-end part was implemented using React, Redux, React Router, and Ant Design, and the back-end part using Node.js, Express.js with Drizzle ORM for interaction with PostgreSQL. JWT authentication mechanisms, user action logging, and basic analytics were implemented. The results of the system's operation are presented and a user experience (UX) analysis is conducted.

In the conclusions, the results of the work are summarized, and it is noted that the set goal has been achieved. The created web application is a ready-made software product. The scientific novelty, which consists in the integration of logging and analytics mechanisms for iterative improvement of the system, and prospects for further development are outlined.

Keywords: web system, knowledge testing, online surveys, JavaScript, Node.js, Express, React, PostgreSQL, Drizzle ORM, JWT, REST API, SPA, logging, analytics.

ЗМІСТ

ВСТУП

У сучасних умовах стрімкої інформатизації освіти та бізнесу вебсистеми для проведення опитувань і тестування знань набули особливої актуальності. Пандемія COVID-19 лише підкреслила цю тенденцію: станом на середину 2020 року 93% домогосподарств із дітьми шкільного віку були залучені до дистанційного навчання, а кількість студентів, що навчалися онлайн, зросла з 2,4 млн у 2019 році до 7,0 млн у 2020 році. Такий різкий перехід до дистанційного навчання і роботи продемонстрував потребу в ефективних інструментах для перевірки знань і збору даних. Онлайн-опитування та електронні тестування сьогодні використовуються всюди – від академічних тестів успішності студентів до опитувань ринку та соціологічних досліджень.

Актуальність теми: Розробка вебсистеми для опитувань і тестування знань є актуальною через зростаючий попит на зручні, масштабовані та надійні інструменти оцінювання. Традиційні паперові методи тестування поступаються місцем цифровим платформам, що забезпечують миттєвий збір і аналіз даних. Однак існуючі рішення мають низку обмежень – від технічних (наприклад, залежність від інтернет-з'єднання) до методичних (наприклад, ризик зниження мотивації без належного зворотного зв'язку). Тому дослідження цієї теми спрямоване на виявлення сильних і слабких сторін сучасних платформ та обґрунтування напрямів їх вдосконалення.

Об'єкт дослідження: Вебсистеми опитувань і тестування знань**,** що забезпечують дистанційне оцінювання результатів навчання та збір анкетних даних.

Предмет дослідження: методи і засоби розробки програмного забезпечення мовою JavaScript**,** моделі та інструменти проєктування, реалізації й аналітики вебплатформи, яка: 1) підтримує питання з кількома правильними відповідями, 2) забезпечує логування дій користувачів, 3) гарантує масштабованість і безпеку даних.

Мета дослідження: Спроєктувати та розробити вебсистему для опитувань і тестування знань, яка усуває обмеження існуючих платформ, підвищує зручність та надійність оцінювання.

Завдання дослідження:

1. Проаналізувати еволюцію систем тестування й опитувань, визначити їх сильні й слабкі сторони.
2. Сформувати вимоги до нової системи (функціональні та нефункціональні).
3. Спроєктувати модель даних у PostgreSQL.
4. Розробити back-end (Node.js + Express + Drizzle ORM) із захистом JWT, логуванням та REST-API. Створити front-end SPA на React з адаптивним інтерфейсом, Ant Design і клієнтським роутингом.
5. Інтегрувати механізми логування й базову аналітику для відстеження взаємодії користувачів та формування звітів.
6. Оцінити результати та визначити напрями подальшого розвитку.

РОЗДІЛ 1
ІСТОРІЯ, ЕВОЛЮЦІЯ ТА АНАЛІЗ СИСТЕМ ТЕСТУВАННЯ ЗНАНЬ І ОПИТУВАНЬ

1.1 Поняття систем тестування знань і опитувань.

Тестування знань – це процес оцінювання рівня засвоєння матеріалу за допомогою спеціально підготовлених завдань [1]. Тести є одним із найбільш широко використовуваних способів перевірки засвоєних знань, адаптованим як для традиційного навчання, так і для дистанційної та змішаної форм навчання. В освітньому процесі тестування дозволяє об’єктивно порівняти рівень знань різних учнів за стандартизованими завданнями.

Опитування (анкетування) – метод збору інформації шляхом постановки запитань певній цільовій групі. Згідно з визначенням, опитування – це метод збирання соціологічної інформації про досліджуваний об’єкт під час безпосереднього (усного) або опосередкованого (письмового) спілкування дослідника з респондентом [2]. На відміну від тесту знань, опитування звичайно не передбачає однозначно “правильних” відповідей – його метою може бути виявлення думок, відгуків, рівня задоволеності чи самооцінки знань.

Науковці також розмежовують ці поняття. Зокрема, тестування знань розглядається як стандартизований засіб об’єктивної перевірки рівня засвоєння матеріалу, тоді як опитування (анкета) призначене для збирання суб’єктивної інформації від респондентів» [3].

Сучасні **системи тестування знань** і опитувань – це програмні або апаратні комплекси, що автоматизують процес контролю знань або збору даних від респондентів. Вони дозволяють створювати запитання різних форматів, проводити тестування чи анкетування, а також фіксувати й обробляти отримані результати. Хоча багато **інтегрованих навчальних платформ (Learning Management Systems, LMS)**, таких як Moodle, включають модулі для тестування, їх основне призначення — організація навчального процесу. Тому, поряд з LMS, активно використовуються й **спеціалізовані онлайн-сервіси**

(наприклад, SurveyMonkey, Kahoot!), які часто пропонують більш глибокий функціонал саме для створення тестів та проведення опитувань.

Методи тестування можна умовно поділити на офлайн (традиційні) та онлайн (комп'ютерні). Офлайн-тестування передбачає проведення перевірки знань без використання інтернет-технологій – наприклад, письмові контрольні роботи на папері, усні опитування або екзамени в аудиторії за присутності екзаменатора. Онлайн-тестування здійснюється за допомогою комп'ютерів, смартфонів чи інших гаджетів, підключених до мережі. Застосування онлайн-технологій у тестуванні відкриває суттєві переваги. Перш за все, це стосується просторово-часової гнучкості: учасники можуть проходити оцінювання дистанційно, у зручний для них час в межах встановленого терміну, на відміну від традиційних іспитів, що вимагають фізичної присутності у визначеному місці та часі. Другою важливою перевагою є автоматизація процесу перевірки електронних тестів, що дозволяє отримувати результати практично миттєво після завершення, забезпечуючи оперативний зворотний зв'язок. Це контрастує з традиційними письмовими роботами, ручне оцінювання яких може тривати значний час, відтерміновуючи надання результатів.

Водночас, онлайн-тестування висуває нові вимоги. Питання автентифікації та контролю є критичним: без належного нагляду важко гарантувати, що відповіді надає саме той студент, який зареєстрований. Також у дистанційному форматі складніше проконтролювати, щоб студент не користувався підручниками або пошуком в інтернеті під час виконання завдань. Для вирішення цих проблем розробляються технології онлайн-прокторингу (відеоспостереження за учасниками, блокування переключення вікон браузера тощо). В аудиторному (офлайн) тестуванні питання доброчесності легше контролювати завдяки безпосередньому нагляду викладача та перевірці документів.

Існують також **гібридні підходи** до тестування, що поєднують офлайн- і онлайн-методи. Такий підхід інтегрує живе спілкування в класі з автоматизованим збором та обробкою результатів. Учитель одразу бачить

статистику відповідей і може проаналізувати, які питання викликали труднощі, хоча учні при цьому не потребують дорогих пристроїв.

Класифікація тестових питань. Тестові завдання прийнято поділяти на дві основні категорії за формою відповіді: відкриті та закриті типи питань. Залежно від способу надання відповіді, тестові завдання поділяються на дві основні категорії. Завдання відкритого типу вимагають від респондента самостійного конструювання відповіді, що може включати заповнення пропущених елементів (слів, чисел, символів) або надання розгорнутого текстового пояснення, як у випадку есе. Альтернативно, завдання закритого типу пропонують учаснику набір попередньо сформульованих варіантів, з яких необхідно обрати один чи декілька правильних. Ця категорія охоплює різні формати, зокрема завдання з єдиним або множинним вибором, завдання на встановлення відповідностей між елементами двох множин, завдання на визначення правильної послідовності та завдання типу "істинно/хибно". Правильне формулювання тестових запитань вимагає чіткості та однозначності, незалежно від їх типу. В педагогічній практиці рекомендується уникати надто складних конструкцій, двозначностей і підказок у формулюваннях питань і варіантів відповідей.

Окрім форми відповіді, тестові питання класифікують і за складністю. Традиційно виокремлюють три рівні складності завдань: легкий, середній та складний. Наприклад, за методикою Ебеля експерти спершу розподіляють завдання по групах складності, а потім визначають, який відсоток респондентів з мінімально необхідним рівнем знань міг би правильно відповісти на завдання кожного рівня. Таким чином можна збалансувати тест: включити достатню кількість простих питань для перевірки базових знань, а також складніших – для диференціації рівня підготовки. Складність тестового питання тісно пов’язана з обсягом знань і умінь, потрібних для правильної відповіді, тому при конструюванні тесту важливо враховувати таксономію освітніх цілей (наприклад, рівні когнітивних навичок за Блумом).

Також тестові завдання розрізняють за призначенням тестування. Залежно від мети оцінювання, виділяють кілька типів тестів. Навчальні тести (тренувальні, формувальні) призначені для поточного контролю засвоєння матеріалу – вони проводяться під час вивчення теми і допомагають студенту та викладачу виявити прогалини у знаннях та закріпити матеріал. Контрольні тести (підсумкові) проводяться для оцінювання знань після завершення вивчення розділу чи курсу; до них належать модульні контрольні роботи, екзаменаційні тести тощо. Окремо можна виділити тести діагностичні (вхідні), які використовуються для перевірки початкового рівня знань (наприклад, при вступі або розподілі по групах), а також тести для самоконтролю, що дають змогу студентам самостійно перевірити свої знання без оцінки викладача. Правильне визначення цілей тестування допомагає сформувати адекватний набір питань потрібного рівня складності та тематики.

1.2 Історія та еволюція систем тестування знань і опитувань

Витоки стандартизованого тестування: ідея стандартизованого тестування знань виникла понад століття тому. На початку XX століття у педагогічній психології розроблялися перші типові тести успішності: під керівництвом Е. Торндайка у 1908 році було створено перший стандартизований тест з арифметики (М. Стоун), а в 1915 році Роберт Єркс запровадив серію тестів для оцінювання інтелекту, що використовували нові системи підрахунку результатів [4]. Ці ранні паперові тести заклали основу для об’єктивного контролю знань, хоча проводились вручну. Упродовж середини XX століття стандартизоване тестування набуло масового характеру – прикладом є тести досягнень у школах та психометричні тести (наприклад, SAT чи тести IQ). Проте всі ці методики до 1960-х років покладалися на паперові анкети та значні трудовитрати на перевірку.

Поява комп’ютеризованого контролю знань: розвиток обчислювальної техніки у другій половині XX століття дав потужний імпульс для автоматизації тестування. Одним із перших прикладів застосування комп’ютера в навчанні стала система PLATO (Programmed Logic for Automatic Teaching Operations),

створена в 1960 році в Університеті Іллінойсу [5]. Вона стала першою універсальною системою комп'ютерного навчання, що успішно використовувалася для навчання і фактично продемонструвала можливості інтерактивного викладання з автоматизованим зворотним зв'язком . Незважаючи на обмеження великих мейнфреймів, PLATO довела здійсненність комп'ютерного тестування. До 1970-х років у різних країнах почали з'являтися експериментальні системи автоматизованого контролю знань. У СРСР та Україні піонером у цій галузі вважається В.П. Беспалько – його дослідження з педагогічного проектування навчання за участю комп'ютерів сприяли появі численних проєктів [6]. Наприклад, у 1980-х роках було створено систему комп'ютерного тестування успішності навчання для вищих медичних закладів, а також комбіновані системи діагностування знань, що поєднували традиційні бланкові тести з електронною обробкою результатів. У світовій практиці відбувався перехід до електронних тестів: у середині 1980-х років було впроваджено computer-based testing (CBT) – комп'ютерні іспити як електронний аналог паперових тестів.

Перехід до вебтестування: з розвитком мережевих технологій і Інтернету в 1990-х роках системи тестування еволюціонували у напрямі веборієнтованих платформ. Почали з'являтися системи дистанційного навчання (LMS – Learning Management Systems), які інтегрували модулі тестування. Перші LMS, такі як WebCT (1996 р.) і Blackboard (1997 р.), дозволяли проходити онлайн-квізи [7, 8]. У 2002 році з'явилася відкрита LMS Moodle, що також містила інструменти для створення тестових завдань і автоматичного оцінювання [9]. Цей етап ознаменував перенесення контролю знань у вебсередовище на постійній основі, що дозволило проводити стандартизовані тестування та сертифікаційні іспити у форматі CBT, наприклад, GRE та GMAT у кінці 1990-х років. В Україні ця тенденція отримала додаткове підтвердження у 1995 році, коли І.Є. Булах опублікував монографію, присвячену теорії і методиці комп'ютерного тестування успішності навчання, що заклало наукове підґрунтя для майбутніх розробок [10].

Розвиток онлайн-опитувань: системи опитувань мають власну траєкторію розвитку, пов’язану з появою Інтернету. Перші соціологічні опитування проводились ще “на папері” і телефоном, але наприкінці 20-го століття почали використовувати електронні засоби. Наприклад, перепис населення США 1890 року проводився за допомогою табулятора, що було одним із перших випадків автоматизації аналізу анкетних даних. З появою персональних комп’ютерів у 1980-х роках виникло поняття CAWI (Computer-Assisted Web Interviewing). Бурхливе зростання мережі у другій половині 1990-х створило сприятливі умови для динамічного розвитку онлайн-опитувань. На цій хвилі у 1999 році в США було засновано сервіс SurveyMonkey, а у 2002 році – платформу Qualtrics, що дозволило швидко створювати вебопитувальники та автоматизувати збір даних [8, 9]. У 2006 році з’явилася функція Google Forms як частина Google Sheets, яка стала безкоштовним інструментом для масових опитувань [11].

Сьогодні вебплатформи для опитувань і тестування знань є невід’ємною частиною освітнього процесу, HR-менеджменту, маркетингових досліджень. Вони постійно вдосконалюються, пропонуючи адаптивні тести, миттєвий аналіз даних, інтеграцію з іншими сервісами. Разом із розвитком технологій виникають і проблеми – від технічних (кібербезпека, масштабованість) до педагогічних (надійність оцінювання, мотивація користувачів). Це свідчить про необхідність розробки більш універсальних і функціонально багатогранних систем, що поєднають сильні сторони різних підходів і дозволять вирішувати типові проблеми за допомогою систем логування та аналітики.

1.3 Класифікація комп’ютерних систем тестування

В науковій літературі запропоновано різні підходи до класифікації систем комп’ютерного тестування знань. В науковій літературі, присвяченій комп'ютерним системам тестування (КСТ), запропоновано підходи до їх класифікації залежно від архітектури бази даних, що використовується [12]. Відповідно до цієї класифікації розрізняють три основні категорії КСТ:

1. **Автономні (локальні) системи.** Клієнтські тестові програми встановлюються безпосередньо на кожен комп’ютер користувача, а всі

файли тестових завдань і результати зберігаються локально на цих машинах (використовується централізована архітектура або СУБД взагалі відсутня) journal.iitta.gov.ua. Подібні системи доцільно застосовувати переважно в локальному середовищі без мережі (наприклад, у комп'ютерних класах під керуванням MS-DOS або за відсутності локальної мережі). Прикладом таких програм є MyTest та MultyTest– прості тестові оболонки, що працюють автономно на кожному робочому місці.

2. **Дворівневі клієнт-серверні системи.** Програма-клієнт також встановлюється на кожному комп'ютері користувача, але тестові завдання і результати централізовано зберігаються на окремому сервері (реалізовано технологію клієнт–сервер) . Порівняно з першою категорією, такі системи забезпечують централізоване зберігання і обробку даних на сервері, що спрощує резервне копіювання та оновлення інформації, а також уможливлює обмеження доступу для підвищення безпеки . Прикладом системи цієї категорії є програма Expert – вона централізує всі дані на одному сервері, а користувачі працюють через встановлені на їх ПК клієнтські додатки.
3. **Трирівневі веборієнтовані системи.** В ролі клієнтського додатка виступає звичайний веббраузер, сервером застосунків є вебсервер, а всі дані зберігаються у централізованій базі даних, керованій СУБД (реалізовано технологію трирівневої архітектури БД) . Системи цієї категорії побудовані за класичною схемою «клієнт – сервер додатків – база даних» і для доступу до них потрібне мережеве з'єднання. Прикладами таких КСТ є популярні платформи дистанційного навчання Moodle та OpenTest, які використовують ве-інтерфейс для проведення тестування.

Найперспективнішими для сучасної освіти є саме системи третьої категорії, створені на основі технології трирівневих баз даних. Вони мають низку важливих переваг порівняно з першою та другою категоріями, а саме:

1. **Зручність супроводження та оновлення ПЗ.** У разі зміни або модернізації програмного забезпечення оновлення достатньо виконати лише на сервері, що значно скорочує час і зусилля на підтримку системи для адміністраторів . Користувачі при цьому автоматично працюють з актуальною версією через браузер, без необхідності перевстановлення програм на кожному робочому місці.
2. **Легка масштабованість.** Для додавання нових користувачів або цілих груп (класів) не потрібно встановлювати жодного додаткового програмного забезпечення на їхні комп'ютери – достатньо наявності будь-якого веббраузера та підключення до мережі (локальної або Інтернет) . Це дозволяє без проблем розгортати систему тестування на великий кількості машин і географічно віддалених користувачів.
3. **Кросплатформеність.** Веборієнтована система не залежить від операційної системи на боці клієнта, тому для доступу підходять різні пристрої та ОС – від застарілих версій Windows до Linux і macOS . Фактично будь-який комп'ютер або планшет з браузером може бути використаний для проходження тестування, що розширює аудиторію та дозволяє заощадити кошти на уніфікації робочих місць.
4. **Придатність до дистанційного навчання.** Системи з трирівневою архітектурою від самого початку орієнтовані на мережеву взаємодію, тому їх відносно легко адаптувати для проведення дистанційного тестування студентів або онлайн-іспитів . Центральний сервер, підключений до Інтернету, забезпечує доступ до тестів у будь-який час і з будь-якого місця, що є важливою умовою для сучасних технологій електронного навчання.

Варто зазначити, що переваги трирівневої архітектури особливо проявляються у середовищі з багатьма користувачами або навчальними класами; натомість для одиничного комп'ютерного класу ефект може бути менш відчутним. Основним недоліком веборієнтованих систем є потреба у надійному мережевому з'єднанні та достатньо продуктивному сервері для обслуговування всіх клієнтів . Попри це, саме трирівневі КСТ наразі відповідають сучасним

вимогам і забезпечують максимальну гнучкість та ефективність у процесі комп’ютерного тестування знань.

1.4 Аналіз існуючих платформ для опитувань і тестування знань

У цьому розділі виконується порівняльний аналіз популярних вебплатформ, що дозволяють створювати та проводити опитування і онлайн-тести. Будуть розглянуті Google Forms, SurveyMonkey та Kahoot! як представники різних підходів: універсальні онлайн-форми, спеціалізований сервіс анкетування та гейміфікована система вікторин [13, 14]. Аналіз ґрунтується на інформації з офіційних джерел і наукових досліджень, з посиланнями на джерела.

Варто зазначити, що подібні системи привертають увагу дослідників. Так, у роботі Чернящук та ін. проведено детальний аналіз популярних платформ для тестування знань і опитувань, окреслено їхні переваги й недоліки, зокрема акцентовано на важливості мотивації студентів та надійності доступу до сервісу* [14]. В іншому дослідженні підкреслено, що головною метою впровадження систем онлайн-тестування є саме спрощення роботи викладача та надання студентам можливості самостійно оцінювати власні знання [15].

Google Forms.

Google Forms – безкоштовний онлайн-інструмент для створення опитувальників і тестів, що входить до складу хмарного пакету Google Drive. Спочатку сервіс з’явився у 2008 році як функція Google Sheets, а з 2016 року був виділений в окремий додаток. Форма дозволяє створювати різноманітні типи питань (текст, варіанти відповідей, шкали) та налаштовувати логіку переходів, а відповіді автоматично зберігаються в Google Sheets.

Переваги:

- Простота використання, інтуїтивно зрозумілий інтерфейс.
- Цілодобова доступність і автоматичне збереження даних.
- Інтеграція з іншими сервісами Google, що дозволяє легко отримувати статистику та аналізувати дані.

Недоліки:

- Обмеження базового функціоналу: відсутність таймера для обмеження часу тестування, недостатня гнучкість для створення складних тестових сценаріїв.
- Невисока кастомізація інтерфейсу, що може бути недоліком для спеціалізованих завдань.

Логування та аналіз в Google Forms: для перевірки ефективності Google Forms можна впровадити логування: збір даних про час заповнення форми, аналіз відсотка респондентів, що припиняють заповнення, дозволить визначити, чи потрібен таймер або додаткові інструкції.

SurveyMonkey

SurveyMonkey – один із найстаріших і найвідоміших комерційних сервісів для створення онлайн-опитувань, заснований у 1999 році. Платформа дозволяє створювати анкети з різними типами питань, використовувати логіку розгалуження, розсилати опитування і автоматично аналізувати отримані дані.

Переваги:

- Широке охоплення аудиторії і швидкість збору даних.
- Автоматизована аналітика, що дозволяє в режимі реального часу переглядати графіки та статистичні дані.
- Різноманітність типів питань і можливість налаштування логіки анкети [9].

Недоліки:

- Низький відсоток завершених опитувань через перевантаженість респондентів та занадто довгі анкети.
- Обмеження безкоштовної версії: мінімальна кількість запитань та відповідей.
- Питання конфіденційності даних, якщо налаштування анонімності не виконані належним чином.

Логування та аналіз у SurveyMonkey: може включати відстеження прогресу респондентів (на якому етапі анкета покидається), аналіз часу відповіді

на кожне питання, що дозволить визначити найбільш проблемні ділянки анкети і, відповідно, вдосконалити її структуру.

Kahoot!

Kahoot! – онлайн-платформа, що використовує гейміфікацію для тестування знань. Запущена у 2013 році норвезькими розробниками, платформа демонструє питання на спільному екрані, а учні відповідають за допомогою смартфонів чи комп'ютерів, змагаючись за бали у режимі реального часу.

Переваги:

- Висока залученість і мотивація учасників завдяки гейміфікації, рейтинговій системі та змаганням.
- Миттєвий зворотний зв'язок та оновлення рейтингів після кожного питання.
- Універсальність: платформа використовується як у школах, так і в корпоративному навчанні.

Недоліки:

- Формат орієнтований на швидкість, що може заважати глибокому опрацюванню матеріалу.
- Залежність від стабільного інтернет-з'єднання та високих технічних вимог до пристроїв учасників.
- Обмеження у форматах питань – підтримка переважно тестових завдань із закритими відповідями.

