Міністерство освіти і науки України

Кам’янець-Подільський національний університет імені Івана Огієнка

Фізико-математичний факультет

Кафедра комп’ютерних наук

Кваліфікаційна робота бакалавра

з теми: «**АВТОМАТИЗОВАНА СИСТЕМА УПРАВЛІННЯ КУРСАМИ ПІДВИЩЕННЯ КВАЛІФІКАЦІЇ ПЕДАГОГІЧНИХ ПРАЦІВНИКІВ ТА КЕРІВНИКІВ ЗО»**

Виконала: здобувач вищої освіти групи KN1-B21

спеціальності 122 Комп’ютерні науки

Сурженко Юлія Віталіївна

Керівник:

Пилипюк Тетяна Михайлівна, кандидат фізико-математичних наук, доцент

Рецензент:

Громик Андрій Петрович, кандидат технічних наук, доцент

м. Кам’янець-Подільський – 2025 р.

АНОТАЦІЯ

У кваліфікаційній роботі розроблено веборієнтовану систему для автоматизації процесу управління курсами підвищення кваліфікації та генерації іменних сертифікатів на їх завершення. Основна увага приділена створенню зручного інтерфейсу для адміністраторів та впровадженню функціоналу імпорту/експорту даних учасників та програм із шаблонних файлів (Excel, Word).

Розробку реалізовано за клієнт-серверною архітектурою, яка включає окремі клієнтські частини для адміністраторів, учасників та розробників програм. Такий підхід забезпечив масштабованість і гнучкість системи, дозволивши розширити її функціонал без порушення загальної структури. На серверному рівні реалізовано базу даних MySQL зі структурованими таблицями для зберігання всієї інформації, що надходить із форм. Система дозволяє автоматично генерувати персоналізовані сертифікати у форматі DOCX, об’єднувати їх у ZIP-архів, а також надає інтерфейс для їх перегляду та завантаження.

Система рекомендована для використання у Центрах підвищення кваліфікації, освітніх установах та інших організаціях, де передбачена видача сертифікатів учасникам курсів.

Ключові слова: автоматизація, генерація сертифікатів, вебсистема, Node.js, React, база даних, підвищення кваліфікації.

ABSTRACT

This qualification thesis presents a web-based system for automating the management of professional development programs and the generation of personalized certificates upon their completion. The work focuses on developing a user-friendly interface for administrators and implementing functionality for importing/exporting participant and program data from template files (Excel, Word).

The system is built on a client-server architecture, which includes separate client interfaces for administrators, participants, and program developers. This approach ensures scalability and flexibility, allowing for future expansion without disrupting the system's core structure. On the server side, a structured MySQL database stores all incoming information. The system automatically generates personalized DOCX certificates, combines them into a ZIP-archive, and provides a user interface for preview and download.

The system is recommended for use in professional development centers, educational institutions, and other organizations issuing course completion certificates.

Keywords: automation, certificate generation, web system, Node.js, React, database, professional development.

ЗМІСТ

ВСТУП

Актуальність теми. Актуальність теми зумовлена цифровізацією освіти та зростанням онлайн-навчання, що робить важливою автоматизацію рутинних завдань, зокрема формування сертифікатів про проходження курсів. Зазвичай цей процес виконується вручну, що забирає багато часу, підвищує ризик помилок і вимагає значних людських ресурсів. Існуючі рішення не підтримують повну автоматизацію обробки зовнішніх даних (Excel, Word). Тому створення вебзастосунку для автоматичного формування сертифікатів є актуальним та необхідним для освіти й професійного розвитку.

Проблема, що розв’язується, — розробка системи, яка інтегрує зовнішні джерела даних, автоматично обробляє інформацію, формує сертифікати за шаблонами та зберігає результати у централізованій базі. Практична цінність полягає в можливості впровадження рішення в різних освітніх організаціях для зниження витрат часу та підвищення точності документообігу.

Об’єкт і предмет дослідження. Об’єктом дослідження є програмна реалізація автоматизованої системи управління курсами підвищення кваліфікації. Предметом дослідження є створення вебзастосунку для генерації сертифікатів на основі обробки зовнішніх файлів (Excel, Word) із використанням сучасних технологій веброзробки.

Мета і завдання дослідження. Метою дослідження є розробка повнофункціонального вебзастосунку для автоматизованого формування сертифікатів з можливістю обробки даних із зовнішніх джерел, управління шаблонами та збереження результатів у базі даних.

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

1. Провести аналіз існуючих рішень для автоматизації генерації сертифікатів.
2. Визначити вимоги до функціональності, безпеки та масштабованості системи.
3. Обрати відповідну архітектуру та стек технологій для реалізації клієнтської та серверної частин.

4. Спроєктувати структуру бази даних.
5. Реалізувати модулі завантаження та обробки даних з Excel- і Word-файлів.
6. Розробити модуль генерації сертифікатів на основі шаблонів.
7. Розробити інтерфейс адміністратора для керування даними та шаблонами.
8. Провести тестування функціоналу системи та оцінити її ефективність.

Методи дослідження. У процесі роботи застосовувались такі методи як аналіз літератури та відкритих джерел для дослідження існуючих рішень, метод системного аналізу для формування функціональних вимог та метод моделювання для побудови структури даних та взаємодії компонентів.

Практичне значення одержаних результатів. Розроблений вебзастосунок дозволяє значно оптимізувати процес формування сертифікатів, зменшити кількість помилок, викликаних людським фактором, а також інтегрувати систему у внутрішні процеси організацій, що проводять навчання. Рішення може бути адаптоване під конкретні вимоги освітніх установ або корпоративних платформ для підвищення кваліфікації персоналу. Система готова до впровадження та має модульну архітектуру, що полегшує подальше масштабування і модифікацію.

Апробація результатів. Результати досліджень були оприлюднені на науковій конференції за підсумками науково-дослідної роботи здобувачів вищої освіти фізико-математичного факультету Кам’янець-Подільського національного університету імені Івана Огієнка у 2024-2025 н.р., 9-10 квітня 2025 (тези доповіді) [1] та у Віснику Кам’янець-Подільського національного університету імені Івана Огієнка. Фізико-математичні науки. Випуск 17 (стаття) [2].

Структура роботи. Кваліфікаційна робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків. У розділах послідовно розкриваються теоретичні засади дослідження, аналізуються технічні рішення, реалізація системи та її практичне застосування.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ. ПРОЄКТУВАННЯ СИСТЕМИ

У цьому розділі описано предметну область систем управління курсами підвищення кваліфікації, виявлено їхні недоліки, сформульовано вимоги та описано основні функції системи. Також розглянуто архітектуру, базові компоненти, принципи побудови бази даних, логіку взаємодії модулів і підключення сторонніх сервісів. Структура системи має бути гнучкою, зрозумілою, захищеною і підтримуваною.

1.1. Аналіз існуючого процесу управління курсами підвищення кваліфікації. Формування вимог до автоматизованої системи

Програмний продукт створено для Центру підвищення кваліфікації педагогів, які надають освітні послуги для вчителів, вихователів і керівників закладів освіти. Під час атестації педагогам необхідно підтвердити підвищення кваліфікації сертифікатом.

Наразі процес формування сертифікатів здебільшого ручний і потребує багато часу та ресурсів. Для кожного слухача сертифікат готують у Word, вносячи особисті дані, назву програми та результати тестів. Дані зберігаються в окремих Excel та Word файлах, що ускладнює обробку.

Поточна процедура включає в себе ручний пошук інформації про кожного учасника курсу, ідентифікацію відповідної програми, відкриття та аналіз пов’язаного документа з результатами навчання, після чого – вставку даних у шаблон сертифіката. Усе це призводить до:

- високої трудомісткості процесу;
- значної вірогідності помилок;
- відсутності єдиної структури документів;

- ускладненого масштабування процесу в разі збільшення кількості слухачів.

З огляду на вищезазначене, постає нагальна потреба у створенні автоматизованої системи, яка б забезпечувала:

- централізоване зберігання даних про учасників та програми;
- імпорт інформації з Excel-файлів;
- обробку результатів із Word-документів;
- динамічне формування сертифікатів за шаблоном.

Впровадження автоматизованої системи підвищить ефективність Центру, зменшить вплив людського фактора, прискорить підготовку сертифікатів і покращить точність та стандартизацію документообігу. На основі аналізу предметної області та недоліків сформульовано вимоги до нової системи для автоматизації управління курсами (таблиця 1.1).

Таблиця 1.1

Вимоги до системи сформовані попередньо

№	Функціональна можливість	Опис
1	Завантаження даних	Учасники заповнюють форму де вказують усі потрібні Центру дані про учасника. Є можливість сформувати Excel файл із цими даними. Розробники програм заповнюють форму де вказують усі потрібні Центру дані про програму. Є можливість сформувати Word файл із цими даними для кожної програми окремо на базі шаблону. Адміністратори можуть створювати/редагувати/видаляти ці дані
2	Витяг інформації	Система автоматично витягує програми та список учасників з помісячним поділом

3	Генерація сертифікатів	Створення docx-файлів для кожного учасника з унікальними даними
4	Перегляд/завантаження	Сертифікати можна переглянути та зберегти в одному архіві
5	Авторизація	Безпечний вхід у систему адміністраторів

Після представлення даних функціональних можливостей було обговорено та враховано побажання працівників Центру та відповідно цього сформульовано нові вимоги до системи, орієнтованої на автоматизацію процесу генерації сертифікатів (таблиця 1.2):

Таблиця 1.2

Вимоги до системи, сформовані на базі відгуку працівників Центру

№	Функціональна можливість	Опис
1	Завантаження даних	Адміністратор завантажує Excel-файл зі списком учасників та Word-файл з шаблоном сертифікату На окремій сторінці відбувається підвантаження назви програми та її результатів
2	Витяг інформації	Система автоматично розпізнає зміст програми та список учасників
3	Генерація сертифікатів	Створення docx-файлів для кожного учасника з унікальними даними
4	Перегляд/завантаження	Сертифікати можна переглянути та зберегти в одному архіві
5	Авторизація	Безпечний вхід у систему адміністраторів

Також варто врахувати нефункціональні вимоги до системи:

- Простота використання для адміністраторів без технічних знань.
- Безпека персональних даних (захист даних учасників).
- Швидка генерація навіть великої кількості сертифікатів.
- Стандартизована база даних (наприклад, MySQL).

