Міністерство освіти і науки України
Криворізький національний університет
Кафедра моделювання та програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти бакалавра

зі спеціальності 121 «Інженерія програмного забезпечення»

На тему: *Розробка онлайн-платформи для підтримки проходження виробничої практики*

Засвідчую, що в цій кваліфікаційній роботі немає запозичень із праць інших авторів без відповідних посилань.
Студент гр. ІПЗ-22-2
_____ Тищенко Б. С.

Керівник кваліфікаційної роботи _____ */ Стрюк А. М. /*

Завідувач кафедри _____ */ Стрюк А. М. /*

Кривий Ріг
2026

Криворізький національний університет
Факультет: *Інформаційних технологій*
Кафедра: *Моделювання та програмного забезпечення*
Освітньо-кваліфікаційний рівень: *бакалавр*
Спеціальність: *121 "Інженерія програмного забезпечення"*

ЗАТВЕРДЖУЮ
Зав. кафедри
_____*Стрюк А. М.*
«___»_____ 2026__ р.

ЗАВДАННЯ
на кваліфікаційну роботу
студенту групи ІПЗ-22-2 Тищенку Богдану Сергійовичу

1. Тема: Розробка онлайн-платформи для підтримки проходження виробничої практики, затверджено наказом по КНУ № _____ від «___» _____ 2026 р.
2. Термін подання студентом закінченого проекту «___» _____ 2026 р.
3. Вихідні дані по роботі: Пояснювальна записка: ____ сторінок, ____ рисунків, ____ таблиць, ____ додатки, ____ використаних у роботі джерел.
4. Зміст пояснювальної записки (перелік питань, що їх треба розробити): аналіз предметної області та існуючих рішень, постановка задачі та визначення вимог до програмного забезпечення; проєктування програмної системи (архітектура, база даних, API та інтерфейс); розробка програмного забезпечення серверної та клієнтської частин; опис методики роботи користувача.
5. Перелік графічного демонстраційного матеріалу: BPMN-модель процесу проходження практики; UML-діаграми (використання, компонентів, розгортання); ER-діаграма бази даних; схема програмної архітектури; копії екранів діючого програмного забезпечення.

Календарний план:

№	Найменування етапів кваліфікаційної роботи	Термін виконання етапів роботи
1	Формулювання мети та задач роботи	13.02.2026 – 20.02.2026
2	Аналіз предметної області та інформаційних джерел	21.02.2026 – 09.03.2026
3	Визначення вимог до програмного забезпечення	10.03.2026 – 18.03.2026
4	Проєктування архітектури системи	19.03.2026 – 23.03.2026
5	Проєктування бази даних	23.03.2026 – 30.03.2026
6	Проєктування API та інтерфейсу користувача	31.03.2026 – 15.04.2026
7	Розробка серверної частини	16.04.2026 – 13.05.2026
8	Розробка клієнтської частини	14.05.2026 – 21.05.2026
9	Тестування програмного забезпечення	21.05.2026 – 31.05.2026
10	Оформлення пояснювальної записки та демонстраційних матеріалів	31.05.2026 – 06.06.2026

Дата видачі завдання: «_____»_____ 2026 р.
Студент _____ Тищенко Б. С.
Керівник роботи _____ Стрюк А. М.

РЕФЕРАТ

Пояснювальна записка: 86 с., 7 табл., 12 рис., 1 дод., 11 джерел.

Об'єкт розробки — процес організації та проходження виробничої практики студентів на промисловому підприємстві.

Метою кваліфікаційної роботи є розробка онлайн-платформи для підтримки проходження виробничої практики студентів на промислових підприємствах, що автоматизує доставку навчального контенту, контроль знань, відстеження прогресу та видачу сертифікатів.