Логування та аналіз у Kahoot!: для Kahoot! важливим є логування таких параметрів, як час відповіді на кожне питання, кількість технічних збоїв, частота збоїв при великій кількості учасників. Це дозволить визначити, чи не впливає акцент на швидкості на якість засвоєння матеріалу, а також оптимізувати технічну інфраструктуру.

Інші рішення (Qualtrics, Moodle тощо).

Окрім описаних платформ, існують інші системи, що займають окремі ніші. Такі як: Mentimeter, Typeform, Microsoft Forms, Poll Everywhere, Slido [16-20]. Наприклад, Qualtrics – потужна платформа для професійних опитувань і

маркетингових досліджень, що використовується бізнес-користувачами і науковцями завдяки широким можливостям аналізу даних [21]. Moodle – відкрита система управління навчанням (LMS), яка інтегрує модулі тестування для навчальних закладів.

Переваги:

- Qualtrics забезпечує високий рівень аналізу даних і захисту інформації.
- Moodle пропонує гнучкі налаштування тестів, підтримує імпорт тестових завдань з різних форматів, зокрема XML.

Недоліки:

- Деякі з цих систем є складними у налаштуванні для звичайного користувача і вимагають додаткових знань для розгортання. Логування та аналіз: у системах LMS логування допомагає відслідковувати успішність тестування: зокрема, аналіз відгуків, статистика часу відповіді, кількість повторних спроб. Це дозволяє викладачам і розробникам виявити слабкі місця в тестах і оптимізувати їх.

1.5 Комп’ютерно математичні моделі, що використовуються у автоматизованих системах контролю знань

Автоматизовані системи контролю знань (АСКЗ) пройшли значний шлях еволюції, перетворившись із простих інструментів для перевірки відповідей на складні програмні комплекси. Ключову роль у цьому розвитку відіграють комп'ютерно-математичні моделі, які дозволяють не лише фіксувати факт правильної чи неправильної відповіді, але й здійснювати глибокий аналіз навчальних досягнень, підвищувати об'єктивність оцінювання та адаптувати процес контролю до індивідуальних особливостей учнів. Застосування таких моделей спрямоване на подолання обмежень традиційних АСКЗ, наближаючи їх до ефективності та гнучкості взаємодії "викладач-студент". Ці моделі забезпечують кількісну основу для оцінки якості тестових завдань, рівня знань

тестованих, а також для аналізу повноти та характеру відповідей, особливо у випадках, що виходять за рамки простого вибору з запропонованих варіантів.

Одним із перспективних напрямків удосконалення АСКЗ є інтеграція експертних систем (ЕС) для аналізу відповідей. Такий підхід детально розроблено у роботі , де запропоновано модель ЕС, що не передбачає прямого діалогу з користувачем, а функціонує як інтелектуальний модуль в межах АСКЗ, аналізуючи відповіді, генеруючи додаткові питання та зберігаючи проміжний стан висновку [22].

Інший важливий напрямок використання математичного апарату в АСКЗ пов'язаний зі статистичними моделями, що лежать в основі сучасної теорії тестування (Item Response Theory, IRT). Ці моделі дозволяють оцінювати не лише знання тестованих, але й характеристики самих тестових завдань. У роботі розглядається застосування таких моделей для обробки результатів тестового контролю знань [23].

Подальший розвиток математичних підходів до аналізу результатів тестування включає методики оцінки повноти відповідей. Наприклад, у роботі “Оцінка повноти відповідей в автоматизованих системах контролю знань” запропоновано використовувати метрики відстаней, такі як відстань Левенштейна для аналізу відповідей на відкриті запитання та Декартову відстань для запитань закритого типу, на встановлення послідовності чи відповідності. Такий підхід дозволяє кількісно визначити, наскільки надана відповідь близька до еталонної, та обчислювати відносну кількість помилок, що дає більш детальне уявлення про рівень знань, ніж просто фіксація факту правильної чи неправильної відповіді [24].

Окрім оцінки індивідуальних відповідей, математичні моделі застосовуються для аналізу якості самих тестових завдань. Так, у роботі “Методика аналізу тестових завдань на основі отриманих результатів тестування” запропоновано методику, де правильність відповіді оцінюється через визначення кількості помилок, а складність завдання – через середню кількість помилок, допущених усіма тестованими. У їхньому підході шаблон

правильної відповіді та відповідь студента представляються як точки у багатовимірному просторі, а квадрат відстані між ними інтерпретується як кількість помилок. Це дозволяє не тільки диференціювати завдання за складністю, але й виявляти некоректно сформульовані або неоднозначні питання на основі статистичного аналізу відповідей [25].

Використання цих статистичних моделей є основою для розробки адаптивного тестування, де кожне наступне завдання підбирається індивідуально для тестованого на основі його відповідей на попередні завдання. Це дозволяє більш ефективно та точно оцінити рівень знань за меншу кількість завдань.

Окремо варто згадати техніку представлення даних, таку як бітові маски, яка хоч і не є математичною моделлю оцінювання сама по собі, але є ефективним інженерним рішенням для зберігання та обробки відповідей на питання з множинним вибором у базах даних АСКЗ , що буде розглянуто детальніше в контексті форматів збереження тестових питань [26].

Застосування комп'ютерно-математичних моделей в автоматизованих системах контролю знань є ключовим фактором їхнього розвитку та вдосконалення. Розглянуті підходи, що включають експертні системи для детального аналізу відповідей та статистичні моделі для оцінки якості тестів і знань студентів , демонструють прагнення до підвищення об'єктивності, надійності та інформативності процесу оцінювання.

Висновок до першого розділу

Проведений аналіз показав, що сучасні вебсистеми для опитувань і тестування знань пройшли довгий шлях розвитку – від перших стандартизованих паперових тестів до інтерактивних онлайн-платформ із мільярдами користувачів. Історичний розвиток свідчить про поступову інтеграцію комп'ютерних технологій у процес оцінювання знань, що дозволило автоматизувати підрахунок результатів і забезпечити більш оперативний зворотний зв'язок. Порівняння популярних сервісів виявило, що кожна платформа має свої сильні та слабкі сторони: Google Forms вирізняється простотою, SurveyMonkey – розширеним функціоналом, а Kahoot! – високою мотивацією завдяки гейміфікації. Проте жодне рішення не є універсальним – існують спільні проблемні точки, такі як обмеженість функціоналу для комплексного тестування, технічні ризики та проблеми мотивації респондентів.

Додатково, аналіз охопив комп'ютерно-математичні моделі, що є ключовими для розвитку автоматизованих систем контролю знань (АСКЗ), як детально описано у розділі 1.5. Було розглянуто застосування експертних систем для глибокого аналізу відповідей , та статистичних моделей сучасної теорії тестування (IRT), що дозволяють об'єктивно оцінювати характеристики тестових завдань та рівень знань тестованих, а також є основою для адаптивного тестування. Також було звернуто увагу на інженерні рішення для представлення даних, такі як бітові маски. Ці підходи спрямовані на підвищення об'єктивності, надійності та інформативності процесу оцінювання, наближаючи автоматизовані системи до гнучкості взаємодії «викладач-студент».

На основі проведеного аналізу можна сформулювати вимоги до розробки власної вебсистеми для опитувань і тестування знань: вона має бути функціонально багатогранною, поєднувати переваги сучасних платформ та усувати їхні недоліки. Особливу увагу слід приділити реалізації систем логування та аналітики, що дозволить перевіряти гіпотези щодо причин можливих проблем та на основі реальних даних вдосконалювати продукт. В наступних розділах буде детально розглянуто проектування архітектури

системи, розробку front-end та back-end частин, а також впровадження запропонованих підходів для досягнення високої ефективності вебсистеми нового покоління.

РОЗДІЛ 2
ВИЗНАЧЕННЯ ВИМОГ ДО ПРОГРАМИ ТА ПОБУДОВА МОДЕЛІ ПРОГРАМНОГО ПРОДУКТУ

2.1 Визначення ключового функціоналу додатку

У цьому підрозділі визначено, які можливості мусить забезпечувати вебдодаток для опитувань та тестування знань. Перелік функціональних вимог сформовано з огляду на потреби кінцевих користувачів (респондентів і адміністраторів) та технічні обмеження платформи.

- **Реєстрація та авторизація користувачів.** Система забезпечує створення облікових записів нових користувачів та вхід зареєстрованих користувачів. Реєстрація передбачає введення унікального логіна (наприклад, email) та паролю, авторизація – перевірку цих даних і надання доступу до захищених розділів сайту. Це необхідно для ідентифікації кожного учасника тестування або опитування та персоналізації його результатів. Крім базової аутентифікації, важливо враховувати безпекові аспекти зберігання облікових даних. У більшості сучасних систем, включаючи цю вебплатформу, паролі користувачів зберігаються у вигляді хешів, що гарантує високий рівень захисту. Застосування бібліотек для хешування (наприклад, bcrypt) дозволяє забезпечити криптографічну стійкість навіть у випадку витоку даних. Такий підхід забезпечує дотримання вимог конфіденційності відповідно до загальноприйнятих стандартів у галузі веббезпеки.
- **Ролі користувачів і розмежування прав.** Вебдодаток підтримує гнучку систему ролей. Як мінімум, передбачено дві основні ролі: звичайний користувач, викладач та адміністратор, проте архітектура дозволяє легко додавати нові ролі за потреби. Користувач може мати одну або декілька ролей одночасно. Звичайні користувачі (з відповідною роллю) можуть проходити тестування чи опитування і переглядати тільки власні результати. Викладач – роль користувача,

яка має доступ до результатів виконання тестів студентами. Викладач може переглядати результати студентів, аналізувати статистику успішності та створювати власні тести (екзамени). Ця роль дозволяє викладачу контролювати процес навчання на основі отриманих даних. Адміністратори (користувачі з роллю "адміністратор") мають розширені можливості – зокрема, створювати та редагувати зміст тестів, керувати питаннями, категоріями тестів, переглядати результати всіх користувачів та потенційно керувати іншими користувачами та їх ролями. Таким чином, у системі реалізовано механізм контролю доступу на основі ролей (RBAC), який визначає, який інтерфейс і дії доступні користувачеві залежно від сукупності його ролей. **Створення та управління тестами (опитуваннями).** Адміністратор повинен мати змогу створювати нові тести або опитувальні анкети, змінювати їхні параметри та вилучати ті, які більше не актуальні. Функціонал управління тестами включає операції додавання тесту, редагування його назви чи опису та видалення тесту. На практиці це дозволяє, наприклад, створити новий набір питань для контролю знань з певної теми або сконструювати опитування для збору відгуків. Обидві ролі можуть виконувати авторизацію, після чого їх можливості відрізняються згідно з призначеними правами.

- **Управління питаннями тесту.** Окрім самого тесту, адміністратор керує переліком запитань, що входять до нього. Функціонал управління питаннями передбачає можливість додати нове запитання до вибраного тесту, відредагувати формулювання існуючого запитання (або варіанти відповіді до нього), а також видалити запитання при потребі. Кожне запитання може містити кілька варіантів відповіді, серед яких для тесту знань зазначається правильний варіант. Таким чином, даний підрозділ функціоналу забезпечує наповнення тестів змістом.

- **Проходження тесту або опитування.** Зареєстрований користувач отримує можливість обрати один з доступних тестів (опитувань) зі списку та пройти його, надавши відповіді на всі запитання. Система повинна відображати користувачеві послідовно всі запитання вибраного тесту разом із переліченням можливих варіантів відповіді до кожного. Після того, як на всі питання надано відповіді, користувач надсилає їх для перевірки. Важливо, щоб інтерфейс був інтуїтивно зрозумілим: наприклад, варіанти відповідей подано у вигляді радіокнопок чи чекбоксів, а після заповнення анкети або тесту передбачена кнопка завершення для відправки результатів.
- **Обробка та збереження результатів.** Після отримання відповідей користувача система автоматично обробляє результати. Для тестування знань це означає співставлення відповідей із правильними варіантами і підрахунок підсумкового бала (кількості правильних відповідей або відсотка успішності). Для опитування автоматична перевірка не потрібна – достатньо зафіксувати вибори користувача. У будь-якому разі формується звіт про проходження: він включає дані про користувача, ідентифікатор тесту, час проходження та отриманий результат (бал або відповіді). Цей звіт зберігається у базі даних для подальшого аналізу. Користувачеві після завершення тесту відображається підсумкова інформація – наприклад, набраний бал та можлива оцінка, або підтвердження про успішне надання відповідей (для опитування).
- **Перегляд результатів (звітів).** Вебдодаток надає засоби для перегляду зібраних результатів тестування. Звичайний користувач має мати доступ до власних результатів – тобто, бачити перелік пройдених ним тестів та деталі (дата, оцінка тощо). Адміністратор, зі свого боку, отримує можливість переглядати результати всіх користувачів по всіх тестах. Це реалізується у вигляді окремого розділу «Звіти» в адміністративній частині сайту, де відображено

таблицю або список спроб тестування з можливістю фільтрації за користувачем чи тестом. Такий функціонал дозволяє адміністратору аналізувати успішність групи респондентів або збирати статистичні дані опитування. У розширених версіях систем подібного типу також можуть бути реалізовані функції статистичного аналізу результатів, наприклад, побудова графіків розподілу балів, середніх оцінок по групах чи тестах, визначення найчастіше допущених помилок. Це може стати корисним для викладачів, HR-аналітиків чи організаторів досліджень, оскільки дає змогу приймати рішення на основі зібраних даних. Хоча в межах цієї роботи реалізовано базовий перегляд звітів, платформа має потенціал для масштабування аналітичного функціоналу.

Перелічені вище пункти охоплюють основний функціонал системи [27]. Для їх реалізації серверна частина додатку організована як набір RESTful API-методів, згрупованих за модулями: User, Exam та Report [28]. Таким чином, архітектура backend через ці API покриває весь визначений ключовий функціонал.

2.2 Вимоги до програмного забезпечення системи

Вимоги до програмного забезпечення вебсистеми для опитувань і тестування знань поділяються на функціональні (що саме повинна робити система) та нефункціональні (які загальні властивості та обмеження повинні бути забезпечені). Нижче сформульовано основні вимоги, враховуючи призначення та актуальний функціонал розробленої системи.

Функціональні вимоги:

- **Реєстрація та автентифікація користувачів.** Система повинна забезпечувати можливість створення нового облікового запису і вхід (логін) для зареєстрованих користувачів. Реєстрація передбачає введення унікальних облікових даних (електронної пошти як логіна) та паролю, з подальшим збереженням цих даних у базі (пароль зберігається лише у вигляді хешу). Авторизація виконується шляхом перевірки введених даних користувача і

надання доступу до функцій системи відповідно до його ролі. Повинна підтримуватися чітка відповідність між обліковим записом та його обліковими даними – кожен користувач має мати один набір валідних логін-пароль, щоб уникнути конфліктів і забезпечити унікальність акаунтів.

- **Розмежування ролей і прав доступу.** У системі передбачено гнучку систему ролей, що дозволяє призначати користувачам одну або декілька ролей (наприклад, «звичайний користувач», «адміністратор», «редактор тестів» тощо). Звичайний користувач має змогу проходити призначені йому тестування або опитування і переглядати лише власні результати. Адміністратор (або користувач з відповідними адміністративними правами, наданими через роль) отримує розширені можливості: створювати й редагувати тести, категорії та запитання, призначати тестування користувачам, переглядати результати всіх респондентів, керувати користувачами та їх ролями. Система повинна відображати різний інтерфейс чи меню залежно від сукупності прав, наданих ролями користувача. Логіка застосунку і перевірки прав доступу також повинна враховувати ролі: запити на адміністративні дії від користувача без належних прав мають відхилятися.
- **Створення і редагування тестів (опитувальників).** Адміністратор повинен мати інтерфейс для створення нового тесту або анкети, налаштування його параметрів (назва, опис, приналежність до категорії, час на проходження, прохідний бал тощо) і можливість видалення або архівування вже непотрібних тестів. Також адміністратор повинен мати можливість керувати категоріями тестів (створювати, редагувати, видаляти). Система має забезпечувати зберігання метаданих тесту і автоматично проставляти мітки часу створення/оновлення.
- **Управління запитаннями.** Система повинна надавати засоби для додавання запитань до тесту та керування ними. Адміністратор має можливість створити нове запитання, вказавши його текст, тип (наприклад, відкрите чи закрите) та варіанти відповідей (якщо потрібні). Для тестових завдань знань особливо важливо мати опцію позначення правильного варіанту **(або варіантів)**

відповіді. Окрім створення, повинна бути можливість редагувати формулювання існуючих запитань і варіанти (наприклад, виправити помилку або уточнити текст) та вилучати зайві запитання з тесту. Під час редагування слід забезпечити збереження цілісності даних – наприклад, неможливість видалити запитання, якщо вже є пов'язані з ним результати (без відповідного попередження та обробки).

- **Проходження тестування/опитування.** Зареєстрований користувач (студент або респондент) повинен мати змогу обрати доступний тест із запропонованого списку і пройти його, відповідаючи на послідовність запитань. Система відображає кожне запитання та варіанти відповідей (для закритих питань – у вигляді перемикачів або прапорців, залежно від того, чи питання з одним правильним варіантом чи з кількома). Користувач надає відповіді і по завершенні відправляє їх на перевірку. Обов'язковою вимогою є зручний інтерфейс проходження тесту: зрозуміле розташування елементів, індикація прогресу (номер питання із загальної кількості) та наявність кнопки для підтвердження завершення. Перед відправкою результатів має контролюватися повнота – всі обов'язкові питання повинні бути відповіді, або користувач має явно підтвердити пропуск.
- **Автоматична перевірка та збереження результатів.** Після подання відповідей система автоматично їх обробляє. Для тестів знань це означає порівняння отриманих відповідей із правильними і підрахунок підсумкового результату (кількості правильних відповідей, відсотка успішності або оцінки за заданою шкалою). Для анкет (опитувань без правильних відповідей) – достатньо просто зберегти відповіді без оцінювання. Результат проходження фіксується у системі у вигляді звіту: зберігаються дані про користувача, ідентифікатор тесту, дата та час проходження, отриманий бал/вердикт, а також, за потреби, перелік відповідей користувача на кожне питання. Звіт має бути занесений до бази даних, щоб забезпечити подальший аналіз і перегляд статистики. Користувачу після завершення тесту надається зворотний зв'язок – наприклад, відображається набраний бал і правильні

відповіді (для навчальних тестів) чи повідомлення про успішну відправку анкети (для опитування).

- **Перегляд результатів.** Система повинна надавати користувачам доступ до інформації про результати тестувань. Звичайний користувач бачить **власні** результати: перелік пройдених ним тестів/опитувань з датами та набраними балами, а також, за потреби, деталізацію (які питання були вирішені правильно, які – ні, з відображенням правильних відповідей для навчання). Адміністратор має інструменти для перегляду результатів **усіх** користувачів – наприклад, окрему панель з таблицею звітів, яку можна фільтрувати за тестом або користувачем. Це потрібно для моніторингу успішності та підготовки підсумкових відомостей. Бажано також забезпечити експорт результатів (наприклад, у форматі CSV або Excel) для зовнішнього аналізу.

Нефункціональні вимоги:

- **Зручність інтерфейсу (юзабіліті).** Інтерфейс вебсистеми має бути інтуїтивно зрозумілим і не вимагати тривалого навчання користувача. Важливо дотримуватися принципів доступності: однорідний дизайн, чіткі написи на кнопках, достатній контраст тексту, адаптивність під різні розміри екранів. Для підвищення залученості бажано використовувати візуальні підказки та зворотний зв’язок (наприклад, підсвічування полів з помилками, повідомлення про успішність дій).

- **Продуктивність і масштабованість.** Система повинна витримувати роботу одночасно значної кількості користувачів без помітного погіршення швидкодії. Час відгуку на основні операції (завантаження списку тестів, отримання питань, відправка відповідей) має бути мінімальним – бажано не більше 2–3 секунд. Для цього потрібно оптимізувати запити до бази даних, використовувати асинхронні операції, а за необхідності – кешування. Масштабованість системи означає, що її можна розгорнути на більш потужній інфраструктурі або додати вузли (наприклад, балансування навантаження на декілька серверів) у разі росту аудиторії користувачів.

- **Безпека даних.** Вебдодаток повинен гарантувати конфіденційність і цілісність інформації. Потрібно реалізувати захист даних користувачів: паролі зберігаються **лише у вигляді хешів** (незворотного шифру), при передачі важливих даних мережою слід використовувати захищений протокол HTTPS. Система повинна перевіряти валідність даних, що вводяться (щоб уникнути SQL-ін'єкцій чи XSS-атак – виконувати валідацію і екранування спеціальних символів на боці сервера). Для захисту від несанкціонованого доступу необхідно передбачити механізм авторизації сесії – використання токенів (наприклад, JWT) або сесійних cookie з належними атрибутами безпеки [29, 30]. Таким чином, після успішного входу кожному клієнту видається унікальний токен, який має надаватися у заголовках при наступних запитах; якщо ж запит надійшов без дійсного токена, сервер відхиляє його (аналогічно фронтенд має не допускати перегляду внутрішніх сторінок без авторизації). Також бажано реалізувати базові засоби протидії спаму та накруткам в опитуваннях (CAPTCHA при реєстрації, обмеження на повторне проходження тесту тощо) [31].
- **Надійність і відмовостійкість.** Програмне забезпечення слід розробити з урахуванням можливих збоїв і помилок користувача. Необхідно забезпечити регулярне резервне копіювання бази даних з результатами тестувань, щоб уникнути втрати важливої інформації. Бекенд-сервіс має коректно обробляти виняткові ситуації – наприклад, недоступність БД або збій при збереженні результату – і повідомляти користувача про проблему без розкриття зайвих технічних подробиць. Інтерфейс, зі свого боку, повинен запобігати типовим помилкам користувачів (наприклад, виводити підтвердження перед видаленням тесту, попереджати при спробі випадково закрити сторінку з незавершеним тестом тощо). Відмовостійкість системи може бути підвищена шляхом розгортання її в кластері або хмарному середовищі, де при відмові одного вузла його функції перебирає інший.
- **Модульність і підтримуваність.** Архітектура додатку повинна бути модульною та багатошаровою, щоб спростити подальшу підтримку і розвиток

[32]. Необхідно виділити окремі компоненти системи за ключовими підсистемами (модуль роботи з користувачами, модуль тестування, модуль звітності тощо) і розділити логіку на шари – маршрутизація (контролери), бізнес-логіка (сервіси) та доступ до даних (рівень роботи з базою). Така багатошарова структура покращує структурованість коду: кожен шар відповідає за свою роль (наприклад, контролер тільки отримує HTTP-запит і формує відповідь, сервіс реалізує прикладну логіку, а репозиторій або ORM-модель безпосередньо взаємодіє з БД). Дотримання цього принципу є нефункціональною вимогою, що підвищує підтримуваність – можна модифікувати або замінювати окремі частини (наприклад, поміняти СУБД або ORM) без значного впливу на інші компоненти. Код програми слід документувати і витримувати в єдиному стилі, що полегшить роботу майбутніх розробників над проєктом. Також важлива портативність – можливість розгорнути систему в різних середовищах (на іншому сервері, у Docker-контейнері, в хмарі) з мінімальними змінами. Це досягається використанням стандартних технологій та чітким документуванням залежностей (версії Node.js, СУБД PostgreSQL, конфігураційні змінні тощо).