1.2. Архітектурні рішення

Архітектура програмного забезпечення — це структура системи, що описує взаємодію її компонентів і зв’язок із зовнішніми системами. Вона будується за допомогою архітектурних шаблонів — це як план будинку, але замість матеріалів — код, бази даних і сервери. Архітектура визначає стабільність, гнучкість, безпеку та можливість розвитку системи. Далі розглянемо основні типи архітектури ПЗ, які нині застосовуються.

Монолітна архітектура означає, що вся програма складається з єдиного блоку коду, де всі частини взаємодіють прямо одна з одною. Якщо це велика програма, у ній є всі можливості та функції в одному місці (Додаток А).

Переваги:

- Простота розробки, особливо на початку.
- Легше тестувати та запускати програму, оскільки вона одна ціла.
- Чітке уявлення про всі взаємозв'язки в програмі.

Недоліки:

- Зі зростанням системи код стає громіздким і важким для підтримки.
- Зміна однієї частини може вплинути на інші частини, що може призвести до помилок.
- Складно масштабувати окремі функції системи.

Мікросервісна архітектура поділяє систему на багато дрібних автономних частин (мікросервісів), де кожен виконує окрему функцію. Це фактично міні-програми. Ці частини взаємодіють між собою через спеціальні протоколи (наприклад, через API) (Додаток Б).

Переваги:

- Легше масштабувати окремі частини системи.
- Можна оновлювати або змінювати один мікросервіс, не торкаючись інших.
- Простота в підтримці великих команд розробників, кожна з яких може працювати над своїм мікросервісом.

Недоліки:

- Складність у налаштуванні взаємодії між мікросервісами.
- Потрібно добре продумувати тестування і безпеку, оскільки система складається з багатьох частин.
- Ускладнене управління інфраструктурою, оскільки кожен мікросервіс потребує окремих ресурсів.

Клієнт-серверна архітектура – архітектура, яка складається з двох частин: клієнт (зазвичай це те, що бачить користувач – наприклад, браузер) і сервер (місце, де зберігаються дані і виконується основна логіка) (Додаток В).

Переваги:

- Чітке розділення на частини: клієнт для взаємодії з користувачем, сервер для обробки даних.
- Сервер можна легко оновлювати і змінювати, не зачіпаючи клієнт.

Недоліки:

- Якщо сервер виходить з ладу, клієнт не може працювати.
- Клієнт і сервер повинні постійно взаємодіяти, що може створювати проблеми з продуктивністю або з'єднанням.

Подійно-орієнтована архітектура (Event-driven architecture) – архітектура, яка побудована навколо подій – маленьких повідомлень, що відправляються, коли щось відбувається в системі. Різні компоненти (сервіси) реагують на ці події (Додаток Г).

Переваги:

- Висока продуктивність, оскільки сервіси реагують на події тільки тоді, коли вони відбуваються.
- Легко масштабувати та розширювати, додаючи нові обробники подій.

Недоліки:

- Важче тестувати та відстежувати всі події.
- Якщо події обробляються неправильно, це може створити "хаос" у системі.

Сервіс-орієнтована архітектура (SOA) – архітектура, де всі частини системи представлені у вигляді незалежних сервісів, які можуть обмінюватися даними один з одним. Це схоже на мікросервісну архітектуру, але орієнтоване на більші сервіси (Додаток Д).

Переваги:

- Гнучкість у використанні різних технологій і мов програмування для різних сервісів.
- Можливість перевиконання сервісів, що зменшує дублювання логіки.

Недоліки:

- Складна інтеграція та налаштування комунікацій між сервісами.
- Важче забезпечувати узгодженість даних між різними сервісами.
- Основні відмінності архітектур ПЗ:

Моноліт — один великий блок коду, мікросервіси — набір автономних частин, що легше масштабувати. Клієнт-сервер чітко розділяє інтерфейс (клієнт) і логіку (сервер). SOA орієнтована на великі системи, мікросервіси —

на менші незалежні сервіси. Архітектура обробки подій фокусується на реакції на події, а не на постійному обміні даними.

Для поточного проєкту найкраще підходить клієнт-серверна архітектура. Вона забезпечує чіткий поділ: фронтенд (React) відповідає за інтерфейс, бекенд (Node.js) — за обробку, зберігання і безпеку (авторизація, контроль доступу). Сервер обслуговує одночасно різні клієнти (веб, мобільні, API), що підвищує масштабованість і гнучкість.

Отже, обираємо клієнт-серверну модель. Структура проєкту: папка atc-generator, у якій — фронтенд atc-admin (client) і бекенд server.

1.3. Моделювання процесів

З метою кращого розуміння структури та логіки функціонування автоматизованої системи управління курсами підвищення кваліфікації було здійснено моделювання ключових бізнес-процесів за допомогою діаграм. Це дозволяє візуалізувати послідовність дій, структуру бази даних, а також зв’язки між сутностями, що взаємодіють у межах системи.

Для моделювання структури даних використано ER-діаграму, створену за допомогою інструменту dbdiagram.io. ER-діаграма дозволяє побачити структуру таблиць, типи полів та логічні зв’язки між об’єктами бази даних (Додаток Ж).

Ключові таблиці системи:

users

Зберігає облікові записи адміністраторів, що мають доступ до адмін-панелі системи. Основні поля: id, username, password, email.

generaldata

Містить файли учасників та шаблони сертифікатів, завантажені адміністратором. Основні поля: id, participants_filename, participants_filedata, template_filename, template_filedata.

programs

Визначає перелік програм підвищення кваліфікації та відповідні результати навчання. Основні поля: id, program_name, results.

certificates

Зберігає згенеровані сертифікати разом із даними для завантаження. Основні поля: id, generaldata_id, certificate_filename, certificate_filedata. Визначимо зв’язки між таблицями.

Таблиця certificates має зовнішній ключ generaldata_id, який посилається на id таблиці generaldata, забезпечуючи логічний зв’язок між згенерованими сертифікатами та відповідним вхідним набором даних.

Таблиці users та programs наразі не мають зовнішніх ключів, проте використовуються у відповідних модулях системи (авторизація, заповнення сертифікатів).

Таким чином, структура бази даних демонструє продуману та масштабовану архітектуру, яка забезпечує чітке відображення логіки системи та готує ґрунт для подальшого розширення функціональності.

РОЗДІЛ 2. АНАЛІЗ ТА ВИБІР ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ

2.1. Огляд сучасних технологій для веб-розробки. Розподіл ролей у веб-застосунку

Сучасна веброзробка швидко розвивається і включає безліч інструментів та методологій. Вибір технологій впливає на ефективність, якість, підтримку та масштабованість продукту. Важливо також забезпечувати кібербезпеку, адже 2024 рік був одним із найскладніших у цій сфері, згідно думки експертів, і це залишається пріоритетом.

Для захисту використовують шифрування, фаєрволи, двофакторну аутентифікацію, регулярні оновлення та бекапи. Перехід на сучасні протоколи, як HTTPS, підвищує безпеку і покращує рейтинг у пошуковиках.

Вебзастосунки мають три основні компоненти: фронтенд, бекенд і базу даних, які взаємодіють для роботи системи.

Frontend – це те, з чим безпосередньо взаємодіє користувач. Це інтерфейс, який відображається в браузері, і всі елементи, які реагують на дії користувача: кнопки, таблиці, форми, повідомлення тощо.

Основні функції frontend:

- Відображення даних, отриманих із сервера через API.
- Реагування на дії користувача (події кліків, введення, перемикання тощо).
- Перевірка введених даних (валидація на стороні клієнта).
- Забезпечення навігації (routing).
- Інтерактивність та UX-анімації.

Для реалізації frontend у сучасних застосунках використовуються фреймворки та бібліотеки, такі як: React, Vue, Angular, Svelte тощо.

Backend – це "невидима" частина системи, яка виконує логіку, обробляє запити, звертається до бази даних і відповідає на клієнтські запити.

Основні завдання backend:

- Обробка HTTP-запитів (GET, POST, PUT, DELETE)
- Валідація та обробка даних
- Авторизація та автентифікація користувачів
- Інтеграція з зовнішніми сервісами (email, SMS)

Розглянемо роботу з базами даних.

Найчастіше бекенд реалізують з використанням таких мов і платформ: Node.js, Python, PHP.

База даних – це центр зберігання всіх структурованих даних системи. Серед типів баз даних виділяють реляційні(SQL) та нереляційні(NoSQL). Реляційна база даних (SQL) – база, у котрій дані зберігаються у форматі таблиць, вони суворо структуровані та пов’язані одна з одною. У таблиці є рядки (rows) та стовпчики (columns), кожен є записом, а стовпчик – поле з

призначеним йому типом даних. У кожній комірці інформація записана згідно шаблону.

Нереляційна база даних (NoSQL) – зберігає дані без чітких зв’язків між собою та без чіткої структури. Замість структурованих таблиць, всередині бази знаходиться безліч різнорідних документів, в тому числі і зображення, відео та навіть публікації у соціальних мережах. На відміну від реляційних БД, NoSQL бази не підтримують SQL запити.

2.2. Фронтенд-технології

У сучасній веброзробці клієнтська частина застосунків (frontend) відіграє критичну роль у забезпеченні зручної та інтуїтивно зрозумілої взаємодії з користувачем. Саме через інтерфейс користувачі здійснюють усі дії – від авторизації до управління даними. Тому для побудови адаптивного, швидкого й масштабованого UI використовуються спеціалізовані фреймворки та бібліотеки.

Серед безлічі інструментів для створення інтерфейсів найпопулярніші представимо в таблиці 2.1.

Таблиця 2.1

Порівняльна таблиця фронтенд фреймворків

Технологія	Опис і переваги	Недоліки	Коли використовувати
React	Бібліотека від Meta (Facebook). Декларативний підхід, компоненти, великий екосистема, підтримка TypeScript	Не є повноцінним фреймворком — потребує додаткових інструментів (роутер, менеджер стану)	Великі та середні SPA, панелі керування, інтерактивні застосунки

Vue.js	Простий у навчанні, зручний синтаксис, двостороннє зв’язування, підходить для MVP	Менша кількість enterprise-проєктів, слабша типізація	Швидка розробка, невеликі комерційні сайти
Angular	Потужний фреймворк від Google. Включає все: роутинг, DI, валідацію, RxJS	Високий поріг входу, складний API	Enterprise-рішення, великі масштабовані застосунки

Обрали React через його простоту та зрозумілість у порівнянні з Vue чи Angular. React легко вивчити, що пришвидшує розробку. Для збірки раніше використовували Webpack або Create React App, але з 2022 року популярність набирає Vite — сучасний інструмент для збірки JS/TS проєктів.