У роботі проаналізовано предметну область і наявні системи дистанційного навчання, сформульовано функціональні та нефункціональні вимоги до програмного забезпечення. Спроєктовано трирівневу клієнт-серверну архітектуру з декомпозицією серверної частини на REST API, Telegram-бот, планувальник нагадувань та обробник подій. Розроблено реляційну базу даних на PostgreSQL з понад двадцятьма сутностями, реалізовано серверну частину засобами Python та FastAPI з асинхронним доступом до даних через SQLAlchemy, авторизацією на основі JWT, послідовним розблокуванням навчального контенту, автоматичною перевіркою тестів та генерацією PDF-сертифікатів з UUID-верифікацією. Клієнтську частину побудовано як односторінковий застосунок на Vue 3 з управлінням станом через Pinia.

У результаті створено працездатну платформу, що дозволяє автоматизувати облік і супровід виробничої практики, скоротити ручну працю кураторів, забезпечити студентам зручний доступ до навчальних матеріалів і об'єктивне оцінювання знань, а підприємству — централізований контроль та аналітику проходження практики.

Результати роботи можуть бути впроваджені на промислових підприємствах з корпоративними програмами виробничої практики, зокрема ПАТ «АрселорМіттал Кривий Ріг».

КЛЮЧОВІ СЛОВА: ВИРОБНИЧА ПРАКТИКА, ОНЛАЙН-ПЛАТФОРМА, ВЕБ-ЗАСТОСУНОК, СИСТЕМА УПРАВЛІННЯ НАВЧАННЯМ, FASTAPI, VUE, POSTGRESQL, JWT-АВТОРИЗАЦІЯ, ЕЛЕКТРОННИЙ СЕРТИФІКАТ, REST API, ТЕСТУВАННЯ ЗНАНЬ.

ABSTRACT

Explanatory note: 86 p., 7 tab., 12 fig., 1 app., 11 references.

The object of development is the process of organizing and completing students' industrial practice at an industrial enterprise.

The aim of the qualification work is to develop an online platform for supporting students' industrial practice at industrial enterprises, which automates the delivery of educational content, knowledge assessment, progress tracking, and certificate issuance.

The work analyzes the subject area and existing distance-learning systems and defines the functional and non-functional requirements for the software. A three-tier client–server architecture is designed, with the server side decomposed into a REST API, a Telegram bot, a reminder scheduler, and an event handler. A relational PostgreSQL database with more than twenty entities is developed; the server side is implemented using Python and FastAPI with asynchronous data access via SQLAlchemy, JWT-based authorization, sequential unlocking of learning content, automatic test grading, and generation of PDF certificates with UUID verification. The client side is built as a single-page application on Vue 3 with state management through Pinia.

As a result, a functional platform has been created that automates the accounting and support of industrial practice, reduces the manual work of supervisors, provides students with convenient access to learning materials and objective knowledge assessment, and gives the enterprise centralized control and analytics of practice completion.

The results can be implemented at industrial enterprises with corporate industrial-practice programs, in particular PJSC "ArcelorMittal Kryvyi Rih".

KEYWORDS: INDUSTRIAL PRACTICE, ONLINE PLATFORM, WEB APPLICATION, LEARNING MANAGEMENT SYSTEM, FASTAPI, VUE,

POSTGRESQL, JWT AUTHORIZATION, ELECTRONIC CERTIFICATE, REST API, KNOWLEDGE TESTING.

ЗМІСТ

ВСТУП 9
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ 12
1.1 Критичний аналіз літературних джерел та існуючих рішень 12
1.2 Актуальність теми та характеристика предметної області 17
1.3 Цілі, завдання та вимоги до програмного забезпечення 21
Висновки до розділу 1 26
2 ПРОЄКТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ 27
2.1 Архітектура програмного забезпечення 27
2.2 Розробка бази даних 32
2.3 Проєктування API та інтерфейсу 41
Висновки до розділу 2 47
3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 49
3.1 Аналіз та обґрунтування обраних програмних засобів 49
3.2 Особливості реалізації серверної частини 55
3.3 Особливості реалізації клієнтської частини 62
3.4 Методика роботи користувача 69
Висновки до розділу 3 77
ВИСНОВКИ 78
ПЕРЕЛІК ПОСИЛАНЬ 81
Додаток А. Текст програми 83
А.1 Генерація та перевірка JWT-токенів 83
А.2 Залежність авторизації студента 84
А.3 Послідовне розблокування навчального контенту 84
А.4 Генерація PDF-сертифіката засобами ReportLab 85
А.5 Планування нагадувань засобами APScheduler 86
А.6 Перехоплювачі HTTP-запитів (Axios) 87
А.7 Сховище стану автентифікації (Pinia) 88
А.8 Таймер блокування повторних спроб (StudentModuleView) 89