- **Масштабованість БД*:*** горизонтальне шард-розбиття, реплікація PostgreSQL
- **Доступність*:*** цільовий SLA ≥ 99.9 %, резервне копіювання кожні 6 год.

Наведімо приклади відповідності наведених вимог специфіці подібних систем. Відомо, що опитувальні платформи (такі як Google Forms, SurveyMonkey) приділяють особливу увагу зручності інтерфейсу та масштабованості, оскільки орієнтовані на масове використання. Зокрема, SurveyMonkey пропонує різноманіття типів запитань і гнучку логіку переходів, що підтверджує високі функціональні вимоги до подібних систем. В навчальних системах тестування наголос робиться на автоматичному оцінюванні та надійності збереження результатів, щоб гарантувати об’єктивність контролю знань. У наукових публікаціях відзначається перспектива використання інтелектуальних методів для підвищення якості тестування – наприклад,

запропоновано застосовувати експертні системи для аналізу відповідей на тестові питання, наближаючи процес автоматизованого тестування до взаємодії викладача зі студентом [22].

У підрозділі 2.4 сформульовано чіткі нефункціональні орієнтири, що доповнюють функціонал із 2.1 та визначають якість майбутньої реалізації. Зокрема, задано метрики продуктивності (≤ 300 мс RTT), доступності (SLA 99,9 %) та безпеки (TLS 1.3, bcrypt 12). Перелік вимог слугуватиме контрольним списком під час розробки й тестування прототипу.

2.3 Модель даних

Щоб забезпечити стабільну роботу системи опитувань і тестування, спроєктовано реляційну схему PostgreSQL, де чітко визначено сутності, їх зв’язки й формат даних, що повертаються клієнтові.

Проєктування бази даних для цієї системи базувалося на принципі "спочатку дані" ("Data-First"). Перш ніж визначати конкретні таблиці та їхню структуру, було проведено ретельний аналіз інформаційних потреб системи, що випливають із ключового функціоналу, описаного в розділі 2.1. Цей аналіз дозволив ідентифікувати основні сутності предметної області (користувачі, тести, питання, відповіді, результати), їхні атрибути та взаємозв'язки. Такий підхід, узгоджений з загальноприйнятими практиками розробки (аналогічно до принципів, що могли б обговорюватися в рамках РКСС – розробки клієнт-серверних систем), спрямований на створення гнучкої та логічно обґрунтованої моделі даних, яка ефективно підтримуватиме всі операції системи. Лише після цього етапу концептуального моделювання було розроблено логічну та фізичну модель даних у PostgreSQL.

База даних системи повинна містити інформацію про користувачів, їхні ролі, тести, питання та відповіді, а також результати проходження тестувань. Користувачі можуть мати роль студента, викладача чи адміністратора.

Основними сутностями є: Користувач, Роль, Тест, Питання та Варіант відповіді, а також Результат тестування.

Сутність Користувач містить дані про кожного користувача системи (ідентифікатор, ім’я, електронну пошту, пароль, роль тощо).

Сутність Роль визначає набір прав доступу (ролі: «Студент», «Викладач», «Адміністратор»). Сутність Тест описує параметри тесту (назва, опис, максимальний час проходження, автор тесту тощо). Кожен тест складається з декількох питань, що зберігаються у сутності Питання.

Сутність Питання містить текст питання та посилання на відповідний тест. Для кожного питання у разі тестів з вибором відповіді передбачена сутність Варіант відповіді, яка містить текст варіанту та позначку того, чи є цей варіант правильною відповіддю.

Сутність Результат тестування фіксує спробу проходження тесту конкретним студентом і містить зв’язок з користувачем-студентом, тестом, набраними балами та датою проходження. Між цими сутностями встановлено такі зв’язки. Кожен тест створюється користувачем з роллю «Викладач» або «Адміністратор» і містить кілька питань. Кожне питання належить певному тесту і має один або кілька варіантів відповіді (для тестів з вибором). Студент може проходити один і той самий тест багато разів; кожна спроба зберігається як окремий запис у таблиці результатів тестування з посиланням на тест і користувача. На рисунку 2.1 зображена структура бази даних.

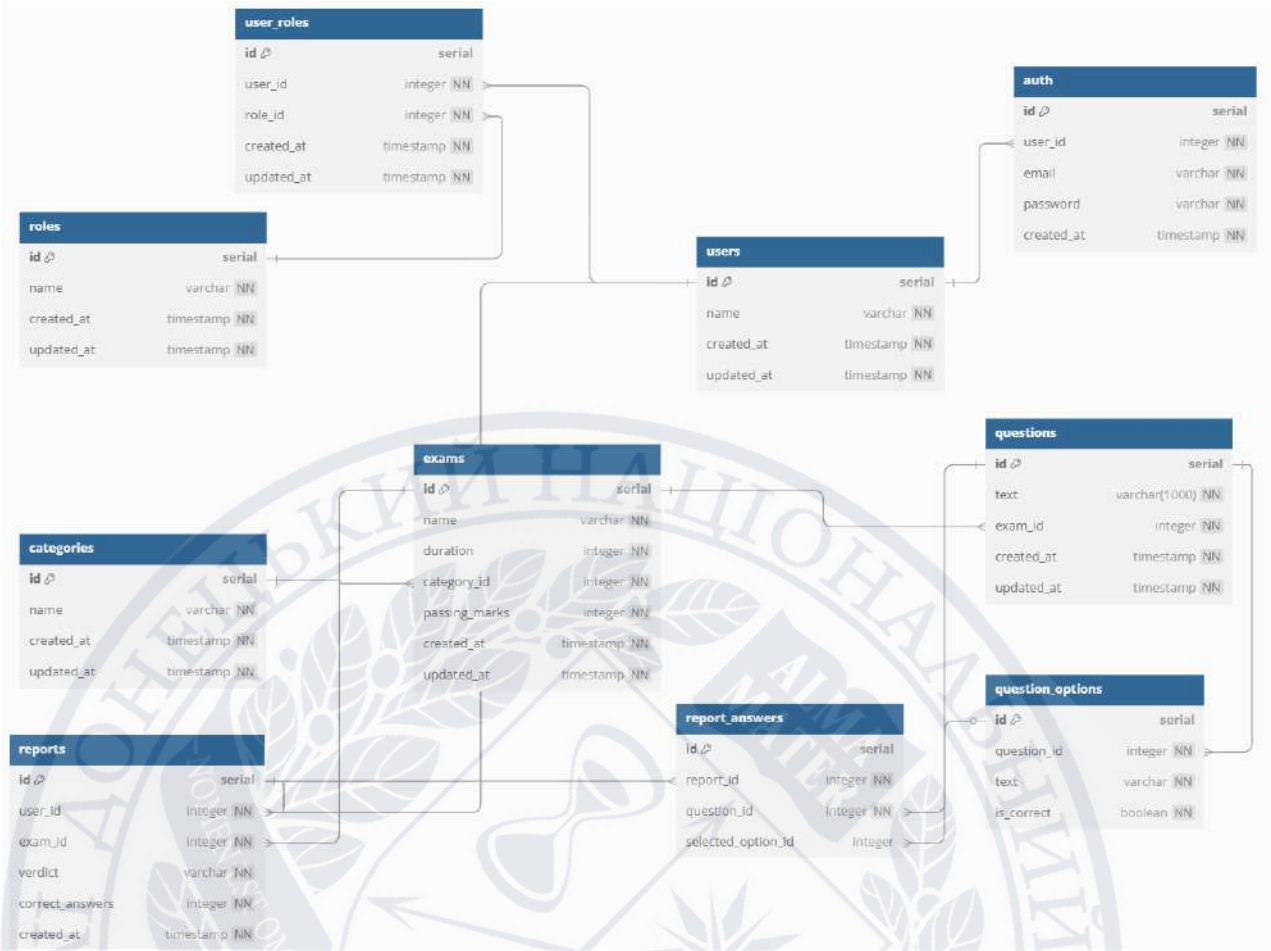


Рисунок 2.1 структура бази даних.

Реалізація цих сутностей у реляційній базі даних відбувається через відповідні таблиці.

Така структура забезпечує гнучкість, нормалізацію даних та підтримку необхідного функціоналу, включаючи можливість призначення користувачам кількох ролей та класифікацію тестів за категоріями. Перевірка на нормалізованість та відсутність аномалій.

Проєктована схема даних відповідає вимогам першої, другої та третьої нормальних форм. Кожна таблиця містить атомарні поля і має первинний ключ; жодна з неключових характеристик не залежить від частини складеного ключа (оскільки складених ключів немає – у кожної таблиці простий PK), і відсутні транзитивні залежності між атрибутами. Це означає, що інформація рознесена по таблицях таким чином, щоб уникнути дублювання і неузгодженостей. Винесення окремих сутностей (ролей, облікових даних, варіантів відповідей, деталей результатів) у свої таблиці було здійснено з метою усунути потенційні аномалії

вставок, оновлень та видалень. Наприклад, жоден користувач не може бути доданий двічі з тим самим email (гарантується унікальністю в auth), при видаленні користувача автоматично видаляються і його облікові дані та звіти (завдяки каскадним зв'язкам), а коригування тексту питання чи варіантів не впливає на інші питання. Таким чином, логічна модель даних є цілісною, узгодженою і позбавленою надлишковості, що забезпечує відсутність аномалій оновлення, видалення чи додавання даних.

Реалізація підтримки запитань із кількома правильними відповідями. Однією з переваг наведеної структури є гнучка підтримка різних типів питань, зокрема таких, що мають декілька правильних варіантів. У старій моделі було передбачено лише одне правильне значення (поле correctAnswer у питанні), тепер же таблиця question_options дозволяє позначити кілька опцій як правильні (is_correct: true). В результаті система може інтерпретувати питання як множинний вибір: на фронтенді для нього відображаються прапорці (checkbox) замість радіокнопок, і користувач може обрати більше ніж одну відповідь. Приклад – питання типу «Виберіть всі правильні твердження…», де кілька відповідей можуть бути правильними одночасно. Під час створення або редагування питання адміністратор має змогу відзначити одну чи кілька опцій як правильні (в інтерфейсі передбачено відповідні прапорці напроти варіантів). Валідація гарантує, що серед варіантів є хоча б один правильний (недопущення ситуації, коли жоден варіант не позначено). При проходженні тесту користувачеві не надсилається інформація про те, які варіанти правильні – поле is_correct на бекенді не передається клієнту, щоб зберегти інтригу до завершення. Коли ж відповіді надійдуть на сервер, механізм перевірки врахує множинність: щоб відповісти правильно на таке запитання, користувач повинен вибрати усі варіанти, позначені як правильні, і не вибирати неправильних. Сервер порівнює множину отриманих від користувача відповідей із множиною правильних відповідей з БД; тільки повне співпадіння рахується правильною відповіддю. Таким чином, підтримка питань з кількома правильними відповідями реалізована як на рівні даних (через question_options.is_correct), так і на рівні

логіки перевірки результатів. Це розширює функціональність системи і наближає її до реальних потреб тестування, де нерідко зустрічаються завдання з вибором кількох правильних пунктів.

Спроєктована модель даних усуває аномалії оновлення й підтримує запитання з декількома правильними відповідями. Це забезпечує масштабованість і цілісність інформації, що стане базою для реалізації бекенду у розділі 3.

2.4 Архітектура системи

Архітектура клієнтської частини. Фронтенд — односторінковий вебдодаток (SPA) на React. Після першого завантаження HTML-файла всі переходи виконуються клієнтським JavaScript без повного перезавантаження сторінок, тому інтерфейс реагує миттєво [33, 34]. Фронтенд відповідає за відображення користувацького інтерфейсу та обробку взаємодії користувача з системою (введення даних у форми, натискання кнопок тощо), тоді як бізнес-логіка й управління даними зосереджені на бекенді. Такий підхід забезпечує чіткий розподіл обов’язків: клієнтська частина виконує роль представлення (View) і частково контролера в архітектурі додатку, підтримуючи інтерактивність та швидкий відгук інтерфейсу, а серверна частина виступає джерелом даних і правил обробки запитів.

Архітектура фронтенду побудована на компонентному підході, що надається бібліотекою **React**. Усі елементи інтерфейсу реалізовані у вигляді незалежних компонентів, кожен з яких відповідає за власну частину UI та, за потреби, власний стан. Така компонентна модель дозволяє будувати складні інтерфейси шляхом композиції простіших компонентів і багаторазово використовувати їх у різних частинах додатку. React забезпечує декларативний підхід до опису інтерфейсу, завдяки чому розробник визначає, яким має бути UI при певному стані даних, а React самостійно оновлює та перерендерює тільки необхідні частини при зміні стану. Це підвищує продуктивність та передбачуваність роботи додатку: використання віртуального DOM мінімізує

операції з реальним DOM та дозволяє швидко відображати зміни без повного перезавантаження сторінки.

Для керування станом на клієнті використано централізоване сховище на базі бібліотеки **Redux**. Цей підхід відповідає шаблону Flux і передбачає односпрямований потік даних у додатку: всі зміни стану проходять через диспатчинг відповідних дій та обробку їх редюсерами. Redux слугує єдиним джерелом істини для ключового стану інтерфейсу, зберігаючи його в одному глобальному сховищі (Store). Компоненти отримують необхідні дані зі сховища та відображають їх, а будь-які взаємодії (наприклад, авторизація користувача чи вибір відповіді на питання) генерують дії (actions), які призводять до оновлення стану через функції-редюсери. Оновлений стан автоматично передається відповідним компонентам як властивості (props), і React виконує повторний рендер потрібних частин UI. Таким чином, досягається керований цикл даних: *стан → інтерфейс → дія → новий стан*, що забезпечує узгодженість між різними частинами додатку [35].

Загальна архітектура системи має три шари — web-клієнт, Node.js + Express-API, PostgreSQL [36]. Така конфігурація концентрує бізнес-логіку на сервері, спрощує підтримку та масштабується горизонтально; саме тому її вибрано для дистанційних сценаріїв тестування. Крім того, така архітектура дає змогу організувати доступ до тестування через Інтернет, що було важливим фактором з огляду на можливість дистанційного навчання. Таким чином, реалізована система повною мірою відповідає сучасним вимогам до КСТ і рекомендаціям щодо архітектури, наведеним у роботі комп’ютерні системи тестування на основі технології трирівневих баз даних [12].

Аналіз Роботи, дозволяє виділити наступні ключові компоненти архітектури та їх взаємозв'язки:

Front-end (Клієнтська частина):

- Технологія: Односторінковий додаток (SPA) на React.
- Основні функції/компоненти:
- Відображення користувацького інтерфейсу (UI).

- Обробка взаємодії користувача.
- Клієнтська маршрутизація (React Router).
- Управління станом (Redux).
- Взаємодія з Back-end через REST API (Axios).
- Сторінки: Реєстрація, Авторизація, Домашня сторінка, Проходження тесту, Перегляд звітів, Панель адміністратора.
- Компонент ProtectedRoute для захисту маршрутів.

Back-end (Серверна частина):

- Технології: Node.js, Express.js.
- Архітектура: Багатошарова: Controller -> Service -> Repository.
- Основні модулі API:
- User Module (реєстрація, автентифікація JWT, профіль).
- Exam Module (CRUD операції над тестами та питаннями).
- Report Module (збереження та отримання результатів тестування).

Middleware:

- Логування запитів,
- CORS,
- парсинг тіла запиту (express.json),
- глобальна обробка помилок,
- JWT-автентифікація.

База Даних (Рівень зберігання даних):

- Технологія: PostgreSQL.
- Взаємодія з Back-end: Через Drizzle ORM.
- Основні таблиці: users, roles, auth, exams, questions, question_options, reports, report_answers.

Основні потоки взаємодії:

- Клієнт (React SPA) надсилає HTTP запити (JSON) до REST API (Node.js/Express).

- REST API (Node.js/Express) обробляє запити, взаємодіє з базою даних PostgreSQL через Drizzle ORM.
- Автентифікація відбувається за допомогою JWT токенів, які клієнт надсилає у заголовках запитів до захищених ендпоінтів.

Структура шарів:

• Controller — приймає HTTP-запити, повертає відповіді.

• Service — бізнес-логіка.

• Repository — доступ до БД через Drizzle ORM.

• PostgreSQL — зберігання даних.

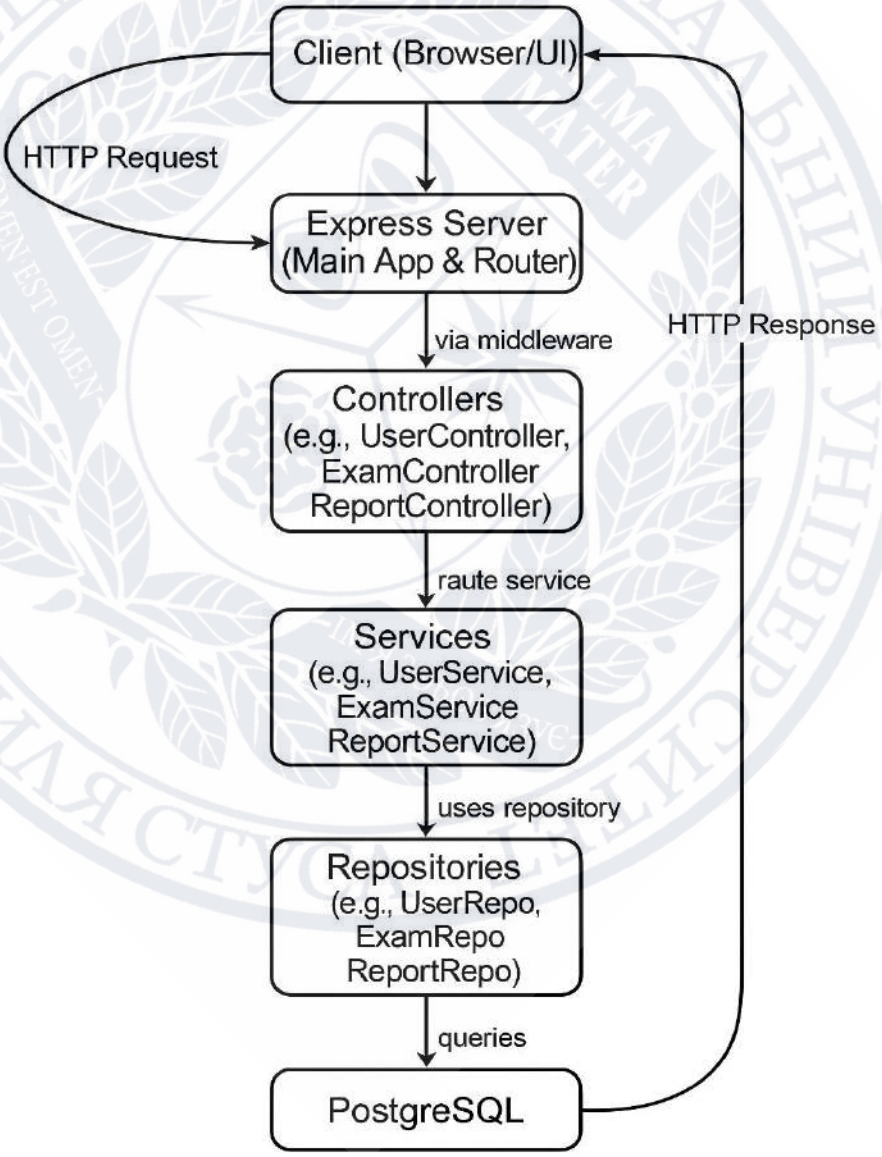


Рисунок 2.2 – Архітектура back-end частини вебдодатку з використанням багатошарової структури.

Описана архітектура є класичною трирівневою (клієнт – сервер застосунків – база даних), що забезпечує чіткий розподіл відповідальності між компонентами. Внутрішня шарувата структура бекенду (контролер-сервіс-репозиторій) є визнаним патерном проєктування, що сприяє кращій організації коду, його тестуванню та підтримці. Відображення цих аспектів на діаграмі рисунку 2.2 та рисунку А.1 [ДОДАТОК А] є важливим для демонстрації якості архітектурних рішень.

Висновок до другого розділу

У другому розділі було закладено теоретичне та проєктне підґрунтя для розробки вебсистеми для опитувань і тестування знань. На основі аналізу предметної області було сформульовано детальні функціональні вимоги, що охоплюють реєстрацію та авторизацію користувачів, управління ролями та правами доступу, створення та адміністрування тестів і питань, процес проходження тестування, а також обробку й перегляд результатів. Окрім функціональних, було визначено ключові нефункціональні вимоги, такі як зручність інтерфейсу, продуктивність, безпека, надійність та модульність системи.

Центральним елементом проєктування стала розробка реляційної моделі даних у PostgreSQL, яка була нормалізована для уникнення аномалій та забезпечення цілісності. Ця модель включає сутності користувачів, ролей, тестів, питань, варіантів відповідей та звітів, а також передбачає гнучку підтримку питань із декількома правильними відповідями.

Було описано архітектуру системи, яка базується на трирівневій моделі (клієнт – сервер додатків – база даних). Клієнтська частина реалізується як односторінковий додаток (SPA) на React, а серверна – на Node.js та Express, із застосуванням багатошарового підходу (Controller-Service-Repository). Також було деталізовано схему взаємодії між клієнтом та сервером, ілюструючи типові сценарії використання системи через послідовність HTTP-запитів та відповідей.

Таким чином, у цьому розділі створено вичерпну проєктну документацію, що визначає структуру, функціонал та архітектуру майбутнього вебдодатку і слугує основою для його подальшої реалізації.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ ДОДАТКУ

3.1 Реалізація front-end частини.

Основою клієнтської частини є бібліотека React.js, обрана через її гнучкість і популярність у розробці інтерфейсів користувача [37]. React є сучасною JavaScript-бібліотекою з відкритим кодом, яка підтримує створення багаторазових UI-компонентів і має велику спільноту розробників. Перевагою React є наявність детальної документації та численних навчальних матеріалів, що спростило процес впровадження технології в проєкт [38]. Крім того, React демонструє високу продуктивність завдяки віртуальному DOM і ефективним алгоритмам дифу елементів, що особливо важливо для інтерактивних додатків з частими оновленнями вмісту [39]. Використання JSX (розширення синтаксису JavaScript для шаблонів UI) дозволило описувати структуру компонентів у зручній декларативній формі, близькій до HTML, що полегшує підтримку та розвиток коду [40].

Для глобального управління станом інтегровано бібліотеку Redux спільно з інструментом React-Redux для зв’язку зі складовими React [35, 41]. Redux обрано для надійного керування спільним станом, зокрема даними автентифікації, профілем поточного користувача, переліком тестів, результатами тощо. Центральне сховище Redux уможливлює спростити обмін даними між далекими по дереву компонентами без передачі через каскад пропсів, а також полегшує відлагодження стану (через Redux DevTools). Односпрямованість потоку даних у Redux робить поведінку додатку більш детермінованою і полегшує написання тестів.