Таблиця 2.2

Порівняльна таблиця інструментів для збірки веб-застосунків

Інструмент	Швидкість dev-запуску	Підтримка сучасного синтаксису	Простота налаштування
CRA	Середня	Обмежена	Висока
Webpack	Потужна, але складна	Повна	Потрібна конфігурація

Vite	Миттєва (HMR)	ESM, JSX, TS, PostCSS	Мінімальна

Обрали Vite через його швидкість, простоту та сучасність — він навіть рекомендований для професійних UI-бібліотек.

Стилізація важлива, бо впливає на перше враження, зручність і доступність інтерфейсу. Для цього проєкту використовуватиметься Tailwind CSS — утилітарний фреймворк для швидкого створення адаптивних інтерфейсів без кастомного CSS — та Ant Design (AntD).

Переваги Tailwind CSS:

- Висока швидкість розробки за рахунок використання утилітарних класів.
- Повна кастомізація дизайну без необхідності перевизначати стилі.
- Адаптивність завдяки мобільно-орієнтованим класам (md:, lg: тощо).
- Добре інтегрується з сучасними фреймворками (React, Vue, Angular).

Ant Design — це потужна UI-бібліотека компонентів, що надає готові інтерактивні елементи (таблиці, форми, модальні вікна тощо), стилізовані відповідно до єдиного дизайну.

Переваги AntD:

- Великий набір готових компонентів (таблиці, форми, алерти, модалки, меню).
- Повна інтеграція з React.
- Підтримка темізації та локалізації.
- Вбудована валідація форм.

Таким чином, поєднання Tailwind CSS для гнучкого макетування та Ant Design для функціональних UI-компонентів забезпечить швидку,

масштабовану і візуально узгоджену реалізацію інтерфейсу адміністрування в системі.

2.3. Бекенд-технології та бази даних

Бекенд є ключовою складовою вебсайту, оскільки тут реалізуються CRUD-операції, складна логіка, безпека та шифрування даних. У нашому випадку вся логіка вибірки даних і генерації сертифікатів відбуватиметься саме на бекенді, тому важливо ретельно обрати технології для його реалізації.

За статистикою Touchlane 2024 року найпопулярнішими серверними технологіями є Django, Express.js, NestJS, Laravel і Ruby on Rails. Для аналізу було обрано Express.js, NestJS і Django — найбільш поширені фреймворки в екосистемах JavaScript/TypeScript та Python, які мають активні спільноти, добру документацію та широкий набір бібліотек. Вони підтримують сучасні архітектурні підходи, такі як RESTful API, middleware та dependency injection, і широко застосовуються у комерційних проєктах.

Таким чином, для технічного рішення було обрано ці три фреймворки, які відповідають сучасним вимогам, сприяють масштабованості, зручності супроводу та безпеці системи (див. таблиця 2.3).

Таблиця 2.3

Порівняльна таблиця бекенд фреймворків

Технологія	Мова	Особливості	Сфера застосування
Express.js	JavaScript / TypeScript	Мінімалістичний, найпопулярніший серед Node.js фреймворків	REST API, мікросервіси

NestJS	TypeScript	Архітектура схожа на Angular, DI, модулі	Великі масштабовані сервіси
Django	Python	Повноцінний фреймворк, ORM, авторизація	Web-застосунки, MVP

Express.js обрали для серверної частини через низку переваг. Він ідеальний для створення REST API завдяки простій структурі маршрутів, що полегшує взаємодію з клієнтом. Має низький поріг входу — мінімалістичний синтаксис, широка документація й багато прикладів, що важливо при обмежених термінах. Повністю сумісний із React/Vite, що спрощує інтеграцію та забезпечує узгодженість форматів даних (JSON). Гнучкий, дозволяє реалізувати будь-яку бізнес-логіку без жорстких обмежень.

Для обробки файлів (Excel, Word, сертифікати) підтримує корисні пакети, як multer, fs, path. Розширювана екосистема middleware та бібліотек безпеки (helmet, cors, bcrypt, jsonwebtoken) допомагає масштабувати й захищати застосунок.

Отже, Express.js забезпечує швидкий старт, зручну інтеграцію та гнучкість у реалізації всіх необхідних функцій — від роботи з файлами до авторизації через JWT.

Розглянемо базу даних. Саме в базі даних відбувається збереження усієї інформації, яку ми створюємо, отримуємо, змінюємо, видаляємо.

Важливо підібрати правильний вид бази даних: ми повинні уже мати певне уявлення щодо вигляду збереження даних, їх зв’язків (якщо такі потрібні) тощо. Тобто звернемося до другого пункту в першому розділі, де було відображено усі процеси в діаграмах.

Надамо порівняльну таблицю реляційних та нереляційних баз даних (таблиця 2.4):

Таблиця 2.4

Порівняльна таблиця реляційних та нереляційних баз даних

Критерій порівняння	Реляційні бази даних (SQL)	Нереляційні бази даних (NoSQL)
Структура даних	Таблиці з чіткою схемою (схема строго фіксована)	Документи, ключ-значення, графи, колонки (схема гнучка)
Мова запитів	SQL (Structured Query Language)	Залежить від типу (MongoDB – BSON/JSON запити, інші – API)
Типи даних	Табличні ряди з визначеними типами	JSON-подібні документи або інші форми зберігання
Гнучкість	Менш гнучка — зміна структури потребує міграцій	Висока — можливе збереження даних без уніфікованої структури
Масштабованість	Вертикальна (збільшення ресурсів одного сервера)	Горизонтальна (додавання нових серверів у кластер)
Цілісність даних	Висока — підтримка зв'язків через foreign keys	Низька — зв’язки реалізуються на рівні застосунку
Приклад використання	Фінансові системи, CRM, ERP, аналітика	Реального часу (чат, IoT), Big Data, динамічні вебсервіси
Приклади СУБД	MySQL, PostgreSQL, Oracle, MS SQL Server	MongoDB, CouchDB, Cassandra, Firebase, Redis
Сценарії з багатьма зв’язками	Підходить найкраще	Потрібна додаткова логіка
Продуктивність	Вища при складних запитах, транзакціях	Вища при швидкому читанні/записі, високій навантаженості

Проаналізувавши відмінності та переглянувши наші діаграми та вимоги було прийнято рішення використовувати реляційну базу даних MySQL. Такий вибір був обґрунтований рядом технічних та організаційних переваг цієї СУБД: чітка та стабільна структура даних, надійність і стабільність, легкість інтеграції Express.js, підтримка складних SQL-запитів.

Враховуючи потребу в надійній, структурованій і масштабованій системі управління даними — вибір MySQL є логічним, безпечним та ефективним для даного проєкту.

2.4. Автентифікація та безпека

Автентифікація – це процес підтвердження того, що лише користувачі, служби й програми з належними дозволами можуть отримувати доступ до корпоративних ресурсів. Вона є важлива, оскільки допомагає організаціям захищати системи, дані, мережі, веб-сайти й програми від атак. Крім того, вона допомагає окремим користувачам забезпечувати конфіденційність персональних даних і вести бізнес, зокрема у сфері банкінгу або інвестицій, онлайн із меншим ризиком. Якщо автентифікація слабка, зловмисникам простіше зламувати облікові записи. Вони або вгадують паролі користувачів, або обманним шляхом змушують жертв передати облікові дані. Це може призвести до таких ризиків:

- порушення безпеки даних або їх ексфільтрації;
- інсталяції шкідливого програмного забезпечення, як-от зловмисних програм із вимогою викупу;
- невідповідності регіональним або галузевим нормам щодо конфіденційності даних.

Сучасні способи аутентифікації передбачають делегування відповідних процесів довіреній окремій системі ідентифікації (на відміну від традиційних, під час яких кожна система підтверджує ідентичності самостійно). Крім того, змінилися типи використовуваних способів аутентифікації. Щоб отримати

доступ до більшості програм, потрібно ввести ім’я користувача й пароль. Однак оскільки зловмисники вдосконалили свої методи викрадення паролів, спільнота фахівців із кібербезпеки розробила кілька нових способів захисту ідентичностей:

Автентифікація за паролем — найпоширеніший метод, проте має суттєві недоліки. Через складність створення та запам’ятовування унікальних паролів користувачі часто їх повторюють, що підвищує ризики. Зловмисники застосовують різні методи для викрадення паролів, тому все більше організацій переходять до безпечніших способів автентифікації.

Сертифікатна автентифікація — це зашифрований спосіб ідентифікації користувачів або пристроїв у системі. Найчастіше використовується через смарт-картки або надсилання цифрових сертифікатів на сервер.

Біометрична автентифікація підтверджує особу користувача за фізичними ознаками, як-от відбитки пальців, обличчя або сітківка. Вона зручна (не потребує запам’ятовування) і безпечніша за паролі, оскільки дані прив’язані до пристрою і важкодоступні для зловмисників.

Автентифікація за маркером (TOTP) — це метод, коли пристрій і система кожні 30 секунд генерують однаковий одноразовий код. Якщо код збігається, користувач підтверджується.

Одноразовий пароль — це код, дійсний лише протягом одного сеансу автентифікації. Його надсилають через SMS, email або апаратний ключ.

Push-автентифікація — це метод, коли користувач отримує сповіщення на телефон із запитом дозволити або відхилити вхід. Щоб уникнути помилкових підтверджень, її часто поєднують з одноразовими кодами для підвищення захисту від фішингу.

Голосова автентифікація передбачає, що користувач відповідає на телефонний дзвінок і вводить код або ідентифікує себе голосом.

Багатофакторна автентифікація зменшує ризик компрометації облікових записів, вимагаючи два або більше способів підтвердження особи. Ефективний захист передбачає подання користувачем будь-яких двох із таких типів даних:

- Щось, що знає лише користувач, як-от пароль.
- Щось, що є лише в користувача, зокрема надійний пристрій, який важко дублювати (наприклад, телефон або апаратний ключ).
- Щось, що належить лише користувачу, зокрема відбиток пальця або обличчя.

Двофакторна аутентифікація – це тип багатофакторної автентифікації, який полягає у використанні двох форм підтвердження ідентичності.

Для зменшення ризиків, у проєкті буде реалізовано низку інструментів та практик, які забезпечуть багаторівневу систему безпеки:

JWT (JSON Web Token) — для перевірки сесій користувачів. JWT — це сучасний, легкий спосіб передачі зашифрованої інформації між клієнтом і сервером. Як це працює:

Після успішного входу сервер генерує токен, який містить зашифрований payload (ID користувача, час сесії тощо).