ВСТУП

Цифровізація вищої освіти є одним із пріоритетних напрямів розвитку освітньої галузі України. Дистанційні та змішані форми навчання, електронні системи управління навчальним процесом і автоматизовані сервіси супроводу здобувачів дедалі ширше впроваджуються закладами вищої освіти та підприємствами-партнерами. Водночас окремі складові освітнього процесу залишаються недостатньо автоматизованими, і однією з них є виробнича практика — обов'язковий компонент освітньо-професійної програми, що забезпечує формування практичних умінь і навичок майбутнього фахівця в реальних умовах професійної діяльності.

На промислових підприємствах, які реалізують власні корпоративні програми практики та стажування, супровід практикантів здебільшого здійснюється засобами, не призначеними для цієї мети: облік ведеться в електронних таблицях, комунікація — електронною поштою та месенджерами, а підтвердження результатів оформлюється паперовими сертифікатами. Така організація характеризується відсутністю централізованого обліку й автоматичного відстеження прогресу, ручною перевіркою знань, відсутністю своєчасних нагадувань та трудомістким формуванням звітності. За одночасного проходження практики десятками й сотнями студентів ці недоліки істотно збільшують адміністративне навантаження та знижують прозорість процесу. Наявні універсальні системи управління навчанням не покривають повного переліку завдань супроводу виробничої практики на підприємстві без значних витрат на адаптацію. Зазначене зумовлює актуальність розроблення спеціалізованої онлайн-платформи автоматизації виробничої практики.

Метою кваліфікаційної роботи є розроблення онлайн-платформи для підтримки проходження виробничої практики, що автоматизує організацію та супровід практики студентів на промисловому підприємстві,

забезпечуючи структуроване подання навчального контенту, автоматичну перевірку знань, відстеження прогресу практикантів та документальне підтвердження результатів практики.

Для досягнення поставленої мети необхідно розв'язати такі завдання:

– проаналізувати предметну область, наявні системи управління навчанням та платформи супроводу практики й обґрунтувати доцільність власної розробки;

– визначити функціональні та нефункціональні вимоги до програмного забезпечення;

– спроєктувати архітектуру програмної системи, логічну модель бази даних та інтерфейс користувача;

– реалізувати серверну частину платформи: реєстрацію та авторизацію користувачів, управління навчальним контентом, автоматичну перевірку тестів, відстеження прогресу, генерацію сертифікатів і підсистему нагадувань;

– реалізувати клієнтську частину у вигляді односторінкового вебзастосунку для трьох ролей користувачів — студента, куратора та адміністратора;

– реалізувати аналітичний інструментарій для куратора з можливістю експорту звітності у форматі Excel та публічну верифікацію сертифікатів;

– провести тестування програмного забезпечення та підтвердити його працездатність.

Об'єктом дослідження є процес організації та супроводу проходження виробничої практики студентів на промисловому підприємстві.

Предметом дослідження є методи й програмні засоби автоматизації супроводу виробничої практики, зокрема управління навчальним контентом, відстеження прогресу, перевірки знань та документального підтвердження результатів.

Методи дослідження. У роботі застосовано методи критичного аналізу наукових джерел та порівняльного аналізу наявних програмних рішень — для обґрунтування доцільності розробки; методи моделювання бізнес-процесів (нотація BPMN) та об'єктно-орієнтованого аналізу й проєктування (мова UML) — для побудови моделі процесу та проєктування системи; методи проєктування реляційних баз даних — для розроблення схеми даних; методи об'єктно-орієнтованого та компонентного програмування — для реалізації серверної й клієнтської частин; методи функціонального тестування — для підтвердження працездатності програмного забезпечення.