Як засіб організації навігації в SPA використано **React Router**. Ця бібліотека забезпечує маршрутизацію на боці клієнта: визначено набір маршрутів (шляхів URL), кожен з яких відповідає певному екрану або сторінці додатку. React Router перехоплює зміни адреси (через history API браузера) та відображає відповідні компоненти без перезавантаження сторінки. Завдяки

цьому користувач може перемикатися між розділами вебсистеми (реєстрація, вхід, проходження тесту, панель адміністратора, перегляд результатів) подібно до багатосторінкового сайту, але без затримок на повторне завантаження – потрібні компоненти вже завантажені додатком і просто підміняються вмістом центральної області. React Router підтримує *динамічні параметри* маршрутів (наприклад, ідентифікатор конкретного тесту у шляху URL /test/:id) та перенаправлення – це використовується для реалізації захищених маршрутів (Protected Routes). Зокрема, налаштовано маршрути, доступні лише після входу: якщо користувач неавторизований, при спробі перейти на такі сторінки його автоматично перенаправляє на сторінку логіну. Маршрутизатор досягає цього за допомогою спеціального компоненту **ProtectedRoute**, який обгортає приватні сторінки: він перевіряє наявність токена авторизації у локальному сховищі або стані та рендерить внутрішню сторінку лише якщо користувач залогінений; інакше – перенаправляє на головну чи показує повідомлення про відмову в доступі. Всі захищені маршрути в конфігурації роутера оголошені із використанням цього компоненту. Наприклад, маршрут панелі адміністратора може бути заданий як:

<Route path="/admin" element={<ProtectedRoute><AdminPanel /></ProtectedRoute>} />

Такий підхід на рівні фронтенду гарантує, що користувач без авторизації не побачить навіть інтерфейс внутрішніх сторінок. Крім того, **бекенд API теж перевіряє авторизацію за токеном для кожного запиту**. Отже, навіть якщо обійти перевірку фронтенду (наприклад, вручну викликати API через інструменти розробника), сервер відхилить запити без дійсного токена. Цей багатоешелонований захист підвищує безпеку системи.

Для прискорення розробки інтерфейсу та забезпечення єдиного стилю використано бібліотеку компонентів Ant Design (AntD). AntD – це набір високоякісних готових UI-компонентів для React, створений на основі однойменної дизайн-системи корпоративного рівня. У даному проєкті Ant Design надає стандартизовані елементи управління (текстові поля, кнопки,

таблиці, модальні вікна, повідомлення тощо) і гнучкий макет сітки для побудови адаптивного дизайну. Використання цієї бібліотеки дозволило зосередитися на функціональній логіці, оскільки багато аспектів стилю й поведінки елементів (наприклад, валідація полів форми, спіннери завантаження, випадаючі меню) реалізовані у готових компонентах AntD. Це суттєво підвищило швидкість розробки та забезпечило професійний вигляд інтерфейсу «з коробки». Окрім того, Ant Design підтримує адаптивність – інтерфейс коректно відображається на різних розмірах екранів без додаткових зусиль розробника [41].

Таким чином, зв'язка **React + Redux + React Router + Ant Design** утворює основу клієнтської частини вебсистеми. Додатково використовуються допоміжні бібліотеки: для виконання HTTP-запитів до сервера застосовано бібліотеку **Axios** (надбудова над браузерним Fetch API) – вона дозволяє зручно здійснювати запити до REST API і автоматично додавати заголовки (наприклад, токен авторизації) до кожного запиту. Для форматування дат/часу в інтерфейсі (наприклад, відображення дати проходження тесту у звітах) використовується або бібліотека **moment.js**, або вбудовані засоби Intl (ECMAScript Internationalization API). Збірка та транспіляція фронтенд-коду здійснюється за допомогою **Create React App** – стандартного інструменту, що налаштовує Webpack/Babel під потреби React-додатку, тому розробник міг не витрачати час на конфігурацію і одразу користуватися сучасним синтаксисом JSX/ES6+.

Основні сторінки та інтерфейси вебдодатку. Фронтенд-частина містить декілька ключових сторінок (екранів), які відповідають основним випадкам використання системи. Кожна сторінка реалізована окремим React-компонентом і відповідає окремому маршруту в React Router [42]. Нижче наведено загальний опис цих сторінок та їх функцій:

- Сторінка реєстрації. Дозволяє новому користувачеві створити обліковий запис у системі. На цій сторінці розміщена форма реєстрації, що містить поля для введення необхідних даних користувача (наприклад, ім'я, адреса електронної пошти та пароль). Поля форми мають вбудовану перевірку коректності введених даних: перевіряється заповнення обов'язкових полів,

формат email, мінімальна довжина та складність пароля тощо. Валідація виконується як на боці клієнта (засобами Ant Design Forms, які відображають підказки або помилки під полями), так і дублюється на боці сервера для гарантії цілісності даних. Після натискання кнопки «Зареєструватися» формується запит до відповідного API-методу (HTTP POST /api/auth/register) з даними користувача. У разі успішної реєстрації користувач автоматично перенаправляється на сторінку авторизації для входу в систему під щойно створеними обліковими даними. Якщо ж трапляються помилки (наприклад, обліковий запис з таким email вже існує), система відображає повідомлення про помилку без перезавантаження сторінки (відповідь сервера із кодом помилки обробляється і видається користувачу у вигляді сповіщення або текстового попередження).

Рисунок 3.1 – Сторінка реєстрації.

- Сторінка авторизації (входу). Забезпечує автентифікацію зареєстрованих користувачів. Інтерфейс сторінки містить форму входу з полями для email та пароля і кнопкою «Увійти». Коли користувач вводить свої облікові дані і підтверджує вхід, фронтенд надсилає HTTP POST запит на endpoint логіну (наприклад, /api/auth/login) з цими даними. Якщо сервер підтверджує

правильність облікових даних, у відповідь повертається токен доступу (наприклад, JWT) або інша інформація сесії, що використовується для автентифікації подальших запитів. Фронтенд зберігає отриманий токен (як правило, в LocalStorage або в пам'яті додатку) і налаштовує загальний заголовок авторизації для всіх майбутніх звернень до API. Після успішного входу здійснюється перенаправлення користувача на внутрішню захищену область – як правило, на домашню сторінку додатку. У разі некоректного логіну (невірний пароль чи email) користувач бачить повідомлення про помилку авторизації, і доступ до внутрішніх сторінок не надається. Механізм входу реалізовано таким чином, щоб зловмисник не зміг отримати доступ шляхом обходу інтерфейсу: навіть при прямому зверненні до внутрішніх маршрутів без токена користувача буде перенаправлено назад на сторінку входу (цей механізм детально описаний у розділі про ProtectedRoute).

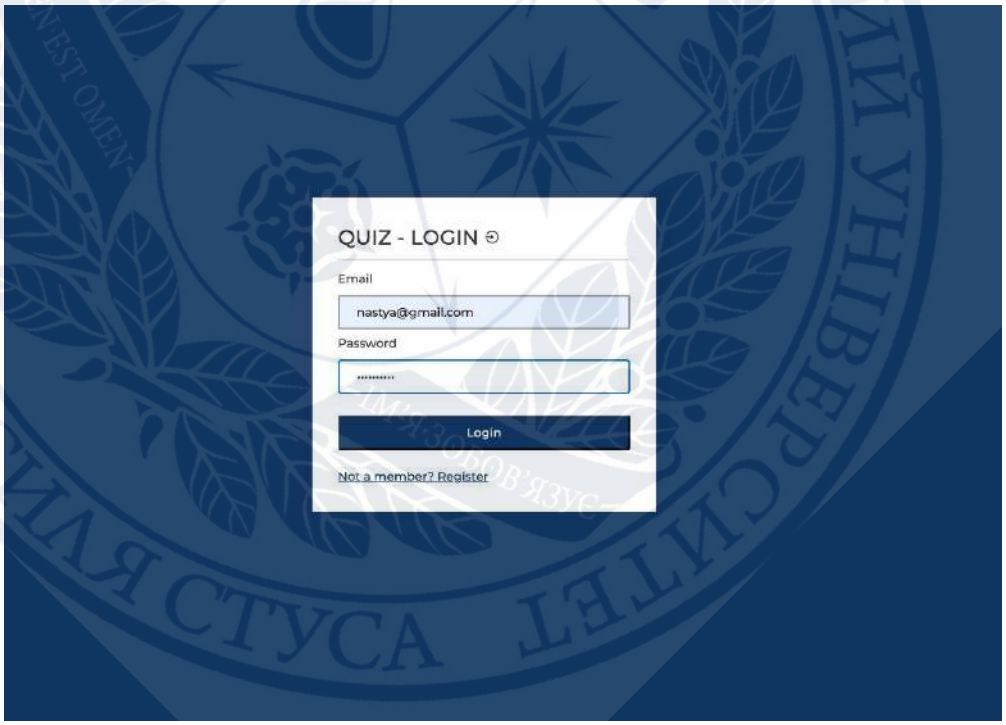


Рисунок 3.2 – Сторінка авторизації (входу).

- Домашня сторінка (після входу). Це головний екран авторизованого користувача, з якого починається робота в системі. Домашня сторінка може містити привітання та коротку інструкцію, а основним її елементом є список доступних опитувань і тестів. Цей список завантажується з серверу через API (HTTP GET запит, наприклад, на /api/exams) і відображає назви тестів, їх

короткий опис (тему, кількість питань, можливо час на проходження) та кнопку для початку проходження кожного тесту. Дані списку можуть кешуватися в стані Redux, щоб уникнути повторних запитів при навігації. На домашній сторінці користувач обирає необхідне опитування/тест із переліку. Натискання кнопки «Почати тест» ініціює навігацію на сторінку проходження тесту та одночасно може виконувати запит до сервера для отримання повної структури вибраного тесту (списку питань тощо). Окрім переліку тестів, домашня сторінка може містити меню навігації (наприклад, посилання на профіль користувача або кнопку виходу з аккаунту), а для адміністратора – посилання до панелі адміністрування.

Рисунок 3.3 – Домашня сторінка (після входу).

- Сторінка проходження тесту. Призначена для безпосереднього проведення опитування чи тестування знань. На цій сторінці фронтенд відображає питання вибраного тесту і варіанти відповідей до них. Структура тесту (список питань і опцій) завантажується з бекенду: при переході на маршрут тестування здійснюється запит (GET /api/exams/{id}) для отримання даних тесту за його ідентифікатором. Отримані дані містять перелік питань (без позначення правильних відповідей) з їхніми текстами та масивами варіантів. Після отримання цієї інформації інтерфейс генерує форму для проходження: послідовно відображає текст кожного питання та пов’язані варіанти відповіді. Користувач обирає відповіді – **для питань з одним правильним варіантом** на

інтерфейсі використовуються радіо-кнопки (відмітити можна лише один варіант із групи), **для питань із кількома правильними варіантами або опитувань без правильних відповідей** застосовуються чекбокси чи інші відповідні елементи (можна обрати кілька варіантів). Кожен вибір користувача одразу фіксується в локальному стані (наприклад, у стані компонента або в Redux, якщо це потрібно для проміжного збереження). Інтерфейс може забезпечувати зручність навігації по тесту: або відображати всі питання списком на одній сторінці (прокручуючи), або показувати по одному питанню з кнопками «Далі»/«Назад» – залежно від дизайну. Також, якщо для тесту задано обмеження часу (поле duration), на цій сторінці активується таймер зворотного відліку. Таймер реалізовано на фронтенді: при завантаженні сторінки починається відлік заданого часу (наприклад, 30 хвилин) за допомогою JavaScript (функція setInterval або React-хук useEffect). Залишок часу відображається користувачу (у форматі хв:сек) і оновлюється щосекунди. Якщо час вичерпано, система автоматично завершує тест – блокує можливість обирати відповіді та ініціює відправлення вже вибраних варіантів на сервер для перевірки. Користувач також може достроково завершити тест натисканням кнопки «Завершити» (або «Надіслати відповіді»); при цьому для підтвердження може з’явитися діалог. Після підтвердження фронтенд формує і надсилає на сервер запит з результатами (HTTP POST, наприклад, на /api/reports або /api/exams/{id}/submit), що містить ідентифікатор тесту та вибрані відповіді (наприклад, у вигляді списку пар «question_id – selected_option_id»). *Рис. 3.4 демонструє сторінку початку тестування з прикладом інтерфейсу перед стартом*.

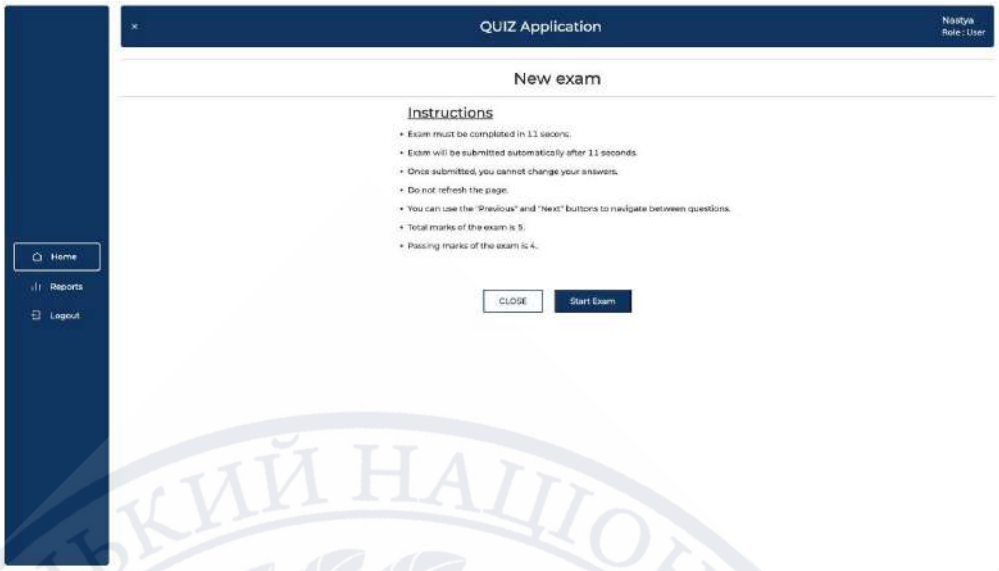


Рисунок 3.4 – Сторінка початку проходження тесту.

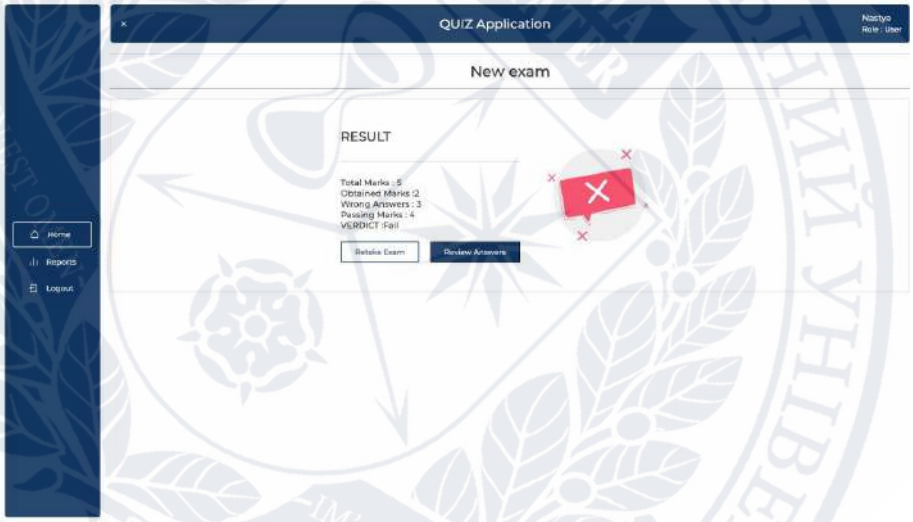


Рисунок 3.5 – Сторінка результату проходження тесту (негативний результат).

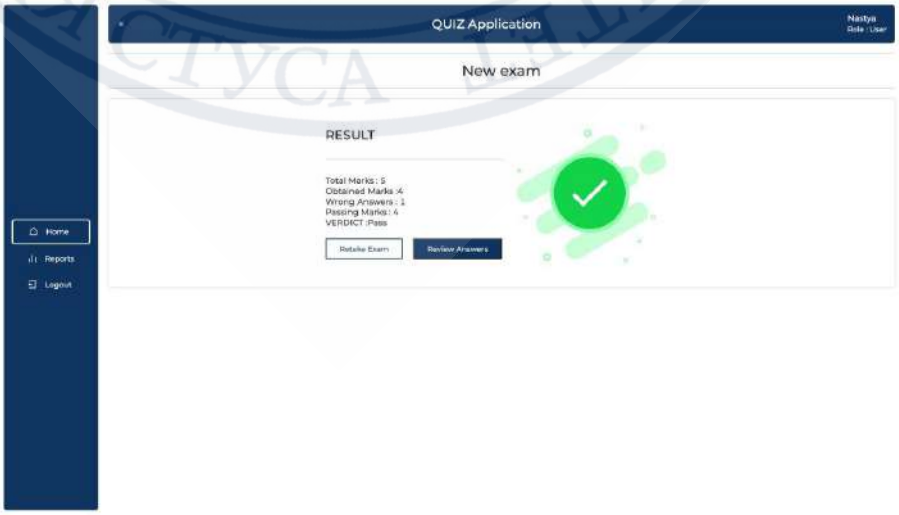


Рисунок 3.6 – Сторінка результату проходження тесту (позитивний результат).

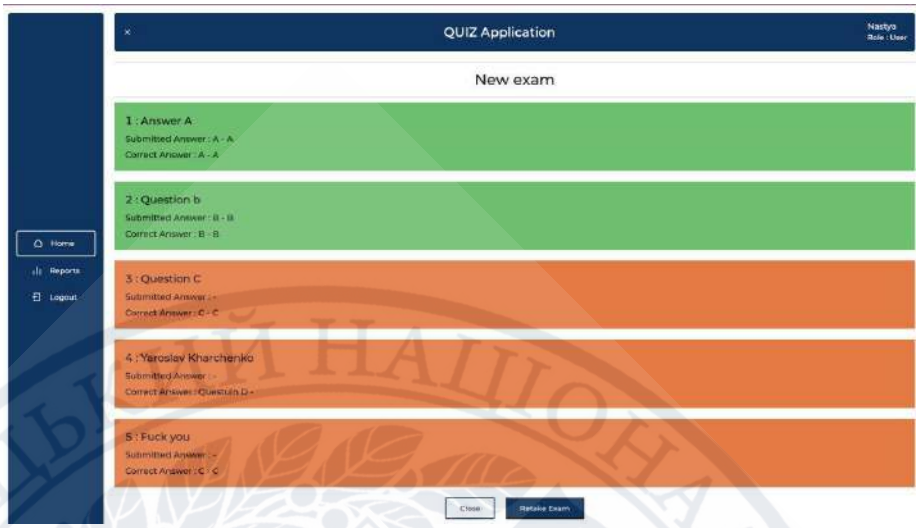


Рисунок 3.7 – Сторінка перегляду відповідей.

- Сторінка звіту (результати тестування). На цьому екрані користувач бачить підсумок пройденого тесту одразу після його завершення. Тут відображається оцінка або кількість набраних балів, відсоток правильних відповідей, а також може надаватися деталізація по питаннях: які з них були вирішені правильно, а які – ні. Якщо це передбачено сценарієм, поруч із кожним питанням може показуватися правильна відповідь для навчального ефекту. Дані для сторінки результатів надходять або безпосередньо в відповіді сервера на запит завершення (сервер повертає зведену інформацію: наприклад, {verdict: "failed", correctAnswers: 3, totalQuestions: 5}), або можуть завантажуватися окремо шляхом GET-запиту до ресурсу звітів (наприклад, /api/reports/{id} для отримання детального звіту за його ID). Сторінка побудована таким чином, щоб користувач міг проаналізувати свої помилки: для цього невірно позначені відповіді можуть підсвічуватися червоним, а правильні – зеленим, або виводитися позначки ✓/✗. Також надається можливість повернутися на головну (до списку тестів) для продовження роботи або, за наявності прав адміністратора, перейти до панелі адміністратора. *Рис. 3.5 та рис. 3.6 ілюструють приклади*

сторінки результатів для негативного і позитивного випадків, відповідно; рис. 3.7 – приклад перегляду відповідей із зазначенням правильності.

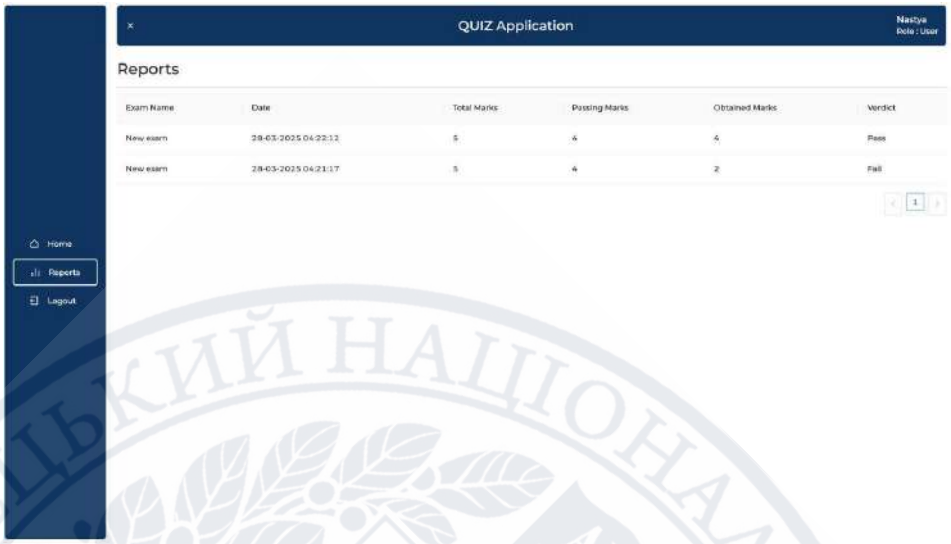


Рисунок 3.8 – Сторінка звіту (результатів тестування).

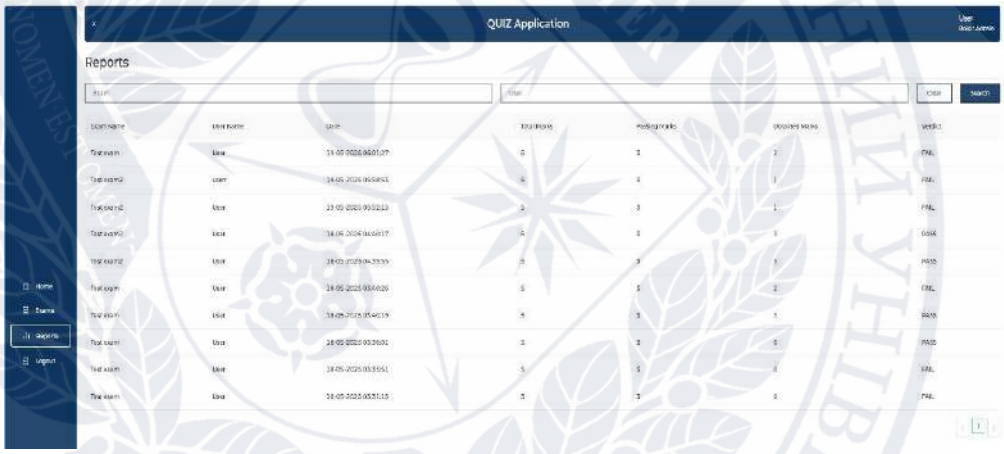


Рисунок 3.9 – Сторінка звіту (результатів тестування, адміністратор).