Цей токен зберігається на стороні клієнта (зазвичай у localStorage) і передається у заголовках запиту (Authorization: Bearer ...) до серверу.

Сервер перевіряє токен при кожному запиті — і якщо він валідний, дозволяє виконати дію.

bcrypt — для хешування паролів.

Паролі користувачів не зберігаються у відкритому вигляді. Замість цього використовується bcrypt — криптографічний алгоритм хешування, який:

додає сіль (salt) для кожного пароля (унікальний випадковий рядок);

застосовує багаторазове хешування (наприклад, 10–12 раундів), що ускладнює brute-force атаки.

CORS + Helmet + HTTPS — для захисту API. Усі запити до API захищені через багатоступеневу конфігурацію на сервері:

CORS (Cross-Origin Resource Sharing): обмежує доступ до API лише з довірених доменів (наприклад, localhost:5173 під час розробки).

Helmet: набір middleware для Express.js, який додає заголовки безпеки:

Content-Security-Policy

X-Frame-Options

X-XSS-Protection

що дозволяє захиститися від XSS, clickjacking та інших атак.

HTTPS: шифрує всі запити між клієнтом і сервером, забезпечуючи цілісність даних.

Використання HTTPS — стандарт у сучасній веб-розробці, обов'язковий для будь-яких систем, які працюють із користувачами.

Обраний набір інструментів забезпечує надійний захист облікових записів, REST API та переданих даних, що особливо важливо для державних і освітніх систем із підвищеними вимогами до безпеки персональної інформації.

2.5. Деплой серверної частини та фронтенд частини проєкту з допомогою відповідних сервісів

Розгортання (деплой) веб застосунку — це процес перенесення проєкту з локального середовища розробки у публічне середовище, де ним можуть користуватись реальні користувачі. Для цього використовуються різноманітні платформи, які надають інфраструктуру для хостингу серверної (бекенд) та клієнтської (фронтенд) частини застосунку.

У табл. 2.5 наведено порівняння найпопулярніших сервісів для деплою застосунків.

Таблиця 2.5

Порівняння сервісів для деплою застосунків

Сервіс	Типи проєктів	Переваги	Недоліки
Render	Fullstack (Node.js, Python, React, PostgreSQL, тощо)	Простота налаштування Підтримка вебсокетів, бекенду та БД CI/CD з GitHub	Деякі обмеження на безкоштовному тарифі

Vercel	Фронтенд (React, Next.js, Vue)	Ідеальний для SSR Автоматичний деплой з Git Дуже швидкий CDN	Обмежена підтримка бекенду
Netlify	Фронтенд (React, Vue, Angular, статичні сайти)	Простота інтеграції з Git Функції “serverless”	Менш придатний для повноцінного бекенду

Як результат - наш вибір очевидно впаде на Render оскільки потрібно задеплоїти Fullstack проєкт.

Основні переваги такого рішення:

Єдиний хостинг для клієнта і сервера: можна окремо налаштувати frontend та backend сервіси, які будуть автоматично оновлюватися при пушах на GitHub.

Підтримка середовищ виконання: Node.js, Python, PostgreSQL, Redis, Docker-контейнери.

Секрети та змінні середовища (ENV): легко конфігуруються у веб-інтерфейсі Render.

Автоматичне масштабування (на платних тарифах).

HTTPS з коробки.

Пізніше для повноцінної роботи проєкту необхідно буде додати відповідні змінні середовища (.env) у панелі Render, а також врахувати, що доступ до MySQL-бази з localhost (127.0.0.1) в середовищі Render призводить до помилки — для цього слід використовувати зовнішній хостинг бази даних або Render PostgreSQL-інстанс. В даному випадку буде використовуватися зовнішній хостинг бази даних — Clever Cloud. Це надійна платформа, яка дозволяє створити власну MySQL-інстанцію з хмарним розміщенням, налаштованими змінними середовища та високою доступністю.

РОЗДІЛ 3. РОЗРОБКА ТА РЕАЛІЗАЦІЯ СИСТЕМИ. ТЕСТУВАННЯ ТА НАЛАГОДЖЕННЯ РОБОТИ. НАПИСАННЯ ДОКУМЕНТАЦІЇ

У даному розділі описується практичне впровадження автоматизованої системи управління курсами підвищення кваліфікації педагогічних працівників. Основна увага приділяється вибору інструментів розробки, реалізації серверної та клієнтської частин, створенню REST API та механізму генерації сертифікатів.

3.1. Початок розробки із вибраними технологіями. Розгортання проєкту

Розглянемо початок розробки із вибраними технологіями та розгортання проєкту **(**React, Vite, Node.js, MySQL, Render); побудову та інтеграцію клієнтської та серверної частин у межах клієнт-серверної архітектури.

Розробка автоматизованої системи управління курсами підвищення кваліфікації базується на клієнт-серверній архітектурі, що забезпечує розподіл відповідальностей між фронтендом і бекендом. Такий підхід сприяє масштабованості, повторному використанню коду, безпеці та гнучкості в обслуговуванні системи.

Фронтенд реалізовано з використанням Create React App — зручного інструменту для створення односторінкових застосунків на React. Він дозволяє використовувати новітні можливості JavaScript, спрощує розробку та оптимізує застосунок для продакшну. Потрібна версія Node ≥ 8.10, npm ≥ 5.6.

Проєкт зберігається у репозиторії atc-generator, який містить як клієнтську, так і серверну частину. Наразі фокус — на клієнтській.

Create React App не відповідає за логіку бекенду чи бази даних, тому може бути інтегрований із будь-яким сервером. Він працює на основі Babel і Webpack.

Для управління залежностями використовується npm — дефолтний менеджер пакетів для Node.js, що дає змогу легко встановлювати та оновлювати бібліотеки.

Ініціалізація клієнтської частини та встановлення залежностей (див. Додаток И).

Налаштування TailwindCSS згідно документації для використання утилітарних класів Tailwind без необхідності ручного написання CSS.

Налаштування головного клієнтського файлу можна переглянути у додатку Л.

Фронтенд частину проєкту розгорнуто із відповідними та потрібними нам залежностями, та сформовано структуру клієнтської частини проєкту (див. Додаток К).

Бекенд

Згідно з вимогами (табл. 1.2), сервер забезпечує:

REST API для ресурсів: users, programs, generaldata, certificates

Авторизацію через JWT

Контроль доступу через middleware

Роботу з файлами (Excel, Word): зчитування, обробка, збереження, архівація

У репозиторії atc-generator, перехід у папку server дозволяє ініціалізувати бекенд (див. Додаток М).

Архітектура

Сервер реалізовано на Node.js + Express. Логіка розділена на контролери, маршрути та сервіси, що забезпечує чисту та масштабовану структуру (див. Додаток П). Головний файл — app.js (див. Додаток Н).

Після налаштування фронтенду та бекенду отримано стабільну архітектуру, яка дозволяє автоматизувати генерацію сертифікатів і керування програмами підвищення кваліфікації.

Інтеграція клієнтської та серверної частини системи

Одним з ключових етапів реалізації автоматизованої системи є забезпечення коректної інтеграції між фронтендом (клієнтською частиною) та бекендом (серверною частиною). Для цього у проєкті було реалізовано взаємодію через REST API, що відповідає сучасним стандартам веб-розробки.

Архітектура інтеграції

Фронтенд-застосунок, реалізований за допомогою React, взаємодіє з сервером, побудованим на Node.js + Express, за допомогою HTTP-запитів, що надсилаються через бібліотеку Axios. Усі ключові дії (авторизація, завантаження даних, генерація сертифікатів, CRUD-операції) виконуються через API, що дозволяє чітко розмежувати обов’язки між частинами системи.

Авторизація та обробка токенів

Щоб забезпечити безпеку доступу до захищених ресурсів (наприклад, перегляд та редагування користувачів, програм тощо), використовується JWT (JSON Web Token). Після успішної авторизації користувач отримує токен, який зберігається у localStorage і надалі автоматично додається до кожного запиту до сервера (див. Додаток Р). Обробки помилок авторизації: у разі, якщо токен є простроченим або недійсним, сервер повертає код 401 Unauthorized, після чого користувача автоматично перенаправляє на сторінку входу (див. Додаток С). Такий механізм забезпечує:

захист адмін-панелі від неавторизованого доступу;

автоматичну обробку завершення сесії;

оновлення або видалення токена в localStorage.

3.2. Розробка серверної частини та механізм генерації сертифікатів

Серверна частина реалізована на Node.js із використанням express, mysql2, xlsx, docx, docxtemplater, jsonwebtoken тощо. Створено REST API для обробки даних, авторизації, роботи з файлами Excel/Word і генерації сертифікатів.

Основні етапи роботи:

Дані учасників зчитуються з Excel-файлу з бази (generaldata), розбираються через xlsx та формуються у масив об’єктів.

Інформація про програми витягується з бази та зв’язується з учасниками.

Сертифікати створюються на основі шаблону DOCX з використанням PizZip та docxtemplater — маркери шаблону замінюються на реальні дані ({{name}}, {{program}}, {{grade}}).

Згенеровані документи зберігаються на сервері й у базі як двійкові файли.

API повертає масив об’єктів з інформацією про створені сертифікати.

Додаткові можливості:

Авторизація через JWT: після логіну видається токен доступу для захищених маршрутів.

Генерація ZIP-архіву з усіма сертифікатами для заданого generaldata_id — без тимчасового збереження файлів на диск.

Таким чином, серверна частина забезпечує повну автоматизацію генерації сертифікатів: від завантаження Excel-файлів до збереження й архівації готових документів. API побудовані з урахуванням безпеки, масштабованості та інтеграції з адміністративним інтерфейсом.

3.3. Розробка клієнтської частини (інтерфейс адміністратора)

Клієнтська частина адміністративної панелі побудована на React з React Router для маршрутизації, контекстом автентифікації (AuthProvider) і бібліотекою Ant Design для UI. Використовується Axios з інтерцепторами для захищених запитів. Основний компонент Dashboard містить кілька сторінок: керування користувачами (/dashboard/users), генерація сертифікатів (/dashboard/generation) та управління програмами підвищення кваліфікації (/dashboard/programs).