Практичне значення одержаних результатів полягає в тому, що розроблену онлайн-платформу AMP доведено до рівня працездатного програмного забезпечення, придатного до впровадження. Платформа автоматизує повний цикл супроводу виробничої практики, усуває недоліки ручної організації, знижує трудомісткість обліку й контролю та підвищує прозорість процесу для всіх його учасників.

Галузь застосування результатів — промислові підприємства та заклади вищої освіти, що реалізують корпоративні програми виробничої практики й стажування. Орієнтиром для розробки слугувала програма практики ПАТ «АрселорМіттал Кривий Ріг».

Пояснювальна записка складається зі вступу, трьох розділів, висновків, переліку посилань та додатків. У першому розділі виконано аналіз предметної області та сформульовано вимоги до програмного забезпечення; у другому — спроєктовано архітектуру системи, базу даних та інтерфейс; у третьому — описано реалізацію серверної й клієнтської частин і методику роботи користувачів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Критичний аналіз літературних джерел та існуючих рішень

Виробнича практика є обов'язковою складовою освітньо-професійних програм підготовки фахівців і виконує роль містка між теоретичною підготовкою у закладі вищої освіти та реальною професійною діяльністю. В умовах цифровізації освіти все більшої актуальності набуває питання інформаційної підтримки цього процесу — від планування й розподілу здобувачів за базами практики до контролю виконання індивідуальних завдань, обліку результатів і формування звітної документації. Перш ніж проєктувати власне програмне рішення, доцільно проаналізувати наукові праці у відповідній галузі та критично оцінити наявні програмні продукти, які потенційно можуть бути застосовані для автоматизації виробничої практики.

Огляд наукових та літературних джерел

У низці досліджень, присвячених інформатизації освітнього процесу, наголошується, що системи управління навчанням (Learning Management System, LMS) стали основним інструментом організації дистанційного та змішаного навчання. LMS визначають як програмну платформу, що забезпечує створення, зберігання й доставку навчального контенту, управління користувачами та ролями, автоматизацію перевірки знань і формування звітності щодо успішності здобувачів [1]. Дослідники відзначають, що впровадження LMS суттєво знижує організаційні витрати, підвищує прозорість навчального процесу та дає змогу відстежувати активність кожного учасника в реальному часі.

Окремий пласт публікацій присвячено саме цифровізації управління практичною підготовкою. У роботах, що розглядають проєктування й реалізацію вебзастосунків для управління практикою (internship management

systems), систему такого класу трактують як централізовану цифрову платформу, що супроводжує повний життєвий цикл практики: від оприлюднення можливостей і розподілу здобувачів до контролю діяльності, оцінювання та формування підсумкових звітів [2]. Автори наголошують, що цифровізація процесу зменшує обсяг паперового документообігу, усуває адміністративні затримки й підвищує підзвітність усіх сторін — здобувача, керівника практики від кафедри та представника підприємства.

У дослідженнях, присвячених моніторингу проходження практики, окрему увагу приділено алгоритмам відстеження прогресу (progress tracking): автоматичному обліку виконаних етапів, ведению цифрового щоденника практики та наданню зворотного зв'язку керівником у режимі, наближеному до реального часу [3]. Такий підхід дає змогу виявляти відставання здобувачів на ранніх стадіях і своєчасно реагувати, що неможливо за традиційного «паперового» обліку. Спільним висновком розглянутих праць є те, що спеціалізовані рішення, орієнтовані на конкретний процес і конкретну організацію, забезпечують вищу ефективність, ніж універсальні платформи загального призначення.

Аналіз існуючих програмних рішень

Серед платформ, які найчастіше використовуються закладами освіти для організації дистанційного навчання й потенційно можуть застосовуватися для підтримки практики, варто розглянути Google Classroom, Moodle та eTutorium.