Панель адміністратора. Спеціальний розділ інтерфейсу, доступний лише користувачам з роллю адміністратора (наприклад, викладачам або модераторам системи). Панель адміністратора реалізована як окрема сторінка (або набір вкладених сторінок) під маршрутом /admin. Вона включає функціонал управління системою: **створення та редагування тестів і опитувань**, **перегляд результатів всіх користувачів**, **керування списком користувачів** тощо. Інтерфейс панелі складається з декількох підсторінок (вкладок або маршрутів):

- *Управління тестами:* відображається список наявних тестів із можливістю додати новий тест, відредагувати або видалити існуючий.

Адміністратор може задавати назву, категорію, параметри тесту (тривалість, бали) та статус доступності (активувати/деактивувати тест для користувачів).

- *Управління питаннями:* інтерфейс для вибору конкретного тесту і перегляду/редагування його питань. Дає змогу додавати нове питання до вибраного тесту, змінювати текст питання, його варіанти відповідей та позначки правильних відповідей. Реалізовано у вигляді модального вікна або окремої сторінки редагування питання (рис. 3.12 показує приклад модального вікна додавання питання). При додаванні питання через форму адміністратор вводить текст, додає кілька варіантів (динамічно додаючи поля для опцій) та відмічає одну чи кілька опцій як правильні. Вбудовані засоби AntD Forms забезпечують валідність введених даних (наприклад, вимагають заповнення тексту та наявність хоча б однієї правильної відповіді).
- *Перегляд звітів:* відображається таблиця зі звітами про пройдені тестування. Для кожного звіту показано ім’я користувача, назву тесту, дату проходження та отриманий результат (бал або статус). Адміністратор може відфільтрувати цю таблицю за конкретним тестом або користувачем, щоб проаналізувати успішність певного студента чи складність окремого тесту. Також передбачено можливість клікнути на конкретний звіт, щоб переглянути деталі – наприклад, які питання були помилково вирішені (можна використати вже існуючу сторінку звіту, але з точки зору адміністратора).
- *Керування користувачами (опціонально):* інтерфейс, де адміністратор може переглядати список користувачів, змінювати їхні ролі (надавати права адміністратора іншим користувачам) або блокувати/видаляти облікові записи при необхідності. Якщо ця функція реалізована, вона дозволяє адмініструвати доступ і стежити за тим, хто має права на створення контенту

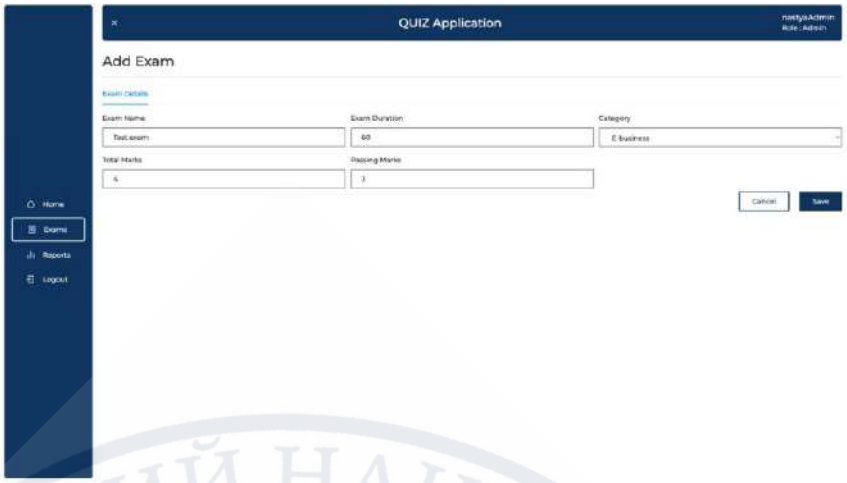


Рисунок 3.10 – Сторінка створення тесту.

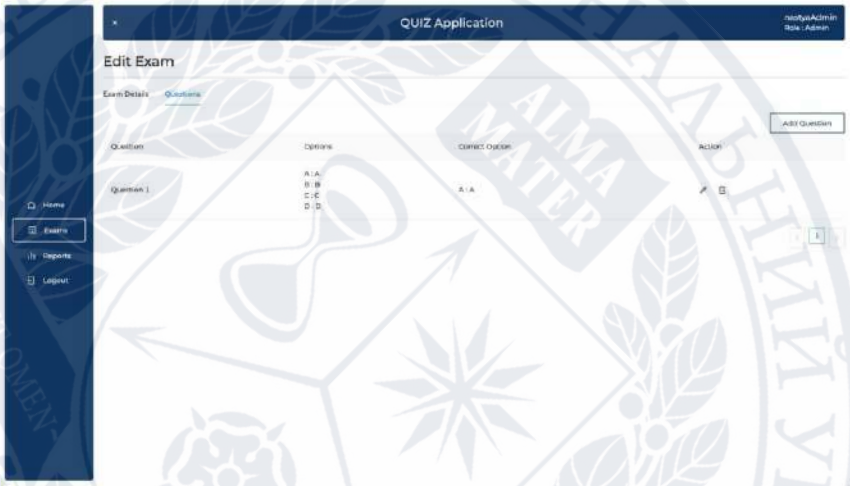


Рисунок 3.11 – Сторінка редагування тесту.

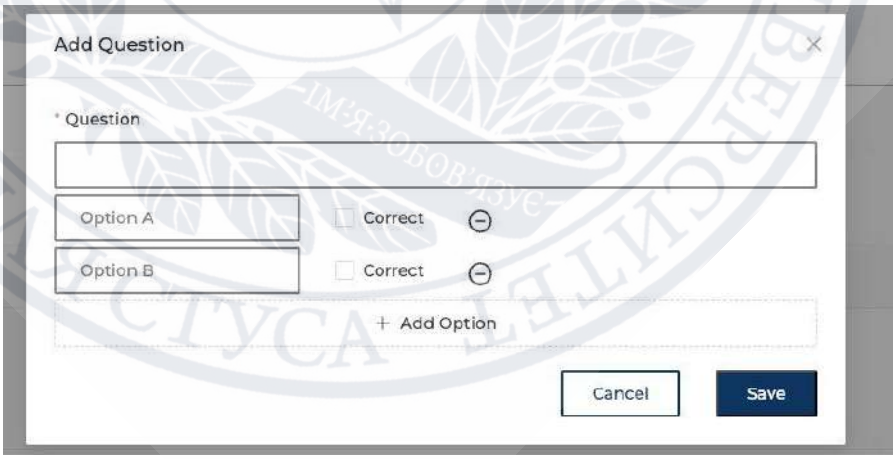


Рисунок 3.12 – Модальне вікно для додавання питання в тест.

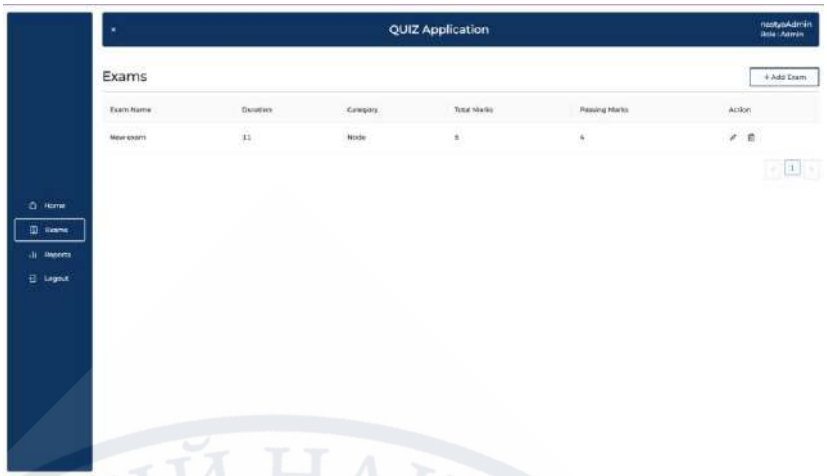


Рисунок 3.13 – Домашня сторінка з точки зору адміністратора (активна кнопка додавання тестів).

Всі адміністративні дії на фронтенді відправляються відповідними запитами до серверного API. Для узгодженості URI зазвичай використовуються REST-стилі: наприклад, створення тесту – POST запит на /api/exams з даними тесту; редагування питання – PUT запит на /api/exams/{examId}/questions/{questionId}; отримання всіх звітів – GET запит на /api/reports (з додатковими параметрами, якщо потрібно). Після успішного виконання операції бекенд повертає або оновлений об’єкт, або повідомлення про успіх, і фронтенд оновлює інтерфейс (додає новий тест до списку, відображає змінені дані тощо). Рис. 3.8–3.13 демонструють приклади інтерфейсу панелі адміністратора: список тестів, форму створення/редагування тесту, модальне додавання питання та вигляд домашньої сторінки для адміністратора з активними елементами керування.

Робота з API та обмін даними з бекендом. Взаємодія фронтенду з сервером відбувається за допомогою RESTful API, розробленого для даної системи. Клієнтська частина надсилає HTTP-запити до бекенду при виникненні певних подій або необхідності отримати/зберегти дані. Типовий сценарій обміну даними можна описати наступним чином: спочатку користувач проходить авторизацію (відправляється запит на логін, сервер повертає токен); далі, при завантаженні домашньої сторінки фронтенд робить запит GET для отримання списку доступних тестів; коли користувач обирає конкретний тест, виконується запит

GET для отримання деталей тесту (питань і варіантів відповідей); під час проходження тесту на сервер можуть надсилатися проміжні запити (наприклад, авто-збереження відповідей, якщо реалізовано), або ж усі відповіді відправляються одним запитом POST після завершення тесту; у відповідь сервер надсилає результати, які фронтенд відображає користувачу. Для виконання запитів використано стандартні методи HTTP:

- POST – для відправлення даних (реєстрація, логін, відправка відповідей, створення нових ресурсів на боці сервера),
- GET – для отримання даних (списку тестів, питань, звітів тощо),
- PUT/PATCH – для оновлення наявних даних (редагування тестів, профілю користувача),
- DELETE – для видалення (видалення тесту, питання і т.д.). Обмін даними відбувається у форматі JSON: клієнт надсилає дані у вигляді JSON-об’єктів у тілі запитів, і отримує від серверної частини JSON-відповіді [43].

На стороні фронтенду ці відповіді обробляються за допомогою вбудованих засобів JavaScript (метод response.json() для розбору) і перетворюються у відповідні структури даних (масиви, об’єкти), які потім використовуються для оновлення стану Redux або локального стану компонентів.

Важливою частиною роботи з API є керування автентифікацією при запитах. Після входу користувача фронтенд отримує JWT-токен, який додається в кожен подальший запит до захищених ресурсів. Це реалізовано через включення HTTP-заголовка Authorization: Bearer <token> у всі відповідні запити (наприклад, при зверненні до /api/exams чи інших приватних ендпоінтів). Такий токен зберігається на клієнті (в сховищі браузера) і автоматично додається або через налаштування Axios інтерцепторів, або вручну при виклику fetch. Якщо токен прострочений або недійсний, сервер відповідає кодом 401 Unauthorized, і фронтенд перехоплює цю ситуацію – очищує некоректний токен, перенаправляє користувача на сторінку входу та виводить повідомлення про необхідність повторної авторизації.

Обробка помилок та виняткових ситуацій здійснюється для кожного запиту окремо. Наприклад, якщо запит на реєстрацію повернув помилку (400 Bad Request) з повідомленням про зайнятий email, фронтенд відобразить користувачу відповідне повідомлення в формі. У разі збоїв з'єднання або помилок сервера (код 500) система також інформує користувача (наприклад, через спливаюче повідомлення про те, що сталася помилка і спробуйте пізніше). Така обробка підвищує надійність і зрозумілість роботи застосунку в очах користувача.

Захист маршрутів і компонент ProtectedRoute. Вебдодаток реалізує механізм захисту внутрішніх сторінок від несанкціонованого доступу. Деякі маршрути (зокрема, домашня сторінка, сторінка проходження тесту, панель адміністратора, перегляд звітів) мають бути доступними тільки після успішного входу користувача в систему. Для цього налаштовано так звані приватні маршрути (Protected Routes) у конфігурації React Router [42]. Принцип роботи полягає в тому, що замість прямого рендерингу компонента сторінки в маршруті, використовується обгортка – компонент ProtectedRoute, який спочатку перевіряє статус автентифікації користувача. Якщо користувач авторизований (наприклад, у глобальному стані Redux є валідний токен або об'єкт поточного користувача), то ProtectedRoute повертає цільовий компонент сторінки, дозволяючи його відображення. Якщо ж користувач не увійшов в систему, то замість контенту сторінки компонент здійснює редирект на сторінку логіну. Таким чином, навіть при спробі вручну ввести URL захищеного розділу, незалогінений відвідувач не побачить внутрішній вміст, а буде переведений на форму входу. Це гарантує, що всі конфіденційні функції (як-то проходження тестів під обліковим записом чи доступ до адмінпанелі) виконуються тільки авторизованими особами.

Практична реалізація ProtectedRoute у проєкті виконана у вигляді функціонального компонента ProtectedRoute, що використовує хук навігації React Router (useNavigate) та стан користувача з Redux (або контекст автентифікації). Він перевіряє умови: наприклад, наявність токена в сховищі або прапор isAuthenticated [43, 44]. Якщо умови не виконуються, викликається редирект navigate('/login'). Для захисту адмінмаршрутів перевіряється додатково

роль: збережена роль користувача (наприклад, user.role) звіряється із необхідною (наприклад, 'admin'), і якщо не співпадає, то користувача перенаправляють на головну чи сторінку відмови в доступі. Всі захищені маршрути в файлі конфігурації роутера оголошені із використанням цього компонента. Наприклад, маршрут панелі адміністратора може бути заданий як: <Route path="/admin" element={<ProtectedRoute><AdminPanel/></ProtectedRoute>} />. Такий підхід дозволяє централізовано визначити логіку доступу і не дублювати перевірки в кожному компоненті сторінки.

Крім захисту на рівні маршрутизатора, додатково передбачено, що бекенд API теж перевіряє авторизацію за токеном для кожного запиту. Таким чином, якщо навіть обійти фронтенд (наприклад, через інструменти розробника спробувати викликати API), сервер відхилить запити без дійсного токена. Цей багатоешелонний захист гарантує цілісність системи безпеки: фронтенд забезпечує зручність (не показує зайвого гостям), а бекенд – надійність (не дає доступу без перевірки прав).

Компонентна структура інтерфейсу та логіка роботи клієнтської частини. Інтерфейс вебдодатку спроєктовано модульно, з розбиттям на дрібні компоненти відповідно до принципів React. На верхньому рівні знаходиться компонент App, який відповідає за глобальне налаштування додатку – підключення провайдерів (Redux Provider для стану, BrowserRouter для маршрутизації), відображення спільних для всіх сторінок елементів (наприклад, шапка сайту або навігаційне меню) та визначення маршрутів. Внутрішні сторінки (реєстрація, авторизація, домашня, тест, звіт, адмінпанель) реалізовані як окремі компоненти-сторінки, що підключаються до маршрутизатора. Кожна сторінка, своєю чергою, може містити ряд підкомпонентів. Наприклад, сторінка тестування може складатися з компоненту QuestionList, який відповідає за відображення списку всіх питань, або набору компонентів Question (кожен – окреме питання з варіантами відповідей), компоненту Timer (відображає таймер і відлічує час), та компоненту SubmitButton (кнопка завершення тесту). Подібно, панель адміністратора може включати компоненти UserTable, ExamEditor,

ReportTable тощо, кожен з яких інкапсулює відповідну функціональність інтерфейсу. Такий підхід полегшує підтримку і розширення системи: окремі компоненти можна модифікувати або повторно використовувати, не зачіпаючи інші частини UI.

Управління станом компонентів здійснюється через комбінацію локального стану (hooks useState, useReducer у функціональних компонентах) та глобального стану Redux. Локальний стан застосовується для дрібних динамічних змін інтерфейсу, що стосуються тільки конкретного компоненту. Наприклад, компонент форми реєстрації може мати локальний стан для контролю вмісту полів (ім’я, email, пароль) та для відстеження повідомлення про помилку, якщо введення некоректне. Використання Ant Design Forms частково спрощує цю задачу – значення полів і їх валідація обробляються внутрішньо бібліотекою, а розробник отримує готові події на підтвердження форми. У випадку більш складних форм (наприклад, створення тесту з довільною кількістю питань) можна використовувати локальний стан масиву питань, додавати або видаляти елементи масиву при натисканні відповідних кнопок «Додати питання» тощо.

Глобальний стан Redux використовується для даних, які повинні бути доступні в різних частинах додатку. Зокрема, в стані зберігається інформація про поточного користувача (ідентифікатор, ім’я, роль, токен) – це дозволяє в будь-якому компоненті перевірити, чи залогінений користувач і які в нього права. Також у Redux стані може зберігатися вибраний тест і список його питань під час проходження, щоб компоненти питання та результат могли до нього звертатися. При отриманні з сервера результатів тесту, ці дані (наприклад, об’єкт звіту) можуть бути теж поміщені в Redux, щоб сторінка звіту отримала їх через підписку. Для керування станом визначено набір Redux-дій: наприклад, LOGIN_SUCCESS (встановлює дані користувача при вході), LOGOUT (очищає дані при виході), LOAD_EXAMS (записує список тестів, завантажених з сервера), START_EXAM (фіксує обраний тест та питання), SUBMIT_RESULTS

(зберігає отриманий звіт про результати) тощо. Редюсери обробляють ці дії і оновлюють відповідні розділи стану.

Логіка роботи з формами побудована з урахуванням UX: всі критичні дії (реєстрація, вхід, подання відповідей тесту, створення нового тесту адміністратором) супроводжуються індикацією процесу (наприклад, відображенням спіннера або деактивацією кнопки на час запиту). Це досягається додатковими змінними стану на кшталт isLoading. При надсиланні форми значення isLoading встановлюється в true, кнопка «Відправити» блокується, а після отримання відповіді – змінюється на false. У разі успіху – відбувається перенаправлення або вивід повідомлення про успішну операцію, у разі помилки – isError та відображення тексту помилки під формою.

Таймер реалізовано як окремий компонент Timer, що приймає початкове значення часу (наприклад, 30:00) та колбек на випадок завершення часу. Він використовує useEffect для запуску інтервалу (setInterval) з періодом 1 секунда, на кожному кроці зменшує значення лічильника і оновлює локальний стан компонента. Коли значення досягає нуля, компонент викликає переданий колбек (що, наприклад, ініціює автоматичну відправку відповідей) і очищує інтервал. Компонент Timer відображає поточний залишок часу у форматованому вигляді і оновлюється автоматично при зміні стану. Він вбудовується в сторінку тестування і починає відлік одразу після завантаження питань. Якщо користувач покинув сторінку або достроково завершив тест, інтервал також очищується (через очищувальну функцію в useEffect, що спрацьовує при розмонтуванні компоненту).

Перевірка результатів тестування здійснюється переважно на сервері, але фронтенд виконує важливу функцію відображення цих результатів та надання зворотного зв’язку. Після отримання від бекенду інформації про результати (наприклад, JSON об’єкта, що містить кількість правильних відповідей, оцінку, та списки ідентифікаторів питань з правильними/неправильними відповідями), фронтенд зберігає ці дані (в Redux або стані сторінки результатів) і будує інтерфейс звіту. Якщо передано детальні

дані, додаток проходить по списку питань тесту і для кожного перевіряє, чи була відповідь користувача правильною. Неправильні відповіді можуть бути виділені кольором або позначкою, а правильні – підтверджені, що дозволяє користувачу навчитися на своїх помилках. У випадку опитувань (де немає "правильно"/"неправильно") звіт просто відображає зібрані відповіді чи статистику. Всі ці операції виконуються динамічно в браузері без потреби повторного звернення до сервера, оскільки вся необхідна інформація вже була отримана в момент завершення тесту.

Загалом, компонентна структура фронтенду та продумана логіка роботи забезпечують інтерактивність і надійність системи. Користувацький інтерфейс реагує на дії негайно: при переходах між сторінками оновлюється тільки відповідний вміст (завдяки React Router), при зміні стану даних оновлюються тільки задіяні компоненти (завдяки механізму React reconcilation), а всі фонові операції (запити до API) виконуються асинхронно, не блокуючи роботу інтерфейсу [TEMP_34]. Це створює позитивний досвід для користувача, оскільки система виглядає швидкою та чуйною. Клієнтська частина тісно взаємодіє з серверною через чітко визначені API-endpoint’и, утворюючи цілісну вебсистему для проведення опитувань та тестування знань.

3.2 Реалізація back-end частини

Система має три шари — web-клієнт, Node.js + Express-API, PostgreSQL.

Побудована система підтримує кілька типів тестових завдань і режимів: зокрема, реалізовано **питання множинного вибору** (закриті питання, де потрібно обрати **одну або кілька** правильних відповідей із наведених варіантів), тестування з обмеженням часу на спробу, а також повністю автоматизоване оцінювання результатів. Усі ці можливості інтегровані в трирівневу модель: дані питань, варіантів відповідей та відмітки правильних опцій зберігаються у базі даних PostgreSQL; інтерфейс вебклієнта відображає питання та приймає відповіді; бізнес-логіка на сервері перевіряє отримані відповіді та обчислює підсумковий бал. Наприклад, для питань з кількома правильними варіантами сервер не знає наперед, скільки саме відповідей позначить користувач – він

отримує масив обраних `selected_option_id` і порівнює цю множину з еталонною множиною правильних `option_id` з бази. Якщо збігаються всі елементи (тобто користувач вибрав усе, що треба, і нічого зайвого) – відповідь вважається правильною; інакше – ні. Таким чином, процес автоматичного оцінювання повністю виконується на рівні серверу застосунків: після завершення тесту користувач майже одразу отримує свій результат (кількість набраних балів, вердикт про проходження тощо), який розрахований програмно на основі порівняння збережених у БД правильних відповідей з надісланими відповідями користувача.

Наприклад, для питань множинного вибору сервер застосунків формує HTML-сторінку з текстом питання і списком варіантів відповіді (згідно з даними з бази), яку браузер користувача відображає як тестову форму. Користувач вибирає свій варіант (або варіанти) і надсилає відповіді назад на сервер, де відбувається автоматична перевірка: система порівнює вибір студента з шаблонами правильних відповідей, що зберігаються в БД, та нараховує бали відповідно до закладених критеріїв. Процес автоматичного оцінювання повністю виконується на рівні серверу застосунків – після завершення тесту користувач одразу отримує результати (кількість набраних балів, оцінку тощо), які розраховані програмно на основі даних у БД без втручання екзаменатора.