Сторінка користувачів дозволяє додавати, редагувати та змінювати права адміністраторів. На сторінці генерації сертифікатів адміністратор завантажує Excel із учасниками та шаблон DOCX, після чого запускає процес

генерації. Якщо сертифікати вже є, доступне завантаження ZIP-архіву або повторний запуск генерації. Для завантаження файлів використано компонент Upload з Ant Design, а запити відправляються через Axios.

Сторінка програм дозволяє адміністратору додавати навчальні програми з описом, датами і годинами, які потім підставляються у сертифікати. Axios інтерцептори додають токен з localStorage у кожен запит; у випадку 401 токен видаляється, а користувача перенаправляють на сторінку логіну.

Навігація реалізована через компоненти Routes і Route. Інтерфейс створено з орієнтацією на зручність, безпеку та інтеграцію з бекендом для обробки файлів і генерації документів.

Додатково розроблено дві форми для учасників і розробників програм, які дозволяють вводити дані без доступу до адмін-панелі. Це забезпечує розмежування доступу та адаптацію інтерфейсів під різні ролі. Хоча бекенд підтримує збереження цих даних, наразі функціонал не активний у продакшні і використовується локально.

Такий підхід із кількома клієнтськими частинами підвищує безпеку, зручність і гнучкість системи, а також дозволяє в майбутньому масштабувати або розділяти функції на окремі сервіси.

3.4. Тестування та налагодження системи

Розглянемо розгортання клієнтської та серверної частин на хмарному хостингу Render; документацію розробки системи

Для забезпечення постійного доступу до автоматизованої системи управління курсами підвищення кваліфікації педагогів було обрано хмарну платформу Render. Render — це сучасна DevOps-платформа з підтримкою безперервного розгортання (Continuous Deployment) вебзастосунків, інтеграцією з GitHub/GitLab, автоматичним встановленням залежностей, налаштуванням змінних середовища та вбудованим SSL.

Перед розгортанням бекенду перевірили конфігурацію запуску в package.json (команда start: "node app.js"). В корені проєкту створено файл .env

з конфіденційними даними (параметри бази даних, SECRET_KEY). У Render створили Web Service, пов’язаний із GitHub-репозиторієм, налаштували середовище Node, команди запуску і встановлення залежностей, порт 3000 (process.env.PORT). Змінні середовища додано вручну через вебінтерфейс для узгодженості з локальним .env.

Для бази даних обрали хмарний сервіс Clever Cloud, оскільки Render не підтримує MySQL напряму. У Clever Cloud створили MySQL-додаток, отримали параметри підключення (хост, порт, користувач, пароль, база даних) та додали їх у .env і Render Environment Variables. Це забезпечує безпечний доступ бекенду до бази без зберігання секретів у коді.

Фронтенд, створений на React та Vite, розгорнуто окремо як Static Site. У фронтенді вказано URL бекенду через змінну VITE_API_URL. У Render налаштовано статичний сайт із репозиторію, команда збірки — npm run build, директорія публікації — dist. Змінна VITE_API_URL дублюється у Render.

Після кожного git push Render автоматично збирає та деплоїть серверну і клієнтську частини, що відповідає принципам CI/CD. У результаті бекенд доступний за https://atc-generator-server.onrender.com/api, фронтенд — за https://atc-generator-client.onrender.com.

Render забезпечує автоматичне розгортання, зручне налаштування змінних, HTTPS за замовчуванням і безкоштовний хостинг для невеликих проєктів. Це робить платформу ефективним інструментом для розгортання вебзастосунків із розділенням клієнтської та серверної частини.

Технічна документація — це сукупність документів, що супроводжують весь життєвий цикл розробки ПЗ. Вона покликана пояснити функціональність продукту, уніфікувати інформацію про проєкт і полегшити взаєморозуміння між розробниками, користувачами та іншими зацікавленими сторонами.

Документація поділяється на дві категорії: документація продукту і документація процесу. Перша описує сам продукт — його архітектуру, вимоги, код, посібники користувача. Друга — фіксує процес розробки: плани, звіти, стандарти.

У цьому проєкті передбачено лише документацію користувача, що допомагає кінцевим користувачам зрозуміти функції системи, уникнути помилок і полегшити інтеграцію. Інструкції з користування наведені в додатку, а архітектурна документація й покрокова інструкція з розгортання — у файлі README.md проєкту.

3.5. Перспективи розвитку та розширення системи

У поточній реалізації системи закладено основу для подальшого розширення функціональності шляхом впровадження окремих клієнтських частин для різних груп користувачів. Зокрема, створено дві форми — для учасників програм підвищення кваліфікації та для розробників програм. Хоча обробка цих форм наразі працює лише локально (згідно з домовленостями з Центром), відповідні API-ендпоінти та таблиці у MySQL вже реалізовані.

У майбутньому планується повноцінне впровадження цього функціоналу у продакшн-системі, що дозволить автоматизувати створення програм: інформація із форми розробника напряму використовуватиметься для генерації описів у Word-файлах на основі шаблону, замість ручного завантаження документів. Аналогічно, дані з форми учасника будуть безпосередньо зберігатися у базу, що усуне потребу у проміжних Excel-файлах для генерації сертифікатів.

На першому етапі нові форми функціонуватимуть паралельно з існуючими механізмами завантаження документів, але згодом можуть їх повністю замінити, забезпечуючи повну автоматизацію процесу введення даних.

Також розглядається впровадження автоматизованої розсилки електронних повідомлень: після генерації сертифікатів чи формування програм учасники отримуватимуть сповіщення про нові або обрані курси, результати навчання та доступність сертифікатів.

Реалізація цих функцій значно зменшить ручну працю, покращить взаємодію з користувачами та забезпечить масштабованість системи без суттєвих змін у її архітектурі.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було досягнуто поставлену мету — створено вебзастосунок для автоматизованої генерації сертифікатів на основі даних про учасників курсів підвищення кваліфікації. У процесі реалізації було виконано повний цикл розробки програмного забезпечення — від аналізу предметної області до тестування й розгортання системи на хмарному сервері.

У першому розділі проведено детальний аналіз предметної області. Виявлено основні недоліки традиційного підходу до управління курсами — надмірну трудомісткість, ризик людських помилок та відсутність єдиної системи обліку. На основі аналізу сформульовано вимоги до системи, які потім були узгодженні та дещо змінені відповідно побажань Центру, а саме звужено розробку системи до автоматизації процесу сертифікації учасників, та обрано архітектурний підхід, що передбачає використання клієнт-серверної моделі.

У другому розділі здійснено огляд сучасних технологій, які можуть бути використані для побудови веб-застосунку. Обґрунтовано вибір таких інструментів, як React для реалізації клієнтської частини, Node.js для серверної логіки, MySQL для зберігання структурованих даних, а також сервісу Render для хостингу обох частин системи. Особливу увагу приділено питанню безпеки — реалізовано механізми автентифікації та захисту API.

У третьому розділі описано процес безпосередньої реалізації системи. Розроблено модулі обробки Excel- і Word-файлів, CRUD-функціонал для керування базою учасників та навчальних програм, а також механізм генерації сертифікатів із використанням шаблонів. Створено адміністративну панель, що забезпечує зручний інтерфейс для керування даними. Проведено тестування основних функцій системи, виправлено виявлені помилки та налаштовано хостинг для безперебійного доступу до сервісу. Описано перспективи розвитку та розширення системи.

Таким чином, у результаті виконаної роботи було створено масштабовану, надійну та зручну в користуванні систему для автоматизації одного з важливих аспектів освітнього процесу — формування сертифікатів. Розроблене рішення може бути легко адаптоване до потреб інших освітніх закладів або організацій, що проводять навчання. Надалі систему можна розширити за рахунок додавання інтеграції з поштовими сервісами для автоматичної розсилки інформації, автоматизації процесу внесення даних про учасників та програми.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

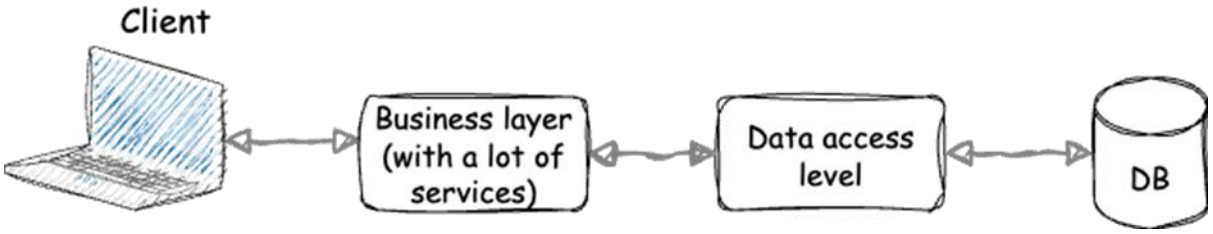
1. Юлія СУРЖЕНКО. Розробка автоматизованої системи управління курсами підвищення кваліфікації педагогічних працівників та керівників закладів освіти. Збірник матеріалів наукової конференції за підсумками науково-дослідної роботи здобувачів вищої освіти фізико-математичного факультету Кам'янець-Подільського національного університету імені Івана Огієнка у 2024-2025 н.р., 9-10 квітня 2025 року [Електронний ресурс]. Кам'янець-Подільський : Кам'янець-Подільський національний університет імені Івана Огієнка, фізико-математичний факультет, 2025. С. 92-94. URL: http://elar.kpnu.edu.ua/xmlui/handle/123456789/8094.
2. Юлія СУРЖЕНКО. Робота з великим набором даних: порівняльний аналіз СКБД та засобів їх обробки. Вісник Кам'янець-Подільського національного університету імені Івана Огієнка. Фізико-математичні науки. Випуск 17. Кам'янець-Подільський : Кам'янець-Подільський національний університет імені Івана Огієнка, 2024. С. 170-173. URL: https://fizmat.kpnu.edu.ua/wp-content/uploads/2024/12/visnyk-17-2024-17-12-2024.pdf
3. Архітектура програмного забезпечення [Електронний ресурс]. URL: https://wezom.com.ua/ua/blog/arhitektura-programmnogo-obespecheniya
4. Основні типи архітектури програмного забезпечення [Електронний ресурс]. URL: https://www.artofba.com/uk/post/main-types-of-software-architecture
5. Most popular backend frameworks in 2024 [Електронний ресурс]. URL: https://touchlane.com/most-popular-backend-frameworks-in-2024/
6. Технічна документація в розробці ПЗ [Електронний ресурс]. URL: https://training.qatestlab.com/blog/technical-articles/types-of-technical-documentation/