Режим обмеження часу реалізований шляхом поєднання функцій клієнта і серверу. При старті тестування сервер фіксує часову мітку початку спроби і надсилає вебклієнту параметр – максимальну дозволену тривалість. На боці клієнта запускається таймер зворотного відліку, що інформує користувача про залишок часу. По завершенні відліку інтерфейс може автоматично завершити сесію або заблокувати можливість внесення відповідей. З боку серверу діє додатковий контроль: при отриманні відповідей перевіряється, чи не вийшов час за межі встановленого ліміту; відповіді, надіслані після закінчення відведеного часу, не приймаються системою або позначаються як протерміновані. Така двостороння реалізація гарантує дотримання часових обмежень навіть у разі спроб обійти їх на клієнтському рівні.

Таким чином, архітектура «вебклієнт – сервер – БД» забезпечує ефективну підтримку різних типів тестів і умов проведення. Вебклієнт відповідає за зручність взаємодії з користувачем, сервер застосунків реалізує всю бізнес-логіку тестування (послідовність питань, перевірку та оцінювання відповідей, контроль часу тощо), а СУБД надійно зберігає контент тестів та результати спроб. Обрана трирівнева система виявилася гнучкою і розширюваною: за потреби до неї можна додавати нові типи тестових завдань або функції (наприклад, відкриті питання з ручною перевіркою, адаптивне тестування тощо) без кардинальної перебудови архітектури – достатньо дописати відповідні модулі на рівні серверної логіки і внести зміни до структури БД. Це підтверджує обґрунтованість вибору архітектури та її відповідність завданням комп'ютерного тестування знань у сучасних умовах.

Також, архітектура серверної частини вебдодатку реалізована за принципами багатошаровості, що покращує структурованість і підтримуваність коду. Застосовано поділ на логічні модулі відповідно до ключових функціональних підсистем (користувачі, екзамени, звіти), а також виділення окремих шарів для маршрутизації (контролери), бізнес-логіки (сервіси) та доступу до даних (репозиторії). Взаємодія між цими компонентами відбувається послідовно: HTTP-запити від клієнта спочатку проходять через глобальні middleware (проміжні обробники), потім перенаправляються на відповідні маршрути контролерів, які викликають методи сервісів; сервіси, своєю чергою, звертаються до шарів збереження даних (через репозиторії або безпосередньо до моделей бази даних) і повертають опрацьовані результати назад через контролер до клієнта у вигляді HTTP-відповіді.

Для забезпечення взаємодії з клієнтською частиною, серверна частина додатку реалізує RESTful API. Це API логічно згруповане за трьома основними модулями, що відповідають ключовим функціональним блокам системи:

- Модуль користувачів (User Module): Надає кінцеві точки (endpoints) для реєстрації нових користувачів (/api/auth/register), їхньої автентифікації

(/api/auth/login) та отримання інформації про поточного користувача (/api/users/me).

- Модуль тестування (Exam Module): Включає набір методів для повного циклу управління тестами (створення, отримання списку, читання, оновлення та видалення тестів через /api/exams) та їхніми питаннями (додавання, редагування, видалення питань через /api/exams/{id}/questions).
- Модуль звітів (Report Module): Забезпечує точки доступу для збереження результатів проходження тесту (/api/reports) та отримання звітів, як загальних, так і для конкретного користувача (/api/reports з відповідними параметрами).

Така структура API дозволяє чітко розмежувати функціональність та забезпечує повне покриття вимог, визначених у розділі 2.1. Далі розглянемо реалізацію кожного з цих модулів детальніше.

Маршрутизація та модулі. Серверний додаток побудований на платформі Node.js з використанням фреймворку Express.js [46, 47]. Завдяки цьому реалізовано гнучку маршрутизацію: кожен функціональний модуль має власний роутер із набором кінцевих точок (endpoint) API. Головний файл серверного додатку імпортує ці маршрути та реєструє їх у додатку Express. Наприклад, для модуля користувачів всі URI починаються з префіксу /api/users, для екзаменів – /api/exams, для звітів – /api/reports. Це забезпечує логічне групування API та спрощує керування маршрутами. Кожен маршрут прив’язаний до певного контролера – функції, що виконується при зверненні на відповідний endpoint.

Middleware та логування. У серверному додатку застосовано middleware – проміжні прошарки обробки запитів Express. Серед них:

- Логування запитів. На рівні middleware інтегровано механізм логування (наприклад, з використанням бібліотеки winston) для реєстрації кожного запиту. Лог включає важливі параметри HTTP-звернення: метод (GET, POST тощо), адресу URI, час запиту та код відповіді сервера . Це дозволяє

відстежувати активність системи та спрощує діагностику під час розробки і в процесі експлуатації.

- Обробка CORS. Для безпечної взаємодії клієнтської частини, що може знаходитись на іншому домені, додано middleware налаштування заголовків CORS (Cross-Origin Resource Sharing). Це забезпечує браузеру дозвіл на виконання запитів до API з зовнішнього походження, що необхідно для коректної роботи вебдодатку у розподіленому середовищі [48].
- Парсинг тіла запиту. Використано вбудований в Express middleware express.json() для розбору JSON-тел запитів. Таким чином, дані форм, що надходять у форматі JSON (наприклад, при реєстрації користувача або надсиланні результатів тесту), автоматично перетворюються в об’єкти JavaScript для подальшої обробки в контролерах.

Глобальна обробка помилок. В системі реалізовано централізований механізм обробки помилок. На рівні Express-сервера зареєстровано спеціальний error-handling middleware, який перехоплює всі винятки та помилки, що не були оброблені на попередніх етапах [49]. Цей обробник формує стандартизовану відповідь про помилку: наприклад, повертає клієнту JSON-об’єкт з полями message (опис помилки) та status (HTTP-код помилки). Якщо помилка виникла з вини клієнта (некоректні дані, відсутня авторизація тощо), повертається код 4xx (наприклад, 400 або 401), якщо ж сталася внутрішня помилка сервера – код 500. Глобальний підхід до перехоплення помилок унеможливлює неконтрольоване «падіння» сервера через неочікувані винятки та забезпечує узгоджену форму відповіді на всі помилки. Логування критичних помилок також виконується: їхні описи записуються у файл журналу або консоль для подальшого аналізу розробниками.

Основний функціонал серверної частини розбитий на три самостійні модулі: модуль користувача, модуль екзаменів (тестування) та модуль звітів. Кожен з них відповідає за свій набір можливостей і має ізольований контролер, сервіс та репозиторій (відповідно до згаданої архітектури). Це дозволяє чітко

відокремити, наприклад, логіку автентифікації користувачів від логіки проведення тестів.

- **Модуль користувача (User Module).** Відповідає за реєстрацію та автентифікацію користувачів, а також за надання інформації про профіль. Реалізовано такі основні endpoints: POST /api/auth/register для створення нового облікового запису; POST /api/auth/login для входу (отримання токена доступу); GET /api/users/me (або /api/auth/me) для отримання інформації про поточного авторизованого користувача. При реєстрації контролер отримує дані від клієнта (ім’я, email, пароль), валідує їх (перевіряє формат email, мінімальну довжину паролю тощо) і звертається до сервісу користувачів. Сервіс здійснює створення нового користувача: спочатку переконується, що email унікальний (немає такого запису в auth), далі хешує пароль (використано алгоритм bcrypt) і створює запис у таблиці users та пов’язаний запис у таблиці auth. Обидві операції виконуються в одній транзакції, щоб гарантувати узгодженість (якщо один із записів не створиться, жоден не буде збережено). У випадку успіху сервіс може автоматично призначити новому користувачу роль «user» за замовчуванням (поле role_id вказує на запис ролі "user" у таблиці roles). При логіні контролер отримує email і пароль, передає їх у сервіс. Сервіс знаходить запис автентифікації за email (в таблиці auth), порівнює хеш паролю з тим, що ввів користувач (bcrypt порівняння). Якщо вони співпадають – генерується **JWT-токен** із шифрованою інформацією про користувача (наприклад, його id та роль). Генерація токена виконується з використанням секретного ключа, збереженого в конфігурації сервера. Токен відправляється клієнту у відповіді. Відтепер клієнт має включати цей токен до заголовків при виконанні захищених запитів. Також при логіні може встановлюватися HttpOnly cookie з токеном або сесійним ID – залежно від вибраної стратегії зберігання; у нашій реалізації застосовано саме Bearer JWT токен [50]. Endpoint **/me** (або /profile) повертає дані про поточного користувача. Він реалізований таким чином: клієнт надсилає GET-запит з токеном у заголовку, middleware авторизації розшифровує токен і визначає userId, після чого контролер викликає сервіс користувачів. Сервіс отримує з бази інформацію

про користувача (з таблиці users та пов’язану роль) і повертає її (крім чутливих даних – пароль не повертається). Таким чином, користувач може отримати свої дані (наприклад, для відображення імені та ролі в інтерфейсі) безпечно, тільки за наявності дійсного токена.

- **Модуль екзаменів (Exam Module).** Забезпечує функціонал створення і керування опитуваннями та тестами знань. Основні кінцеві точки API цього модуля реалізують CRUD-операції над тестами, а також операції над питаннями в рамках тестів. Зокрема: POST /api/exams – додати новий тест; GET /api/exams – отримати список усіх доступних тестів; GET /api/exams/{id} – отримати деталі конкретного тесту за його ідентифікатором; PUT /api/exams/{id} – відредагувати параметри тесту; DELETE /api/exams/{id} – видалити тест. Окрім того, передбачено вкладені ресурси для питань: POST /api/exams/{id}/questions – додати нове питання до тесту; PUT /api/exams/{id}/questions/{qId} – відредагувати формулювання чи варіанти відповіді певного питання; DELETE /api/exams/{id}/questions/{qId} – вилучити питання. Такий підхід (вкладені URI) спрощує взаємозв’язок між тестами і їхніми питаннями на рівні API та однозначно вказує, до якого тесту належить питання.

Логіка екзаменаційного сервісу включає перевірку прав доступу: додавання/редагування тестів і питань доступне лише адміністратору (або авторизованому викладачу), тоді як отримати список тестів або деталі тесту можуть усі авторизовані користувачі. Це контролюється як на рівні фронтенду (адміністратор має окремий інтерфейс), так і на рівні бекенду – наприклад, middleware може перевіряти роль користувача в токені і відхиляти заборонені дії. При створенні нового тесту сервіс формує запис у базі даних з основною інформацією (назва, категорія, тривалість, бали тощо). Відповідно до бізнес-логіки, можна одразу додати питання разом із тестом (якщо API це дозволяє – наприклад, передавши масив питань у запиті), або ж спочатку створити тест, а потім окремо надсилати запити для додавання питань. У нашій реалізації для простоти питання додаються окремими викликами. Створення питання відбувається таким чином: контролер питання отримує від клієнта текст питання,

список варіантів і позначки правильних. Сервіс екзаменів спочатку створює новий запис у таблиці questions (зв'язавши його з відповідним тестом через exam_id), отримує з БД згенерований id питання, а тоді формує кілька записів у таблиці question_options – по одному на кожен надісланий варіант відповіді. При цьому зберігається і текст варіанту, і прапорець is_correct, який передався від клієнта. Додавання всіх опцій також проводиться в транзакції, щоб гарантувати цілісність (якщо якась опція не збережеться, питання теж не буде додано). Після успішного створення повертається підтвердження (можна повернути повний перелік питань тесту, щоб фронтенд оновив свій стан). Операції редагування питання та варіантів працюють аналогічно: спочатку оновлюється текст питання (в таблиці questions), потім оновлюються або перезаписуються варіанти (у таблиці question_options). В нашому випадку для спрощення реалізації при редагуванні ми видаляємо старі варіанти питання і вставляємо нові зі зміненими полями, замість складного порівняння і оновлення кожного варіанту окремо – це припустимо, оскільки обсяг варіантів невеликий, а цілісність підтримується простіше. Вилучення питання через API (DELETE /exams/{id}/questions/{qId}) видаляє відповідний запис з таблиці questions і автоматично (через ON DELETE CASCADE) всі зв'язані з ним варіанти з таблиці question_options; сервіс може також перевіряти, чи не існує вже звітів, пов'язаних з цим питанням, і в разі небезпеки порушення історичних даних – або заборонити видалення, або помітити питання як архівне. Таким чином, модуль екзаменів охоплює весь життєвий цикл тестового контенту – від створення та редагування до надання питань для проходження. *Варто зазначити,* що для передачі питань на фронтенд сервіс **ніколи не відправляє правильні відповіді** (прапорці is_correct), щоб користувач не міг дізнатися їх до завершення тесту. Ці прапорці використовуються тільки на сервері при перевірці. Натомість клієнту надсилаються лише тексти питань та список опцій для кожного (зазвичай у випадковому порядку, якщо передбачено перемішування).

Модуль звітів (Report Module). Відповідає за фіксацію та отримання результатів проходження тестів/опитувань. Основні точки доступу: POST

/api/reports для створення нового звіту (збереження результатів після проходження тесту) та GET /api/reports для вибірки звітів. Передбачено також можливість отримати лише звіти конкретного користувача – для цього використовується параметр або окремий маршрут, наприклад, GET /api/reports?userId=... чи /api/reports/user (остання повертає результати тільки поточного авторизованого користувача). Запит на створення звіту надходить від клієнта після завершення тестування: він містить ідентифікатор тесту, а також структуру відповідей користувача. Наприклад, це може бути масив об'єктів {questionId: ..., selectedOptionId: ...}, по одному на питання, або інший формат (в нашій системі – масив ID обраних опцій для кожного питання). Сервіс звітів отримує ці дані і виконує кілька дій: перевіряє права (звичайний користувач не може записати звіт від імені іншого, userId береться з токена), обчислює результат для тестів знань, порівнюючи надіслані відповіді з правильними (дані про правильні відповіді він отримує з таблиці question_options для відповідних question_id). Підрахунок балів здійснюється шляхом проходження по кожному питанню: для кожної відповіді порівнюється множина обраних опцій з множиною правильних опцій. Якщо відповідь правильна (як описано вище – повне співпадіння), лічильник правильних відповідей збільшується. У підсумку обчислюється загальна кількість правильних (correctAnswers) і формується вердикт – наприклад, *"passed"* чи *"failed"* залежно від порогу, або просто текстове повідомлення. Сервіс створює новий запис у таблиці reports з посиланням на користувача, тест, часом і результатами (вердикт, число правильних). Паралельно він може зберегти детальні відповіді: для цього по кожному питанню зберігається запис у report_answers із зазначенням обраної опції. Таким чином, зберігається повна інформація про спробу. Потім сервіс повертає клієнту успішну відповідь, що містить зведення результату (або ID створеного звіту, за яким можна окремо довантажити деталі).

Отримання звітів (GET /api/reports) реалізовано з урахуванням ролей: звичайний користувач отримує лише свої власні звіти, адміністратор – усі звіти. Для обмеження доступу сервіс аналізує роль із токена або використовує окремі

маршрути: скажімо, /api/reports (без параметрів) – для адміністратора (повний список), а /api/reports/user – для користувача (його власні). Так чи інакше, контролер або сервіс фільтрує вибірку по user_id у разі необхідності. Отримані звіти можуть віддаватися у вигляді списку з базовою інформацією, або детально (якщо запитаний конкретний звіт). У нашій реалізації для сторінки «Мої результати» користувача робиться запит /api/reports?userId=<ID> і сервіс повертає масив його звітів із основними даними (назва тесту, дата, бал), а для перегляду деталей окремого результату (наприклад, при натисканні) може виконуватися /api/reports/{reportId} – який поверне список питань з помітками правильності та обраних відповідей (щоб відобразити, що саме було зроблено правильно чи ні).

Модуль звітів, таким чином, забезпечує зберігання цінної аналітичної інформації, яку надалі можна використовувати для оцінювання успішності навчання або аналізу даних опитування. Завдяки добре спроєктованим зв’язкам, ці дані легко вибирати: наприклад, щоб отримати історію тестувань користувача, достатньо зробити запит по його user_id; щоб отримати повний звіт, достатньо знати report_id.

Таким чином, три основні модулі бекенду (користувачі, екзамени, звіти) взаємодіють між собою і покривають увесь визначений функціонал системи: від управління обліковими записами і безпечного доступу, через створення тестового контенту, до автоматичного оцінювання, збереження та перегляду результатів.

У вебдодатку використовується реляційна база даних PostgreSQL для надійного збереження всієї інформації (контенту тестів, результатів тощо). Структура даних задана за допомогою ORM-моделей (Drizzle ORM) і включає кілька взаємопов’язаних таблиць: users, roles, user_roles, auth, categories, exams, questions, question_options, reports та report_answers. На рис. 2.1 показано логічну модель даних (діаграма класів), що відображає основні сутності (користувач, тест, питання, звіт) та зв’язки між ними. (Рисунок 2.1 потребує оновлення для

відображення актуальної структури таблиць, включаючи таблиці user_roles, categories, та зміни у зв'язках і полях існуючих таблиць відповідно до наданої схеми Drizzle ORM). Нижче описано призначення і структуру кожної таблиці.

Таблиця roles (Ролі користувачів): містить перелік можливих ролей у системі, що дозволяє гнучко керувати правами доступу. Кожен запис відповідає окремій ролі. Поля:

- id: унікальний ідентифікатор ролі (serial, первинний ключ).
- name: назва ролі (varchar, not null, наприклад, "admin" або "user"), за замовчуванням – "user".
- created_at: дата й час створення запису (timestamp, значення за замовчуванням now(), not null).
- updated_at: дата й час останнього оновлення запису (timestamp, значення за замовчуванням now(), not null).

Первинним ключем є поле id. Таблиця roles не містить прямих зовнішніх ключів до інших основних сутностей, але пов'язана з таблицею users через проміжну таблицю user_roles відношенням "багато-до-багатьох", що дозволяє одному користувачеві мати декілька ролей, а одній ролі бути присвоєною багатьом користувачам. Таблиця roles задовольняє вимоги нормалізації (1НФ, 2НФ, 3НФ), оскільки всі атрибути атомарні та повністю залежать від первинного ключа, а транзитивні залежності відсутні. Зберігання даних ролей окремо усуває аномалії оновлення.

Таблиця user_roles (Ролі користувачів - зв'язок): є проміжною таблицею, що реалізує зв'язок "багато-до-багатьох" між користувачами та ролями. Це дозволяє призначати одному користувачеві декілька ролей одночасно. Поля:

- id: унікальний ідентифікатор запису зв'язку (serial, первинний ключ).
- user_id: ідентифікатор користувача (integer, not null, зовнішній ключ, що посилається на users.id). Політика видалення: cascade (при видаленні користувача відповідні записи у user_roles також видаляються).

- role_id: ідентифікатор ролі (integer, not null, зовнішній ключ, що посилається на roles.id). Політика видалення: cascade (при видаленні ролі відповідні записи у user_roles також видаляються).
- created_at: дата й час створення запису (timestamp, значення за замовчуванням now(), not null).
- updated_at: дата й час останнього оновлення запису (timestamp, значення за замовчуванням now(), not null).

Первинним ключем є поле id. Комбінація (user_id, role_id) повинна бути унікальною, що забезпечується на рівні бізнес-логіки або додатковим унікальним індексом. Ця таблиця є класичним рішенням для реалізації зв'язку "багато-до-багатьох" і повністю нормалізована.

Таблиця users (Користувачі): зберігає облікові записи користувачів системи. Кожен запис містить базову інформацію профілю. Поля:

- id: унікальний ідентифікатор користувача (serial, первинний ключ).
- name: ім’я користувача або логін (varchar, not null).
- created_at: час створення запису (timestamp, за замовчуванням now(), not null).
- updated_at: час останнього оновлення запису (timestamp, за замовчуванням now(), not null). Первинний ключ – id. Таблиця users пов'язана з таблицею roles через проміжну таблицю user_roles (зв'язок "багато-до-багатьох").

Також з таблицею users пов’язана таблиця auth у співвідношенні "один-до-одного" (кожен користувач має рівно один запис автентифікації, що забезпечується унікальністю поля auth.user_id). Нормалізація: таблиця перебуває у 3НФ, оскільки всі поля залежать від первинного ключа id і не залежать одне від одного транзитивно. Видалення поля role_id з таблиці users та використання проміжної таблиці user_roles дозволяє уникнути аномалій при призначенні кількох ролей та спрощує управління правами доступу.

Таблиця auth (Дані автентифікації): містить облікові дані для входу в систему (логін і пароль). Рішення про відокремлення цих даних від основної таблиці користувачів (users) прийнято з кількох міркувань. По-перше, це підвищує безпеку, дозволяючи ізолювати чутливі аутентифікаційні дані (особливо хеші паролів) та потенційно застосовувати до них більш суворі політики доступу на рівні СУБД. По-друге, таке розділення забезпечує значну гнучкість для майбутнього розширення системи аутентифікації, наприклад, для підтримки декількох методів входу для одного користувача (традиційний email/пароль, OAuth через Google, Facebook, вхід за номером телефону тощо). Кожен такий метод міг би представлятися окремим записом, пов'язаним з user_id, що було б складно реалізувати в об'єднаній таблиці без порушення нормалізації або створення надлишкових полів. По-третє, це відповідає принципу єдиної відповідальності, де таблиця users зберігає профільну інформацію, а auth – виключно аутентифікаційну. Поля:

- id: унікальний ідентифікатор запису автентифікації (serial, первинний ключ).
- user_id: ідентифікатор користувача, якому відповідають ці облікові дані (integer, not null, зовнішній ключ на users.id, унікальний). Це забезпечує зв'язок "один-до-одного" між користувачем та його основними аутентифікаційними даними. Видалення користувача призводить до автоматичного видалення його запису auth (політика onDelete: 'cascade').
- email: електронна пошта користувача (varchar, not null, унікальна) – використовується як логін для входу.
- password: хеш паролю (varchar, not null). Паролі зберігаються тільки у вигляді незворотного криптографічного хешу (наприклад, bcrypt), що забезпечує конфіденційність.
- created_at: час створення запису (timestamp, за замовчуванням now(), not null). Первинний ключ – id. Зовнішній ключ user_id пов’язує запис з конкретним користувачем. Таблиця знаходиться у 3НФ.

Розділення таблиць users і auth усуває можливі аномалії та забезпечує кращу масштабованість і безпеку системи..

Таблиця exams (Тести): представляє окреме тестування або опитувальник. У ній зберігається загальна інформація про кожен тест. Поля:

- id: унікальний ідентифікатор тесту (serial, первинний ключ).
- name: назва тесту (varchar, not null), коротке текстове найменування.
- duration: тривалість тесту в хвилинах (integer, not null). Якщо встановлено обмеження часу, фронтенд забезпечує таймер.
- category_id: ідентифікатор категорії тесту (integer, not null, зовнішній ключ, що посилається на categories.id). Використання зовнішнього ключа до таблиці categories забезпечує нормалізацію даних про категорії. Політика видалення: cascade (при видаленні категорії, відповідні тести також можуть бути видалені; альтернативно, залежно від бізнес-логіки, може бути встановлено SET NULL чи RESTRICT). У наданій схемі вказано cascade.
- passing_marks: поріг проходження (integer, not null) – мінімальний бал для зарахування успішного проходження. Це значення встановлюється адміністратором і може бути абсолютним числом.
- created_at: час створення запису (timestamp, за замовчуванням now(), not null). updated_at: час останнього оновлення запису (timestamp, за замовчуванням now(), not null). Первинний ключ – id. Поле

questions (один тест містить багато питань) та з таблицею reports (один тест може мати багато звітів про його проходження). Таблиця перебуває у 3НФ.

Таблиця questions (Питання): містить перелік питань, що входять до тестів. Кожен запис відповідає одному питанню конкретного тесту. Поля:

- id – унікальний ідентифікатор питання (serial, PK).
- text – текст формулювання питання (varchar(1000), not null).
- exam_id – ідентифікатор тесту, до якого належить це питання (integer, not null, FK посилається на exams.id). Таким чином забезпечено зв’язок: кожне питання прив’язане до певного екзамену. Встановлено правило onDelete:

CASCADE – при видаленні екзамену всі його питання автоматично видаляються.

- created_at – час створення питання (TIMESTAMP).
- updated_at – час останнього редагування питання (TIMESTAMP).

Первинний ключ – id. Зовнішній ключ exam_id реалізує зв'язок «багато-до-одного» з таблицею тестів. Додатково питання пов'язані з таблицею question_options (варіанти відповідей) – одне питання може мати багато варіантів. Таблиця questions задовольняє вимоги 1НФ/2НФ/3НФ: текст питання та посилання на тест є елементарними атрибутами, що залежать тільки від id питання. Завдяки декомпозиції (винесенню варіантів в окрему таблицю) тут відсутні повторювані групи чи надлишкові дані, а зміни одного питання не впливають на інші.

Таблиця question_options (Варіанти відповідей): забезпечує зберігання варіантів відповіді для закритих питань. Раніше варіанти відповідей зберігалися як масив строк у полі питання, але для гнучкості й підтримки кількох правильних відповідей їх виділено в окрему таблицю. Кожен запис відповідає одному варіанту відповіді, прив'язаному до конкретного питання. Поля:

- id – унікальний ідентифікатор варіанта (serial, PK).
- question_id – ідентифікатор питання, до якого належить цей варіант (integer, not null, FK → questions.id). Cascading onDelete: при видаленні питання усі його варіанти теж видаляються.
- text – текст варіанта відповіді (varchar, not null).
- is_correct – позначка правильності (boolean, not null, за замовчуванням FALSE). Вказує, чи є даний варіант правильним. У типовому тестовому питанні одна чи кілька опцій матимуть значення TRUE, решта – FALSE.

Первинний ключ – id. Зовнішній ключ question_id пов'язує варіант із конкретним питанням (зв'язок багато-до-одного). Таблиця повністю нормалізована (3НФ): всі поля залежать від унікального ідентифікатора варіанта, транзитивних залежностей немає. Виділення варіантів у окрему таблицю усуває

аномалії вставлення/видалення/оновлення: можна додавати або редагувати варіанти не зачіпаючи інших питань, видалення питання автоматично очищає пов’язані варіанти, а зміна правильності окремого варіанту не створює несузгодженостей (оскільки інформація про правильні відповіді не розподілена по кількох записах або масивах).

Таблиця reports (Звіти): зберігає результати проходження тестів. Кожен запис – це результат одного завершеного тестування певним користувачем. Поля:

- id – унікальний ідентифікатор звіту (serial, PK).
- user_id – ідентифікатор користувача-респондента (integer, not null, FK → users.id). Зв’язок: один користувач може мати багато звітів. При видаленні користувача його звіти видаляються (onDelete: CASCADE).
- exam_id – ідентифікатор пройденого тесту (integer, not null, FK → exams.id). Один тест може мати багато звітів (onDelete: CASCADE для підтримання цілісності – видаляючи тест, видаляємо й пов’язані результати).
- verdict – підсумок спроби (varchar, not null, тип Verdict). Це фіксований результат на момент проходження тесту, наприклад, "passed" чи "failed".
- correct_answers – кількість правильних відповідей, отримана користувачем (integer, not null, за замовчуванням 0). Це також історичний результат, зафіксований на момент завершення тесту.
- created_at – час проходження тесту (TIMESTAMP, now() by default).

Зберігання verdict та correct_answers безпосередньо в таблиці reports фіксує історичний результат користувача на момент проходження тесту, що важливо для аудиту та відображення користувачеві того, за що він отримав оцінку. Однак, якщо в самому тесті (в таблицях questions або question_options) буде виправлена помилка (наприклад, змінено правильний варіант відповіді), ці збережені результати не зміняться автоматично. Для забезпечення можливості перегляду актуалізованих результатів, система повинна надавати функціонал (ймовірно, для адміністраторів) для динамічного перерахунку балів на основі поточного

стану питань та варіантів відповідей, використовуючи дані з таблиці report_answers. Такий перерахований результат може відображатися окремо або використовуватися для аналітики.

Первинний ключ – id. Зовнішні ключі user_id та exam_id встановлюють зв’язки «багато-до-одного» з таблицями користувачів і тестів відповідно. Таблиця reports перебуває у 3НФ (атрибути на кшталт оцінки або кількості правильних залежать тільки від конкретного звіту). Завдяки правильній структурі, операції зі звітами не викликають аномалій: не можна зберегти звіт без реального користувача чи тесту (вимога цілісності через FK), видалення користувача чи тесту автоматично знищує пов’язані звіти, а редагування інформації про тест чи користувача не призводить до втрати консистентності у результатах (оскільки у звіті зберігаються лише посилання та підсумкові дані).

Таблиця report_answers (Деталізація відповідей у звіті): реалізує зберігання конкретних відповідей користувача по кожному питанню, як доповнення до узагальненого звіту. Якщо таблиця reports містить агреговані результати (кількість правильних, оцінку тощо), то report_answers дає можливість зберегти, що саме вибрав користувач для кожного питання. Кожен запис відповідає одній відповіді на окреме питання в межах певного звіту. Поля:

- id – унікальний ідентифікатор запису відповіді (serial, PK).
- report_id – ідентифікатор звіту, до якого належить ця відповідь (integer, not null, FK → reports.id). Зв’язок: один звіт має багато записів report_answers (по одному на кожне питання тесту). Cascading delete: при видаленні звіту його деталізація теж видаляється.
- question_id – ідентифікатор питання, на яке надано відповідь (integer, not null, зовнішній ключ, що посилається на questions.id). Політика onDelete: 'cascade' означає, що при видаленні питання з системи, відповідні записи про відповіді на це питання у всіх звітах також будуть видалені. Це може призвести до втрати повноти історичних даних у звітах, якщо питання видаляється фізично. Для збереження історичної цілісності звітів, більш консервативною стратегією могло б бути логічне видалення питань (позначка як "архівне") або використання

політики ON DELETE RESTRICT чи ON DELETE SET NULL для report_answers.question_id, що потребує відповідної обробки на рівні додатку. Поточна схема передбачає каскадне видалення.

- selected_option_id – ідентифікатор вибраного варіанту відповіді (integer, зовнішній ключ, що посилається на question_options.id, може бути NULL). Якщо користувач обрав якийсь варіант, тут зберігається посилання на нього. Політика onDelete: 'set null' означає, що якщо варіант відповіді буде видалено з таблиці question_options (наприклад, через редагування питання), то в історичних звітах посилання на цей варіант обнулиться, але сам запис про відповідь на питання залишиться. Це дозволяє зберегти факт відповіді, хоча сам текст видаленого варіанту буде втрачено для цього запису.

Таблиця знаходиться у 3-й нормальній формі, оскільки кожен її атрибут описує конкретний факт про обрану відповідь і однозначно пов’язаний з ключем id. Виділення цієї таблиці усуває аномалії: наприклад, можна зберігати кілька обраних варіантів (кілька записів для одного питання в межах звіту) у разі, якщо питання передбачало декілька правильних опцій – без порушення нормалізації. Збереження детальних відповідей у цій таблиці є ключовим для можливості динамічного перерахунку результатів тестів, якщо вихідні питання або критерії оцінювання змінюються. Так само вилучення або зміна варіантів відповідей не призводить до непослідовності: історичні дані залишаються незмінними, хіба що без тексту видаленого варіанту (NULL), що є прийнятним для архівних записів.

Для зберігання даних застосовано реляційну СУБД PostgreSQL [51], а для роботи з нею на рівні коду – ORM-бібліотеку **Drizzle**. У межах проєкту визначено моделі для всіх сутностей, описаних у розділі 2.2: таблиці користувачів, ролей, автентифікації, тестів, питань, варіантів відповідей, звітів і відповідей у звітах. ORM забезпечує типізований доступ до цих таблиць, генерує SQL-запити та дозволяє оперувати записами як об’єктами JavaScript. Це значно спростило реалізацію рівня доступу до даних. Крім моделей, було використано шаблон Repository для найбільш часто використовуваних операцій – щоб не повторювати код запитів у різних сервісах. Наприклад, клас PgUserRepository містить методи

на кшталт *findById*, *findByEmail*, *create*, які інкапсулюють відповідні SQL-запити до таблиць users та auth. Аналогічно, PgExamRepository управляє вибірками і змінами в таблицях exams, questions, question_options, а PgReportRepository – в таблицях reports і report_answers. В результаті бізнес-логіка сервісів залишається читабельною і сфокусованою на правилах, а деталі збереження даних сховані всередині репозиторіїв.

На завершення варто зазначити, що бекенд дотримується принципів безпеки: перед збереженням інформації паролі шифруються (bcrypt), всі маршрути перевіряють авторизацію, вводимі дані валідуються (з використанням бібліотеки **zod** для схем), а чутливі дані (наприклад, паролі, токени) не журналюються в логах відкрито [52]. Це робить серверну частину надійною і стійкою до типових атак. Результатом роботи бекенду є стабільний REST API, який обслуговує потреби фронтенду та забезпечує весь необхідний функціонал системи тестування.

Back-end реалізовано на Node.js 14 LTS + Express 4. Шарова архітектура (Controller → Service → Repository) спрощує підтримку та тестування, Drizzle ORM забезпечує типізований доступ до PostgreSQL [49, 53, 55]. Упроваджені middleware helmet, cors, errorHandler та JWT-автентифікація гарантують безпеку [56 - 60], а централізована валідація схемами zod — цілісність даних. Побудований REST-API повністю покриває функціональні вимоги, сформульовані у розділі 2.

3.3 Взаємодія клієнт — сервер і UX-флоу

Поряд із внутрішньою логікою та даними важливим аспектом є те, як кінцевий користувач взаємодіє з системою через вебінтерфейс. У розробленому додатку клієнтська частина побудована на основі фреймворку React, що забезпечує динамічний інтерфейс і швидку реакцію на дії без перезавантаження сторінок. Взаємодія користувача з сайтом відбувається шляхом послідовності дій (кліків, введення даних), на кожен з яких фронтенд ініціює відповідні запити до серверної частини (REST API). На рис. 3.15 показано послідовність типових

запитів і відповідей між клієнтом та сервером під час проходження користувачем тесту.

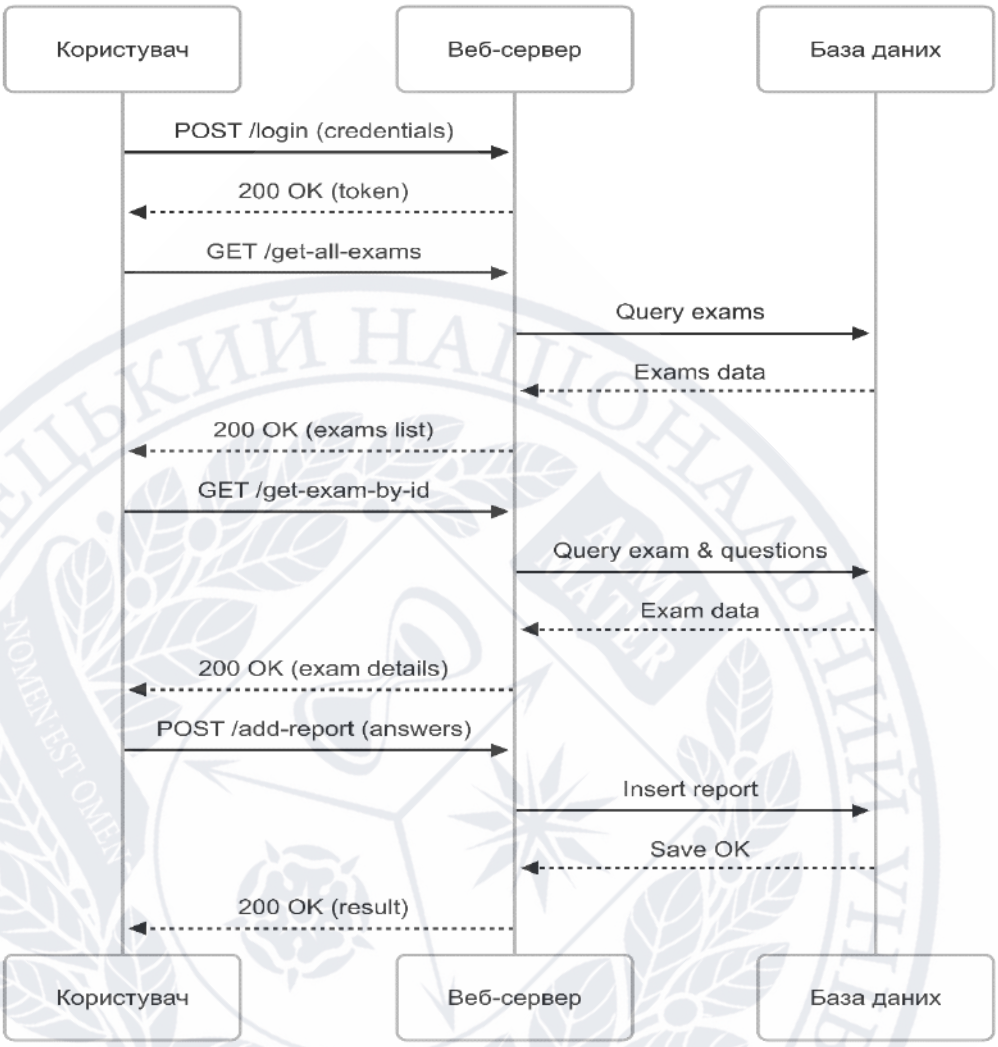


Рисунок 3.15 – Послідовність типових запитів і відповідей між клієнтом та сервером під час проходження користувачем тесту.

Діаграма відображає основні кроки: користувач вводить облікові дані для входу, обирає тест, отримує питання, надсилає відповіді і зрештою бачить результат.

Як демонструє рис. 3.15, початку користувач проходить авторизацію: вводить свій логін і пароль на сторінці входу. При натисканні кнопки «Увійти» браузер (React-додаток) відправляє HTTP-запит типу POST на endpoint /login з обліковими даними. Вебсервер (backend на Express.js) отримує цей запит, перевіряє правильність даних (звіряє email і хеш паролю з записом у базі) і у разі успіху формує позитивну відповідь (наприклад, 200 OK), що може містити токен

авторизації або сесійну інформацію [49]. Браузер отримує відповідь і, якщо вхід успішний, перенаправляє користувача до основного інтерфейсу – наприклад, на сторінку списку доступних тестів.

Далі користувач бачить перелік тестів/опитувань, що доступні для проходження (це результат попереднього GET-запиту до /get-all-exams, який фронтенд здійснює при завантаженні сторінки тестів). Користувач обирає потрібний тест зі списку і ініціює його проходження (натискає кнопку «Почати»). В цей момент фронтенд надсилає запит GET на endpoint /get-exam-by-id, передаючи ідентифікатор вибраного екзамену. Сервер звертається до бази даних, отримує об’єкт тесту та пов’язані з ним питання (як було описано в підрозділі 2.2), після чого повертає клієнту дані про тест – зокрема список питань і варіанти відповідей до них. Інтерфейс користувача динамічно відображає отримані питання: генерує форму або серію екранів, де кожне запитання представлене текстом і варіантами відповідей (наприклад, у вигляді набору радіокнопок). Користувач послідовно відповідає на кожне із питань, обираючи один або кілька варіантів (залежно від типу питання).

Після заповнення відповідей на всі питання користувач натискає кнопку завершення тесту (наприклад, «Надіслати відповіді»). Фронтенд збирає всі обрані варіанти відповідей і відправляє їх на сервер одним запитом – як правило, це POST-запит на endpoint /add-report з даними: ідентифікатор тесту, відповіді користувача по кожному питанню, та можливо токен користувача для авторизації цього запиту. Серверна частина приймає отримані відповіді і виконує дві основні дії: (1) перевіряє та оцінює результати (для тесту знань) – тобто порівнює відповіді з правильними і рахує бал; (2) створює новий запис Report у базі даних, що фіксує спробу проходження. У процесі оцінювання сервер може підрахувати відсоток правильних відповідей або визначити оцінку за наперед заданою шкалою. Сформований звіт зберігається (через звернення до бази даних PostgreSQL – вставку документа в колекцію reports). Далі сервер формує відповідь клієнту про успішне збереження результатів, додавши до неї

підрахований бал або інші дані для відображення. Браузер отримує відповідь (наприклад, 200 OK) разом з результатом тесту [44].

Завершальним кроком взаємодії є відображення результату користувачу. Інтерфейс React оновлюється, показуючи підсумкову інформацію: це може бути окрема сторінка або модальне вікно з повідомленням про результат тестування (наприклад: «Тест завершено. Ваш результат: 8/10 правильних відповідей»). Також користувачу може бути запропоновано переглянути детальніші результати або повернутися до списку тестів для продовження роботи. Якщо ж це було опитування без правильних/неправильних відповідей, інтерфейс може просто подякувати за участь і підтвердити, що відповіді збережено.

Варто зазначити, що взаємодія адміністратора з системою відбувається за аналогічною схемою запитів і відповідей, але через інші інтерфейси. Адміністратор після авторизації потрапляє до панелі управління, де може, наприклад, обрати розділ «Створити тест». Фронтенд відображає форму для введення параметрів нового тесту; після заповнення і підтвердження ця інформація відправляється POST-запитом на /add (додається новий Exam у базі). При редагуванні існуючого тесту або питань інтерфейс аналогічно збирає змінені дані і надсилає їх на відповідні endpoints (/edit-exam-by-id, /edit-question-in-exam тощо). Таким чином, усі дії адміністратора (створення тестів, питань, перегляд звітів) також здійснюються через чітку послідовність клієнт-серверних запитів.

Загалом, послідовна клієнт-серверна взаємодія гарантує швидку реакцію UI та цілісність даних. Користувацький досвід полягає у швидкому отриманні потрібних даних (питань, результатів) без зайвих перезавантажень, що робить процес проходження тестів та опитувань зручним і зрозумілим. Всі етапи – від входу в систему до отримання результатів – відпрацьовуються через передбачені сценарії взаємодії, гарантуючи коректність і повноту функціоналу, визначеного у підрозділі 2.1.

3.4 Результати роботи вебсистеми для проведення опитувань та тестування знань

Оцінка користувацького досвіду (UX) є критично важливою для успіху будь-якої вебсистеми, оскільки саме від зручності та інтуїтивності інтерфейсу залежить, наскільки ефективно користувачі зможуть взаємодіяти з її функціоналом. У цьому підрозділі проведемо самокритичний аналіз розробленої системи з точки зору потенційних користувачів – звичайного користувача (респондента) та адміністратора, спираючись на представлені у розділі 3.3 скріншоти інтерфейсів (Рис. 3.1 - 3.13).

Аналіз UX з точки зору звичайного користувача (респондента):

- **Реєстрація та вхід (Рис. 3.1, 3.2):** Процес виглядає стандартним та відносно простим. Форми містять необхідні поля, є повідомлення про помилки.
  - *Сильні сторони:* Мінімалістичний дизайн, чіткі поля.
  - *Можливі недоліки/покращення:* Варто розглянути можливість входу через соціальні мережі (OAuth) для пришвидшення реєстрації. Повідомлення про помилки могли б бути більш детальними (наприклад, конкретні вимоги до складності пароля).
- **Домашня сторінка та вибір тесту (Рис. 3.3):** Список тестів з кнопками "Start" є зрозумілим.
  - *Сильні сторони:* Наочне представлення доступних тестів.
  - *Можливі недоліки/покращення:* Не вистачає можливості сортування або фільтрації тестів (наприклад, за категорією, датою додавання, популярністю), особливо якщо тестів буде багато. Інформація про тест (кількість питань, тривалість) могла б бути більш помітною.
- **Проходження тесту (Рис. 3.4, 3.7):** Інтерфейс відображення питань та варіантів відповідей (чекбокси на Рис. 3.7) виглядає функціональним. Наявність таймера (Рис. 3.7) є позитивним моментом.
  - *Сильні сторони:* Чітке відображення питання, варіантів, таймера.
  - *Можливі недоліки/покращення:* Навігація між питаннями (якщо їх багато і вони на різних сторінках) має бути очевидною (кнопки "Далі", "Назад",

можливо, індикатор прогресу по питаннях). Необхідно забезпечити збереження прогресу у випадку випадкового закриття вкладки браузера або втрати інтернет-з'єднання, з можливістю продовжити тест. Попередження перед завершенням тесту, якщо не на всі питання дано відповідь.

- **Отримання результатів (Рис. 3.5, 3.6, 3.7, 3.8):** Відображення результату (Passed/Failed), кількості правильних відповідей та підсвічування правильних/неправильних відповідей є корисним.
  - *Сильні сторони:* Миттєвий зворотний зв'язок, візуалізація помилок.
  - *Можливі недоліки/покращення:* Для деяких користувачів може бути корисною опція "пояснення правильної відповіді" для складних питань (якщо адміністратор надасть такі пояснення). Можливість експорту або друку результатів.

Аналіз UX з точки зору адміністратора:

- **Загальна навігація та доступ до адмін-панелі (Рис. 3.13):** Кнопка "Add Exam" на домашній сторінці є прямим шляхом до однієї з ключових функцій.
  - *Сильні сторони:* Швидкий доступ до створення тесту.
  - *Можливі недоліки/покращення:* Потрібна більш структурована адмін-панель з чітким меню для доступу до всіх функцій (управління тестами, питаннями, категоріями, користувачами, перегляд звітів).
- **Створення та редагування тестів (Рис. 3.10, 3.11):** Форми виглядають комплексними, дозволяючи налаштувати різні параметри тесту.
  - *Сильні сторони:* Наявність необхідних полів для налаштування тесту.
  - *Можливі недоліки/покращення:* Форма створення/редагування тесту могла б бути розбита на логічні кроки або вкладки для кращого сприйняття. Важливою є функція попереднього перегляду тесту "очима студента" перед публікацією. Можливість копіювання існуючих тестів для створення нових на їх основі.