7. Fielding, R. T. (2000). Architectural Styles and the Design of Network-based Software Architectures. Doctoral dissertation, University of California, Irvine.
8. Richardson, L., & Ruby, S. (2007). RESTful Web Services. O'Reilly Media.
9. Zakas, N. C. (2012). Maintainable JavaScript: Writing Readable Code. O'Reilly Media.
10. Krol, K. (2018). Designing RESTful Web APIs. O'Reilly Media.
11. St. Laurent, S. (2009). Understanding Open Source and Free Software Licensing. O'Reilly Media.
12. Сидоренко В. В. (2021). Інформаційна безпека веб-додатків: навчальний посібник. Київ: КНУ імені Т. Шевченка.
13. Dhanjani, N., Rios, B., & Hardin, B. (2009). Hacking: The Next Generation. O'Reilly Media.
14. Poston, R. (2022). API Security in Action. Manning Publications.
15. Newman, S. (2015). Building Microservices: Designing Fine-Grained Systems. O'Reilly Media.
16. Що таке автентифікація [Електронний ресурс]. URL: https://www.microsoft.com/uk-ua/security/business/security-101/what-is-authentication

ДОДАТКИ

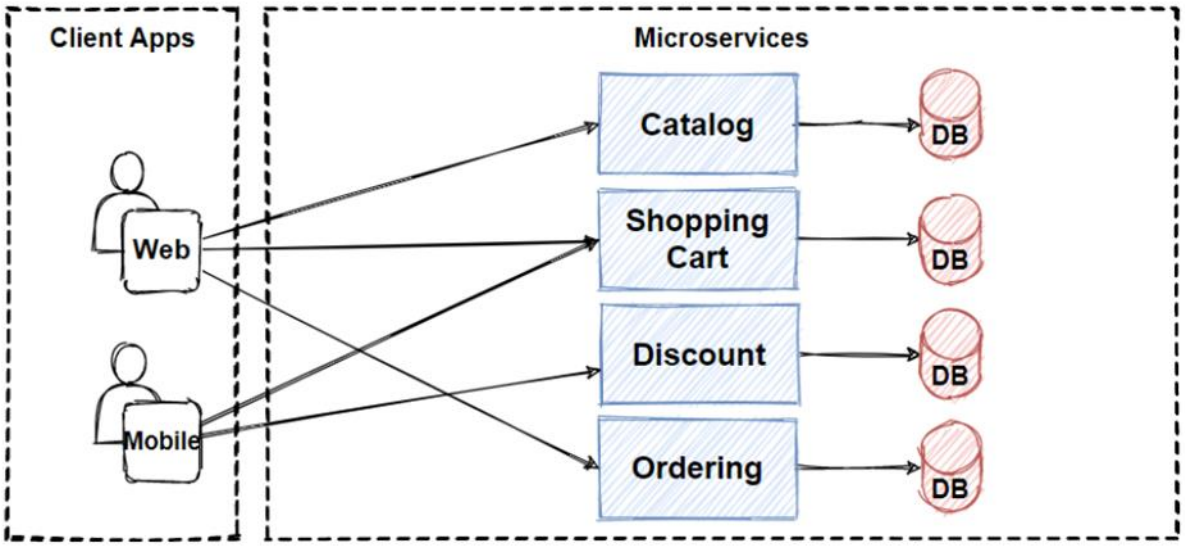
Додаток А

Монолітна архітектура



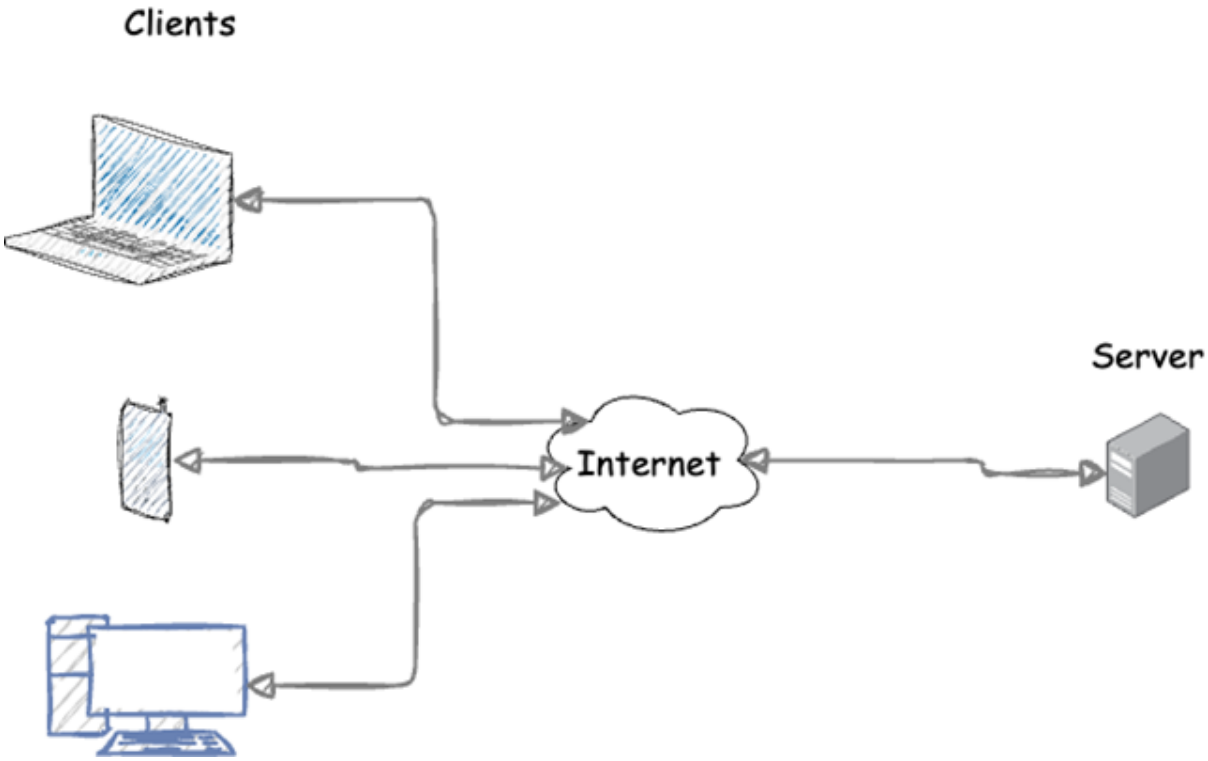
Додаток Б

Мікросервісна архітектура



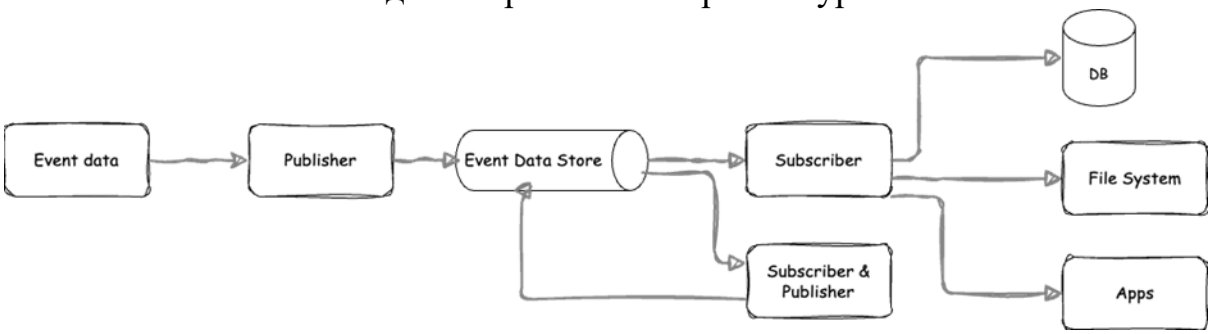
Додаток В

Клієнт-серверна архітектура



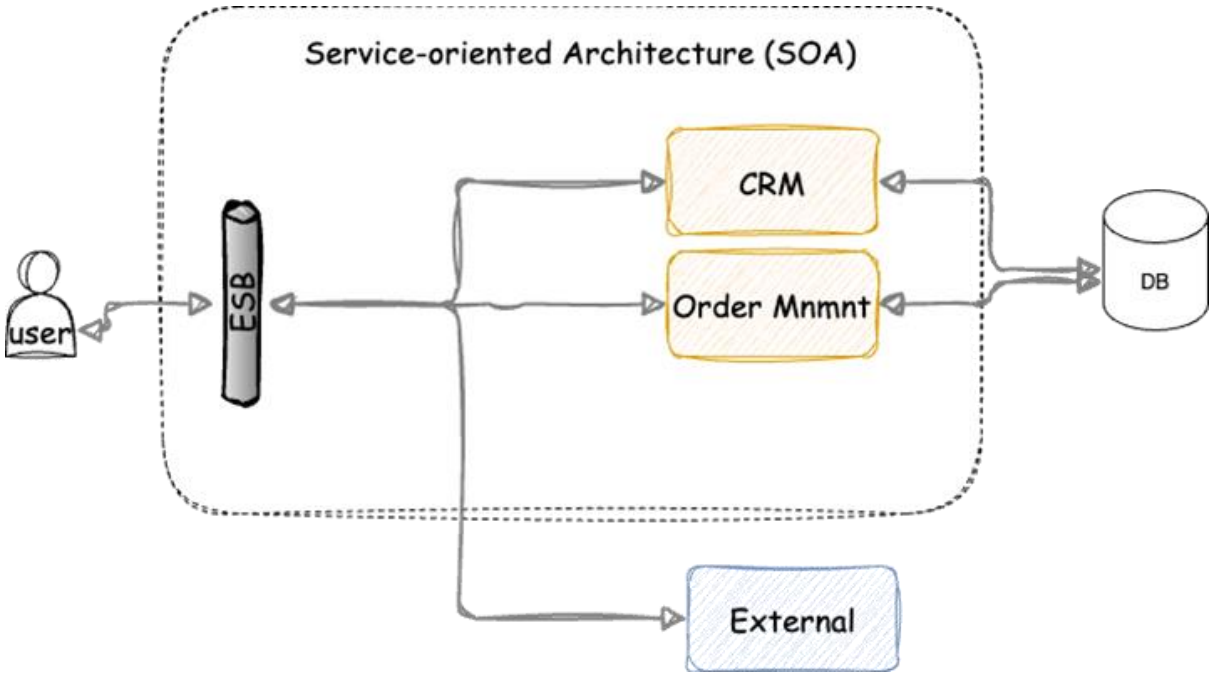
Додаток Г

Подійно-орієнтована архітектура



Додаток Д

Сервіс-орієнтована архітектура



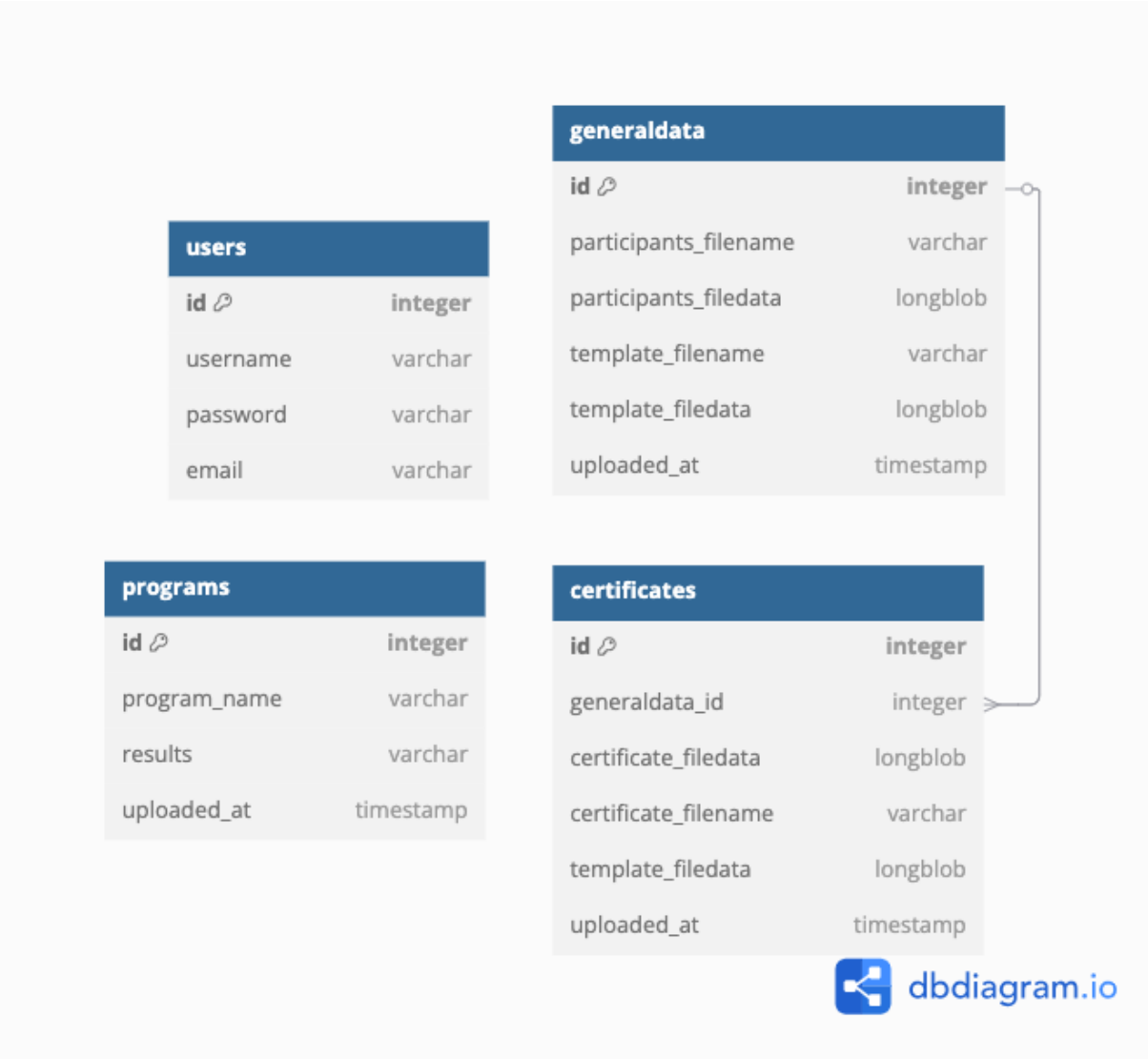
Додаток Е

Вигляд клієнт-серверної арітектури проєкту

atc-generator
- atc-admin
- server

Додаток Ж

ER-діаграма бази даних



Додаток З

Команди ініціалізації фронтенд частини проєкту

```
npm create vite@latest atc-admin --template react-ts
cd atc-admin
npm install
npm run dev

npm install react-router-dom axios zustand tailwindcss
postcss autoprefixer
npx tailwindcss init -p
```

Додаток И. Команди встановлення додаткових залежностей для фронтенд частини.

Додаток К

Структура клієнтської частини проєкту

```
atc-admin/
├── public/                 # Публічні файли (favicon, index.html)
├── src/                    # Основний код проєкту
│   ├── api/                # API-запити (axios)
│   ├── components/         # Компоненти (Sidebar, Button, Inputs)
│   ├── hooks/              # Кастомні хуки (useAuth)
│   ├── layouts/            # Макети сторінок (DashboardLayout)
│   ├── pages/                  #  Сторінки  (Login,  Users,  Programs,
ParticipantsData)
│   ├── routes/            # Налаштування маршрутизації (React Router)
│   ├── store/              # Сховище (Redux/Context API)
│   ├── styles/             # Глобальні стилі (Tailwind CSS)
│   ├── App.tsx             # Головний компонент
│   ├── main.tsx            # Точка входу в додаток
│   └── vite.config.ts      # Налаштування Vite
├── package.json            # Залежності проєкту
├── tsconfig.json           # Налаштування TypeScript
├── tailwind.config.js      # Налаштування Tailwind CSS
├── postcss.config.js       # Налаштування PostCSS
└── .gitignore              # Файл ігнорування Git
```

Додаток Л

Налаштування головного клієнтського файлу (App.tsx)

```
import { BrowserRouter as Router, Route, Routes } from
'react-router-dom';
import { AuthProvider } from './context/AuthContext';
import Login from './pages/Login';