- **Управління питаннями (Рис. 3.12):** Модальне вікно для додавання питання виглядає функціонально.
  - *Сильні сторони:* Можливість додавати варіанти та відмічати правильні.
  - *Можливі недоліки/покращення:* Для тестів з великою кількістю питань управління ними через модальні вікна може бути незручним. Варто розглянути окрему сторінку для управління банком питань з можливістю їх масового імпорту/експорту (наприклад, з CSV або GIFT формату), пошуку та фільтрації питань, повторного використання питань у різних тестах. Підтримка різних типів питань (наприклад, на відповідність, відкриті питання з ручною перевіркою) значно розширила б можливості.
- **Перегляд звітів (Рис. 3.9):** Табличне представлення звітів з можливістю пошуку є базовим необхідним функціоналом.
  - *Сильні сторони:* Огляд результатів, пошук.
  - *Можливі недоліки/покращення:* Розширені можливості фільтрації (за датою, категорією тесту, результатом). Агрегована статистика: середній бал по тесту, розподіл оцінок, аналіз найскладніших питань (на які найчастіше дають неправильні відповіді). Можливість експорту звітів у форматах CSV/Excel для подальшого аналізу.

Загальні пропозиції щодо вдосконалення UX:

1. **Адаптивність:** Забезпечити повну адаптивність інтерфейсу для коректного відображення та зручної роботи на мобільних пристроях та планшетах.
2. **Локалізація:** Розглянути можливість підтримки кількох мов інтерфейсу.
3. **Доступність (Accessibility):** Впровадити практики вебдоступності (WCAG) для забезпечення можливості користування системою людьми з обмеженими можливостями (наприклад, підтримка навігації з клавіатури, достатній контраст, ARIA-атрибути).

4. **Система сповіщень:** Розширити систему сповіщень для користувачів (наприклад, про нові призначені тести, про завершення перевірки тестів з ручною оцінкою).

5. **Довідкова система/FAQ:** Інтегрувати розділ допомоги або підказки для користувачів щодо використання функціоналу системи.

6. **Кастомізація:** Для адміністраторів може бути корисною можливість легкої кастомізації зовнішнього вигляду тестів (наприклад, додавання логотипу).

Цей аналіз показує, що хоча базова функціональність реалізована і система є робочою, існує значний простір для покращення користувацького досвіду. Впровадження запропонованих змін та доповнень дозволить зробити систему більш зручною, ефективною та привабливою для широкого кола користувачів. Частина цих пропозицій може бути реалізована в рамках подальшого розвитку проєкту.

Висновок до третього розділу

У третьому розділі було детально описано процес практичної реалізації вебсистеми для опитувань та тестування знань, спираючись на вимоги та проєктні рішення, визначені у другому розділі.

Було реалізовано клієнтську частину (front-end) як односторінковий додаток (SPA) за допомогою бібліотеки React. Для управління станом застосовано Redux, для маршрутизації – React Router, а для побудови інтерфейсу та забезпечення єдиного стилю – бібліотеку компонентів Ant Design. Розроблено ключові сторінки: реєстрації, авторизації, домашню сторінку зі списком тестів, сторінку проходження тесту (з таймером) та сторінку результатів. Окрему увагу приділено реалізації панелі адміністратора, яка надає інструменти для управління тестами та питаннями, а також перегляду звітів. Реалізовано механізм захисту маршрутів (ProtectedRoute) для розмежування доступу. Взаємодія з сервером відбувається через REST API за допомогою бібліотеки Axios.

Серверна частина (back-end) була побудована на платформі Node.js з використанням фреймворку Express.js. Застосовано багатошарову архітектуру (Controller, Service, Repository) для чіткого розподілу логіки. Для взаємодії з базою даних PostgreSQL використано Drizzle ORM. Реалізовано основні модулі API: User (реєстрація, JWT-автентифікація), Exam (CRUD для тестів та питань, включаючи підтримку множинного вибору) та Report (збереження та отримання результатів). Впроваджено важливі middleware для логування, обробки CORS, парсингу запитів та глобальної обробки помилок. Особливу увагу приділено безпеці, зокрема хешуванню паролів (bcrypt).

На завершення розділу було представлено результати роботи системи у вигляді скріншотів основних інтерфейсів та проведено самокритичний аналіз користувацького досвіду (UX) як для звичайного користувача, так і для адміністратора. Цей аналіз дозволив виявити сильні сторони реалізованого інтерфейсу та визначити потенційні напрямки для його подальшого вдосконалення, підтверджуючи працездатність та функціональність створеного продукту.

ВИСНОВОК

У ході виконання бакалаврської роботи було успішно досягнуто основної мети – спроєктовано та розроблено вебсистему для проведення опитувань та тестування знань, яка відповідає сучасним вимогам до функціональності, зручності та безпеки.

На першому етапі було проведено комплексний аналіз існуючих систем тестування та опитувань, досліджено їхню історію, еволюцію та класифікацію. Це дозволило виявити переваги та недоліки популярних платформ (Google Forms, SurveyMonkey, Kahoot! та ін.) і сформулювати обґрунтовані вимоги до нової системи, враховуючи необхідність усунення виявлених обмежень.

У другому розділі було здійснено проєктування системи. Визначено ключові функціональні та нефункціональні вимоги. Розроблено реляційну модель даних для PostgreSQL, яка забезпечує цілісність, нормалізована до третьої нормальної форми та підтримує питання з кількома правильними відповідями. Спроєктовано трирівневу архітектуру "клієнт-сервер-БД" та деталізовано сценарії взаємодії користувача з системою.

Третій розділ був присвячений практичній реалізації спроєктованої системи. Розроблено front-end частину як SPA на React з використанням Redux, React Router та Ant Design, що забезпечило створення інтерактивного та адаптивного інтерфейсу. Реалізовано back-end на Node.js та Express.js із застосуванням багатошарової архітектури та Drizzle ORM для роботи з PostgreSQL. Впроваджено механізми безпеки (JWT, bcrypt) та логування. Було реалізовано весь запланований функціонал: від реєстрації та управління тестами до проходження тестування та аналізу результатів. Оцінка результатів роботи та користувацького досвіду підтвердила працездатність системи та дозволила виявити шляхи для її подальшого покращення.

Наукова новизна роботи полягає у застосуванні комплексного підходу, де інтегровані механізми логування та аналітики слугують не лише для моніторингу, а й як інструмент для ітеративного вдосконалення системи на

основі реальних даних про взаємодію користувачів, що підвищує її надійність та інформативність.

Практична цінність полягає у створенні готового програмного продукту, який може бути використаний у навчальних закладах, бізнесі та наукових дослідженнях для ефективного проведення тестувань та опитувань.

Перспективи подальшого розвитку включають розширення функціоналу (нові типи питань, адаптивне тестування), поглиблення аналітичних можливостей та подальше вдосконалення UX на основі зворотного зв'язку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Мороховець Г. Ю. Тестування як форма контролю та діагностики знань здобувачів вищої освіти. *Освіта та розвиток обдарованої особистості*. 2018. № 3(70). С. 11–14.

2. Шостак І. В. *Анкетування: методичні рекомендації щодо організації та проведення соціологічного дослідження*. Острог : НаУ «Острозька академія», 2021. 40 с. URL: https://eprints.oa.edu.ua/8593/1/Anketuwania_metodyczka.pdf (дата звернення: 01.06.2025).

3. Повідайчик М. М., Повідайчик О. С. Сучасні комп'ютерні технології тестування знань студентів. *Науковий вісник Ужгородського національного університету. Серія «Педагогіка. Соціальна робота»*. 2011. Вип. 21. С. 108–111.

4. Фігурська Л. В. Становлення та розвиток тестування як методу педагогічної діагностики. *Народна освіта*. 2009. Вип. 1(7). URL: http://www.narodnaosvita.kiev.ua/?page_id=191 (дата звернення: 01.06.2025).

5. PLATO. *Encyclopædia Britannica*. URL: https://www.britannica.com/topic/PLATO-education-system (дата звернення: 01.06.2025).

6. Безпалько В. П. *Педагогика и прогрессивные технологии обучения*. Москва, 1995. 336 с

7. WebCT. *Wikipedia*. URL: https://en.wikipedia.org/wiki/WebCT (дата звернення: 01.06.2025).

8. Blackboard Learn. *Wikipedia*. URL: https://en.wikipedia.org/wiki/Blackboard_Learn (дата звернення: 01.06.2025).

9. SurveyMonkey. *SurveyMonkey (офіційний сайт)*. URL: https://www.surveymonkey.com/ (дата звернення: 01.06.2025).

10. Булах І. Є. *Теорія і методика комп'ютерного тестування успішності навчання*. Київ : Освіта, 1995. 216 с.

11. Google Forms. *Wikipedia*. URL: https://en.wikipedia.org/wiki/Google_Forms (дата звернення: 01.06.2025).

12. Антонов Ю. С. Комп'ютерні системи тестування на основі технології трирівневих баз даних. *Інформаційні технології і засоби навчання*. 2008. № 2. URL: http://www.nbuv.gov.ua/e-journals/ITZN/em6/content/08aystdt.htm (дата звернення: 01.06.2025).

13. Kahoot!. *Wikipedia*. URL: https://en.wikipedia.org/wiki/Kahoot! (дата звернення: 01.06.2025).

14. Чернящук Н. Л., Бортник К. Я., Каганюк О. К., Іщук О. М., Гамонін Н. М. Аналіз web-ресурсів для організації та проведення тестів і опитувань у навчальному процесі. *Комп'ютерно-інтегровані технології: освіта, наука, виробництво*. 2021. Вип. 44. С. 76–80.

15. Юзефович К. В., Ступенко М. В. Онлайн тестування як засіб оцінювання якості знань студентів. *Актуальні питання гуманітарних наук*. 2023. Вип. 59, т. 3. С. 332–336. DOI: 10.24919/2308-4863/59-3-52.

16. Mentimeter. *Mentimeter (офіційний сайт)*. URL: https://www.mentimeter.com/ (дата звернення: 01.06.2025). (Оригінальний №17)

17. Typeform. *Typeform (офіційний сайт)*. URL: https://www.typeform.com/ (дата звернення: 01.06.2025).

18. Microsoft Forms. *Microsoft (офіційний сайт)*. URL: https://www.microsoft.com/en-us/microsoft-365/online-surveys-polls-quizzes (дата звернення: 01.06.2025).

19. Poll Everywhere. *Poll Everywhere (офіційний сайт)*. URL: https://www.polleverywhere.com/ (дата звернення: 01.06.2025).

20. Slido. *Slido (офіційний сайт)*. URL: https://www.slido.com/ (дата звернення: 01.06.2025).

21. Qualtrics. *Qualtrics (офіційний сайт)*. URL: https://www.qualtrics.com/ (дата звернення: 01.06.2025).

22. Антонов Ю. С., Смоктій К. С. Використання експертних систем для аналізу відповідей у автоматизованих системах контролю знань. *Вісник Хмельницького національного університету. Технічні науки*. 2024. № 4 (339). С. 323–331. URL:

https://heraldts.khmnu.edu.ua/index.php/heraldts/article/view/368 (дата звернення: 01.06.2025).

23. Федорук П. І. Адаптивні тести: статистичні методи обробки результатів тестового контролю знань. *Математичні машини і системи*. 2007. № 3, 4. С. 122–138. URL: http://dspace.nbuv.gov.ua/handle/123456789/778 (дата звернення: 01.06.2025).

24. Антонов Ю. С. Оцінка повноти відповідей в автоматизованих системах контролю знань. *Наукові праці ДонНТУ. Серія "Інформатика, кібернетика та обчислювальна техніка"*. 2012. Вип. 15 (203). С. 113–117.

25. Антонов Ю. С., Космінська О. М. Методика аналізу тестових завдань на основі отриманих результатів тестування. *Інформаційні технології і засоби навчання*. 2008. № 2. URL: http://www.nbuv.gov.ua/e-journals/ITZN/em6/emg.html (дата звернення: 01.06.2025).

26. Lesandrini D. Multiple Selection Through Bitmasks. *Database Journal*. 2008. URL: https://www.databasejournal.com/ms-access/multiple-selection-through-bitmasks/ (дата звернення: 01.06.2025).

27. Nielsen J. 10 Usability Heuristics for User Interface Design. *Nielsen Norman Group*. URL: https://www.nngroup.com/articles/ten-usability-heuristics/ (дата звернення: 01.06.2025).

28. Richardson L., Amundsen M., Ruby S. *RESTful Web APIs: Services for a Changing World*. Sebastopol : O'Reilly Media, 2013. 404 p.

29. Що таке файли cookie? *Cloudflare*. URL: https://www.cloudflare.com/learning/privacy/what-are-cookies/ (дата звернення: 01.06.2025).

30. Jones M., Bradley J., Sakimura N. JSON Web Token (JWT). *RFC 7519. IETF*, 2015. URL: https://www.rfc-editor.org/rfc/rfc7519 (дата звернення: 01.06.2025).

31. CAPTCHA. *Wikipedia*. URL: https://uk.wikipedia.org/wiki/CAPTCHA (дата звернення: 01.06.2025).

32. Sommerville I. *Software Engineering*. 10th ed. Harlow : Pearson, 2016. 816 p.

33. Fontaine D. *Mastering PostgreSQL in Application Development*. Birmingham : Packt Publishing, 2021. 356 p.

34. Що таке SPA у програмуванні? *Foxminded*. URL: https://foxminded.ua/spa-u-prohramuvanni/ (дата звернення: 01.06.2025).

35. Redux. *Redux.js (офіційна документація)*. URL: https://redux.js.org/ (дата звернення: 01.06.2025).

36. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures. *Doctoral dissertation*. Irvine : University of California, 2000. URL: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm (дата звернення: 01.06.2025).

37. Banks A., Porcello E. *Learning React: Modern Patterns for Developing React Apps*. 2nd ed. Sebastopol : O'Reilly Media, 2020. 307 p. (Оригінальний №54)

38. React. *React.dev (офіційна документація)*. URL: https://react.dev/ (дата звернення: 01.06.2025).

39. Boduch A. *React and React Native*. Birmingham : Packt, 2017. P. 34–45.

40. Rascia T. Understanding the DOM — Document Object Model. *DigitalOcean*, 2020. URL: https://www.digitalocean.com/community/books/understanding-the-dom-document-object-model (дата звернення: 01.06.2025).

41. Ant Design: офіційна документація UI-бібліотеки Ant Design. URL: https://ant.design/ (дата звернення: 01.06.2025).

42. React Router. *ReactRouter.com (офіційна документація)*. URL: https://reactrouter.com/ (дата звернення: 01.06.2025).

43. JSON. *MDN Web Docs*. URL: https://developer.mozilla.org/uk/docs/Learn/JavaScript/Objects/JSON (дата звернення: 01.06.2025).

44. Socrative. *Socrative (офіційний сайт)*. URL: https://www.socrative.com/ (дата звернення: 01.06.2025).

45. Як використовувати JSON Web Tokens (JWT) для автентифікації. *DevZone*. URL: https://devzone.org.ua/post/iak-vykorystovuvaty-json-web-tokens-jwt-dlia-avtentyfikatsiyi (дата звернення: 01.06.2025).

46. Node.js. *Node.js (офіційний сайт)*. URL: https://nodejs.org/ (дата звернення: 01.06.2025).

47. Express. *Express.js (офіційна документація)*. URL: https://expressjs.com/ (дата звернення: 01.06.2025).

48. Cross-Origin Resource Sharing (CORS). *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS (дата звернення: 01.06.2025).

49. Drizzle ORM. *Drizzle ORM (офіційна документація)*. URL: https://orm.drizzle.team/docs (дата звернення: 01.06.2025).

50. Chukhray A., Yashina E. Models and software for intelligent web-based testing system in mathematics. *CEUR Workshop Proceedings*. 2021. Vol. 3003. URL: http://ceur-ws.org/Vol-3003/paper1.pdf (дата звернення: 01.06.2025).

51. Bcrypt. *Crypto node.js (офіційний сайт)*. URL: https://nodejs.org/api/crypto.html (дата звернення: 01.06.2025).

52. Zod. *Zod (офіційний сайт)*. URL: https://zod.dev (дата звернення: 01.06.2025).

53. Casciaro M., Mammino L. *Node.js Design Patterns: Design and implement production-grade Node.js applications using proven patterns and techniques*. 3rd ed. Birmingham : Packt, 2020. 664 p.

54. Helmet.js. *Helmet.js (офіційний сайт)*. URL: https://helmetjs.github.io/ (дата звернення: 01.06.2025).

55. PostgreSQL. *PostgreSQL (офіційний сайт)*. URL: https://www.postgresql.org/ (дата звернення: 01.06.2025).

56. JWT. *JSON web token. (офіційний сайт)*. URL: https://jwt.io/introduction (дата звернення: 01.06.2025).

57. Helmet.js: офіційна документація модуля Helmet для Express.js. URL: https://helmetjs.github.io/ (дата звернення: 01.06.2025).

58. Masse M. *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces*. Sebastopol : O'Reilly Media, 2011. 112 p. ISBN 9781449310509.

59. REST APIs. *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Glossary/REST (дата звернення(Оригінальний ня: 01.06.2025).

60. Helmet.js. *Helmet.js (офіційний сайт)*. URL: https://helmetjs.github.io/ (дата звернення: 01.06.2025).

61. Obe R., Hsu L. *PostgreSQL: Up and Running: A Practical Guide to the Advanced Open Source Database*. 3rd ed. Sebastopol : O'Reilly Media, 2021. 192 p.

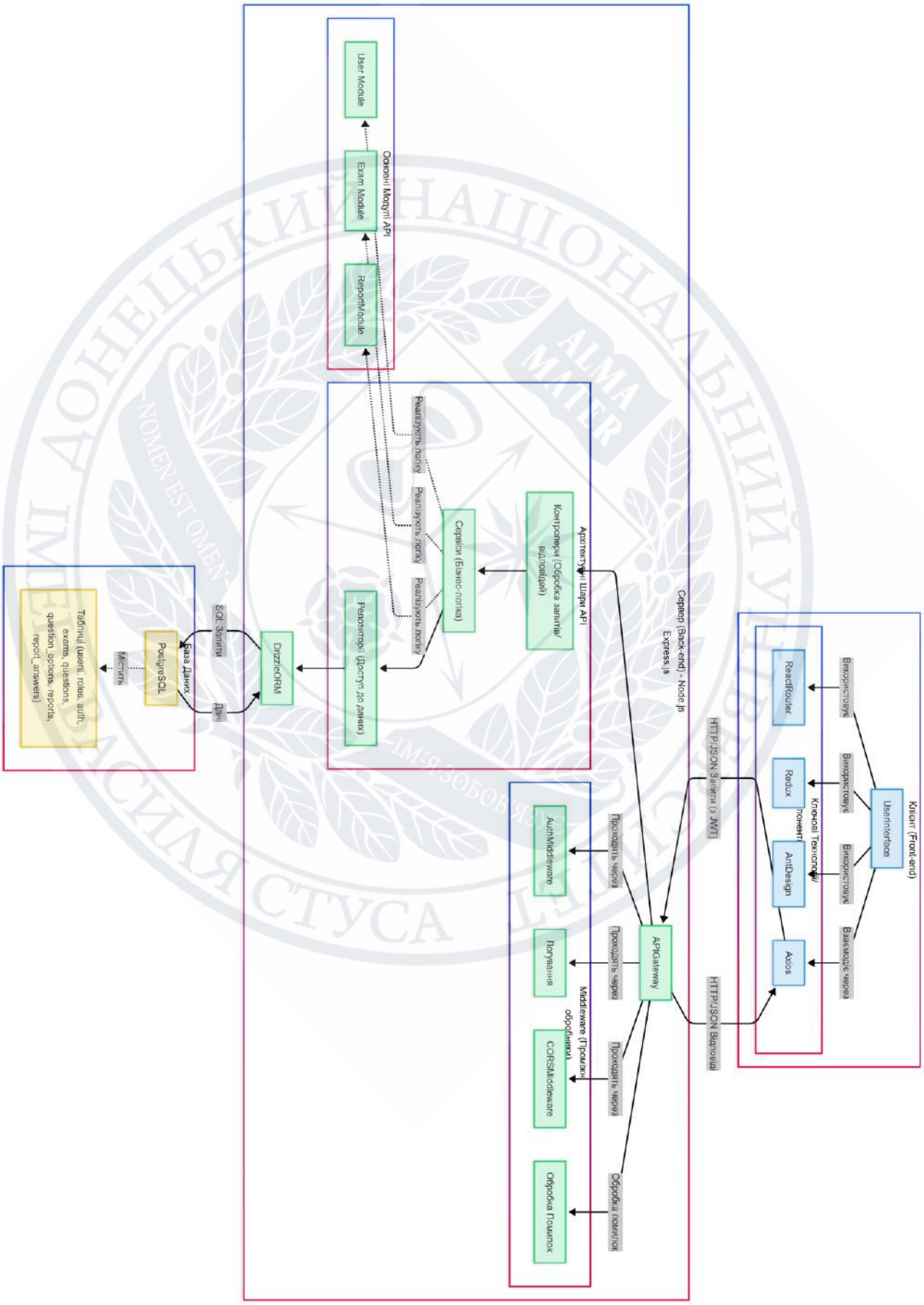
62. Krug S. *Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability*. 3rd ed. Berkeley : New Riders, 2014. 200 p.

63. Sue V. M., Ritter L. A. *Conducting Online Surveys*. 2nd ed. Thousand Oaks : SAGE Publications, 2012. 264 p.

64. HTTP authentication. *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/Authentication (дата звернення: 01.06.2025).

ДОДАТОК А

Рисунок А.1– Класична трирівнева архітектура back-end частини вебдодатку



ДЕКЛАРАЦІЯ

про дотримання академічної доброчесності

Я, _____

_____

_____

_____

_____

Повністю вказується ПІБ та статус (освітня (освітньо-наукова) програма – для здобувачів вищої освіти, назва кваліфікаційної роботи)

що нижче підписалась/підписався, розуміючи та підтримуючи загальновизнані засади справедливості, доброчесності та законності,

ЗОБОВ'ЯЗУЮСЬ:

дотримуватися принципів та правил академічної доброчесності, що визначені законодавством України, локальними нормативними актами Донецького національного університету імені Василя Стуса, положеннями, правилами, умовами, визначеними іншими суб'єктами, та не допускати їх порушення.

ПІДТВЕРДЖУЮ:

що мені відомі положення статті 42 Закону України «Про освіту»;

що у даній роботі не представляла/представляв чиїсь роботи повністю або частково як свої власні. Там, де я скористалася/скористався працею інших, я зробила/зробив відповідні посилання на джерела інформації;

що дана робота не передавалась іншим особам і подається вперше, не порушує авторських та суміжних прав закріплених статтями 21-25 Закону України «Про авторське право та суміжні права», а дані та інформація не отримувались в недозволений спосіб.

УСВІДОМЛЮЮ:

що ця робота може бути перевірена університетом на плагіат або інші порушення академічної доброчесності, в тому числі з використанням спеціалізованих сервісів;

що у разі порушення академічної доброчесності, до мене можуть бути застосовані процедури, передбачені законодавством України та Кодексом академічної доброчесності та корпоративної етики Донецького національного університету імені Василя Стуса, іншими локальними нормативними актами університету, та я можу бути притягнута/притягнутий до академічної відповідальності.

_____ _____

(дата) (підпис)

ДОНЕЦКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВАСИЛЯ СТУСА
ALMA MATER
NOMEN EST OMEN
ІМ'Я ЗОБОВ'ЯЗУЄ