import Dashboard from './pages/Dashboard';
import Users from './pages/Users';
import Programs from './pages/Programs';
import GenerationPage from './pages/GenerationPage';
import axios from 'axios';
import { message } from 'antd';

const App = () => {

  return (
    <AuthProvider>
      <Router>
        <Routes>
          <Route path="/" element={<Dashboard />} />
          <Route path="/login" element={<Login />} />
          <Route path="/dashboard" element={<Dashboard
/>} >
            <Route path="users" element={<Users />} />
            <Route  path="programs"  element={<Programs
/>} />
```

```
            <Route                          path="generation"
element={<GenerationPage />} />
          </Route>
        </Routes>
      </Router>
    </AuthProvider>
  );
};

export default App;
```

Додаток М

Команди ініціалізації бекенд частини проєкту

```
npm init -y
npm install mysql2 xlsx mammoth docx fs path express
multer cors
node app.js
```

Додаток Н

Налаштування головного серверного файлу (app.js)

```
const express = require('express');
const cors = require('cors');
const routes = require('./routes/routes');

const app = express();

app.use(express.json());
app.use(express.urlencoded({ extended: true }));

const corsOptions = {
    origin: 'http://localhost:5173',
    methods: 'GET,HEAD,PUT,PATCH,POST,DELETE',
    credentials: true,
    allowedHeaders: 'Content-Type,Authorization',
  };

app.use(cors(corsOptions));

app.use('/api', routes);

const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
```

```
    console.log(`□ Сервер працює на порту ${PORT}`);
});
```

Додаток П

Структура серверної частини проєкту

```
server/
├── _config.yml   # (опціонально) Конфігураційний файл для сторонніх сервісів (наприклад, деплой на GitHub Pages або Render)
├── app.js       # Головний файл запуску сервера (точка входу Node.js)
├── controllers/      #Логіка обробки запитів (CRUD-операції, бізнес-логіка)
├── models/          # Модулі для роботи з базою даних (підключення, запити)
├── node_modules/    # Встановлені залежності Node.js
├── package-lock.json  # Фіксація версій пакетів для відтворюваного білду
├── package.json     # Основний файл проєкту: назва, версії, скрипти, залежності
├── readme.txt        # (опціонально) Документація або короткий опис серверної частини
├── routes/           # Оголошення маршрутів (API endpoints)
└── services/    # Додаткові сервіси: генерація сертифікатів, читання Excel/Word, обробка файлів
```

Додаток Р

Автоматичне додавання токена

```
axios.interceptors.request.use(
  (config) => {
    const token = localStorage.getItem("token");
    if (token) {
      config.headers.Authorization      =      `Bearer ${token}`;
    }
    return config;
  },
  (error) => Promise.reject(error)
);
```

Додаток С

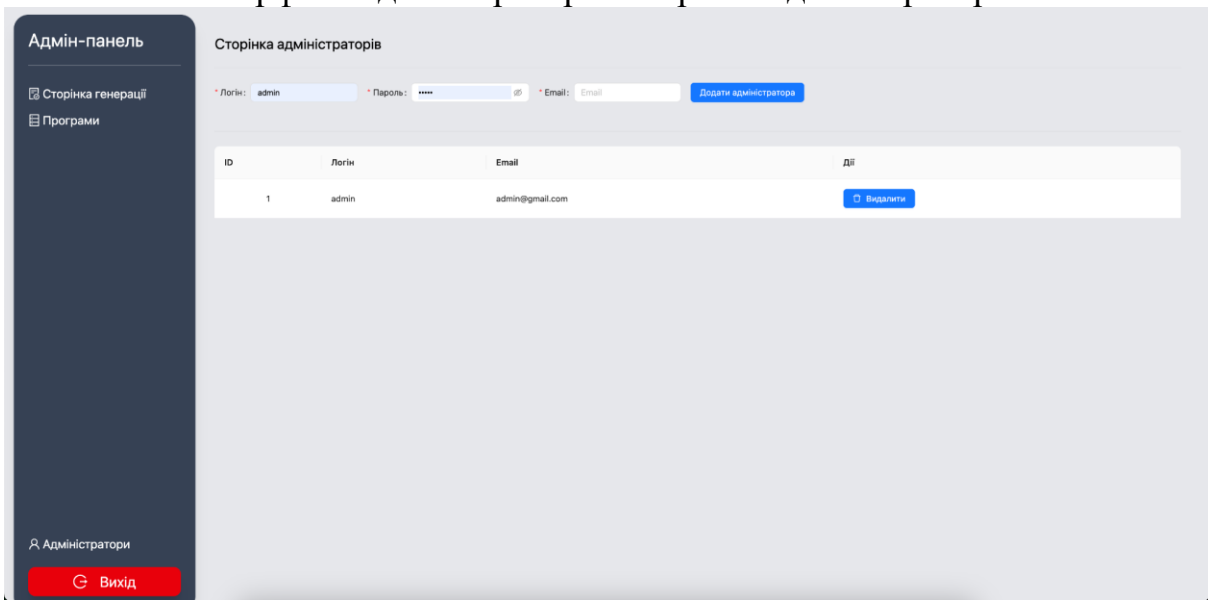
Автоматичне перенаправлення на сторінку входу при 401 помилці

```
axios.interceptors.response.use(
  (response) => response,
  (error) => {
    if (error.response?.status === 401) {
      message.error('Сесія     завершена.     Увійдіть повторно.');
      localStorage.removeItem('token');
```

```
        window.location.href = '/login';
      }
      return Promise.reject(error);
    }
);
```

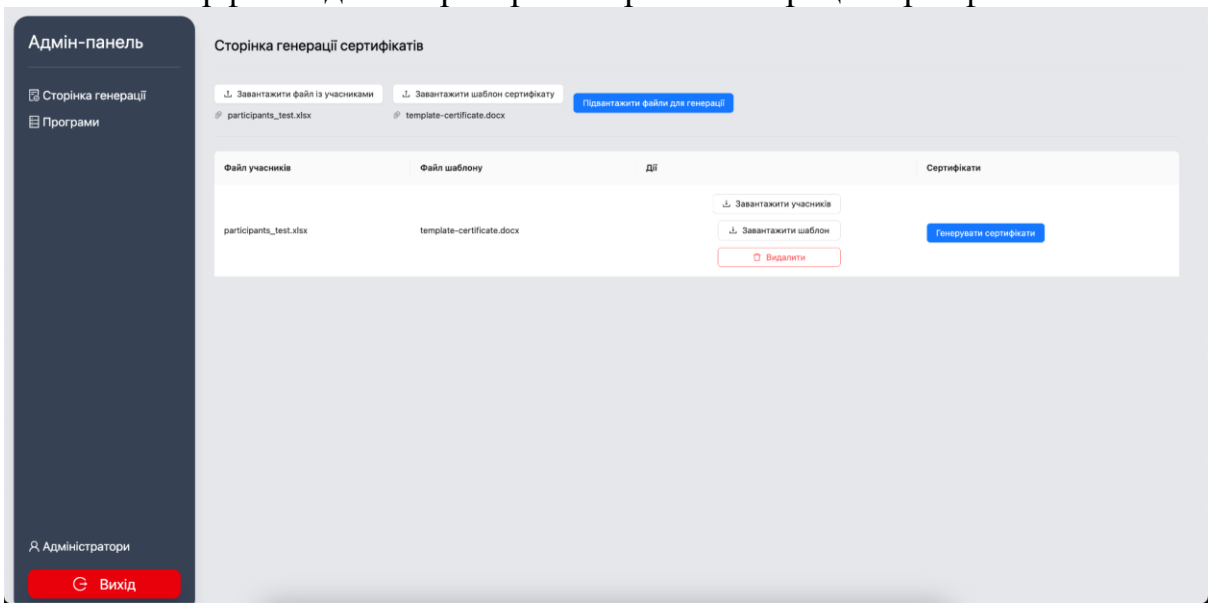
Додаток Т

Інтерфейс адміністратора. Сторінка адміністраторів



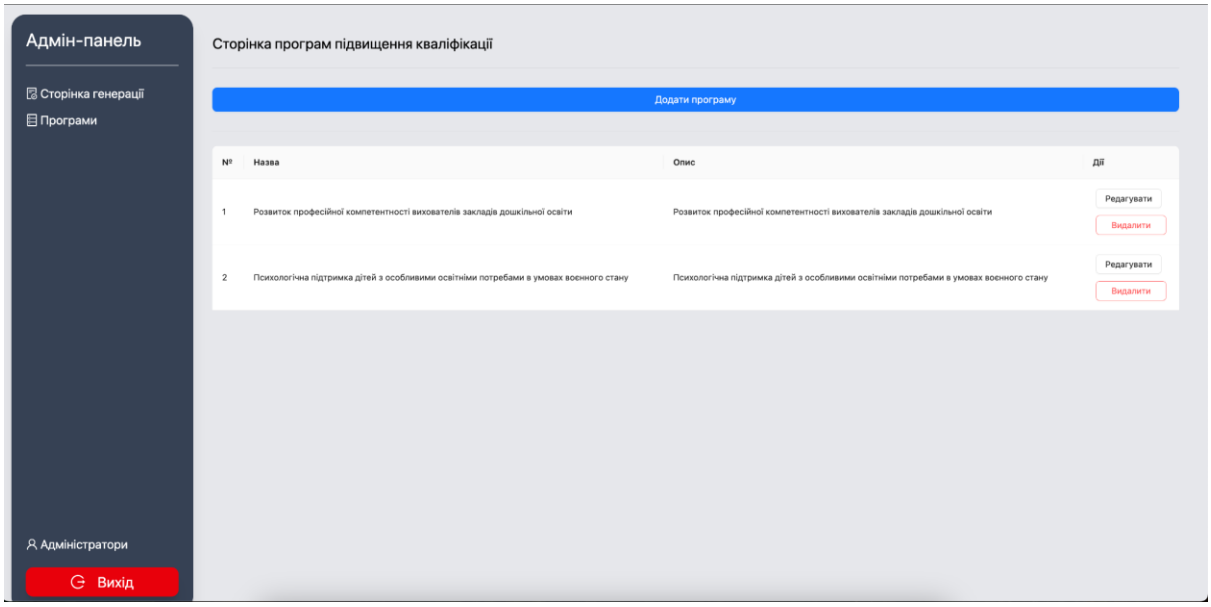
Додаток У

Інтерфейс адміністратора. Сторінка генерації сертифікатів



Додаток Ф

Інтерфейс адміністратора. Сторінка програм підвищення кваліфікації



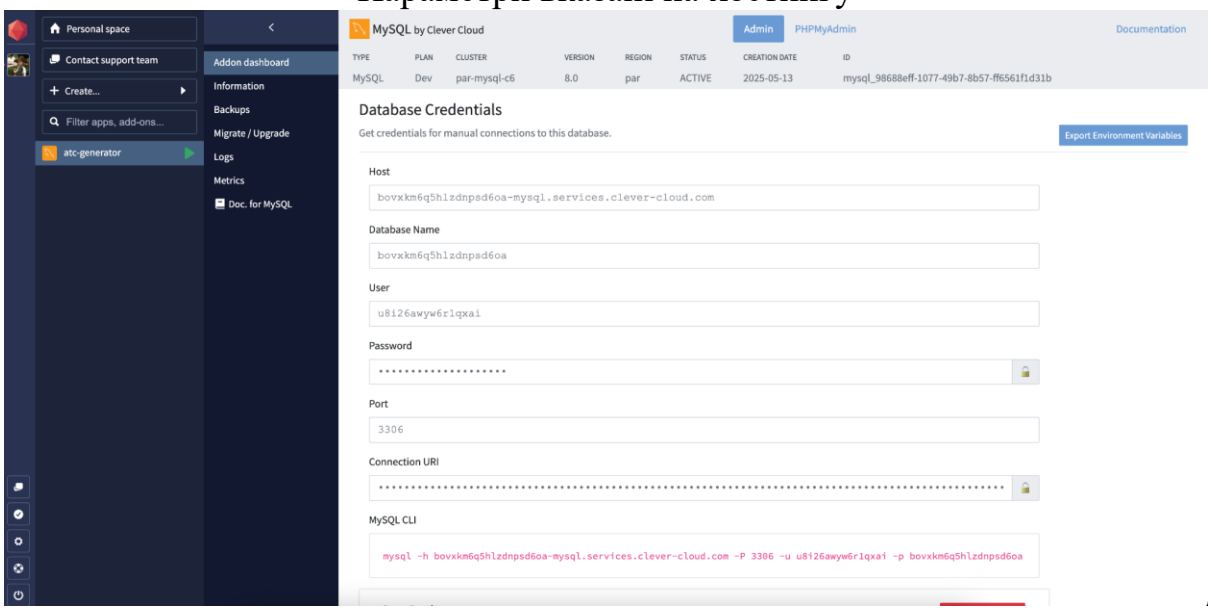
Додаток Х

Архітектура репозиторію після додавання форм

```
atc-admin/
├── atc-admin
├── forms
└── server
```

Додаток Ц

Параметри вказані на хостингу



Додаток Ш

Документація користувача

Для правильної генерації сертифікатів користувач повинен надати файл із даними учасників у форматі Excel. Цей файл має містити строго визначені колонки з назвами:

- Ім'я
- Прізвище
- По батькові
- Найменування програми
- Дата зарахування
- Дата завершення
- Оцінка

Ці колонки мають бути написані без помилок та відповідати наведеному формату, оскільки будь-яке відхилення призведе до некоректної обробки даних або помилок під час генерації сертифікатів.

Приклад того, як система обробляє ці дані, наведено нижче у вигляді фрагмента коду:

```
return sheet.map(row => ({
    name: row["Ім'я"],
    surname: row["Прізвище"],
    third_name: row["По батькові"],
    program: row["Найменування програми"],
    start_date: row["Дата зарахування"],
    end_date: row["Дата завершення"],
    grade: row["Оцінка"],
}));
```

Після зчитування даних ці значення автоматично підставляються у шаблон сертифіката. Щоб система коректно заповнила шаблон, у файлі-шаблоні (формат DOCX) необхідно використовувати відповідні змінні у фігурних дужках. Наприклад: `{name}`, `{surname}`, `{third_name}`, `{program}`, `{start_date}`, `{end_date}`, `{grade}`.

Ці змінні в шаблоні повинні точно відповідати назві полів, що повертаються з коду. У разі відсутності хоча б однієї змінної або помилки в її написанні (наприклад, зайвий пробіл чи неправильна назва) — сертифікат буде згенеровано некоректно або процес завершиться помилкою.

Однак важливо, щоб усі програми та їх результати були попередньо внесені в систему до початку процесу генерації сертифікатів. Якщо програма, вказана в Excel-файлі учасників, не знайдена в базі або до неї не прикріплено відповідний файл із результатами, система не зможе завершити генерацію і видасть повідомлення про помилку.