Google Classroom — хмарний сервіс компанії Google, що входить до пакета Google Workspace for Education. Його основними перевагами є простота інтерфейсу, безкоштовність для закладів освіти та безшовна інтеграція з іншими сервісами Google (Drive, Docs, Meet) [4]. Платформа орієнтована переважно на середню та старшу школу й розрахована на базові сценарії: публікацію матеріалів, видачу та збирання завдань, просте оцінювання. Водночас Google Classroom не передбачає гнучкого

структурування навчального контенту за модулями з послідовним розблокуванням, не має вбудованих засобів автоматичної генерації сертифікатів та спеціалізованої аналітики, а його логіка не адаптована під специфіку виробничої практики на підприємстві.

Moodle — найпоширеніша у світі система управління навчанням із відкритим кодом. Її ключовою перевагою є висока гнучкість і розширюваність завдяки великій бібліотеці плагінів, підтримці гейміфікації, інтерактивних елементів та розвиненим засобам оцінювання [5]. Незалежні аналітичні огляди відзначають Moodle як одне з найкращих рішень за можливостями оцінювання, мобільної підтримки та інтеграцій [6]. Проте за цю гнучкість доводиться платити складністю: блоковий багаторівневий інтерфейс є неінтуїтивним для авторів курсів, а розгортання й супровід власного екземпляра Moodle потребують значних адміністративних і технічних ресурсів. Будь-яка специфічна бізнес-логіка (наприклад, інтеграція з Telegram-ботом підприємства, автоматична верифікація сертифікатів за UUID або вивантаження звітності у форматі замовника) реалізується лише шляхом доопрацювання чи написання власних плагінів.

eTutorium — українська платформа, що поєднує систему дистанційного навчання з вебінарним майданчиком та конструктором курсів, тестів і опитувань [7]. Сервіс орієнтований передусім на репетиторів, експертів та освітні центри, які монетизують онлайн-курси, і його сильною стороною є вбудовані засоби проведення вебінарів та комунікації. Водночас платформа є пропрієтарною та надається за моделлю передплати, не розрахована на тісну інтеграцію з внутрішніми інформаційними системами промислового підприємства й не покриває специфічних завдань обліку виробничої практики.

Для наочного зіставлення проаналізованих платформ за критеріями, релевантними для предметної області, складено порівняльну таблицю (табл.

1.1), у якій поряд із наявними рішеннями наведено очікувані характеристики розроблюваної онлайн-платформи AMP.

Таблиця 1.1 – Порівняльний аналіз платформ дистанційного навчання

Критерій	**Google Classroom**	**Moodle**	**eTutorium**	**AMP (розробка)**
Модель розповсюдження	хмарний сервіс	відкритий код	пропрієтарна, передплата	власна розробка
Цільова орієнтація	загальна освіта	універсальна LMS	курси, вебінари	виробнича практика
Структурування контенту за модулями	обмежене	повне	часткове	повне
Послідовне розблокування контенту	немає	через плагіни	немає	вбудоване
Автоматичне тестування з обмеженням спроб	базове	є	є	є (3 спроби, кулдаун)
Генерація сертифікатів з верифікацією	немає	через плагіни	базова	вбудована (UUID)
Нагадування Email / Telegram	частково (Email)	через плагіни	Email	Email + Telegram-бот
Аналітичний дашборд та експорт Excel	обмежений	є	базовий	є (звіт для куратора)
Інтеграція з системами підприємства	обмежена	потребує доробки	обмежена	проєктується під замовника
Вартість для ЗВО	безкоштовно	безкоштовно (+супровід)	передплата	власна (без ліцензій)

Аналіз таблиці 1.1 свідчить, що жодне з розглянутих рішень не покриває повного переліку вимог до автоматизації виробничої практики на промисловому підприємстві. Google Classroom є надто спрощеним і не підтримує необхідної бізнес-логіки; Moodle забезпечує потрібну

функціональність лише ціною значного доопрацювання та ресурсомісткого супроводу; eTutorium орієнтований на інший сегмент і обмежений пропрієтарною моделлю. Спільним недоліком усіх універсальних платформ є відсутність «з коробки» таких специфічних механізмів, як послідовне розблокування навчального контенту, автоматична генерація сертифікатів із верифікацією за унікальним ідентифікатором, інтеграція з корпоративним Telegram-ботом та формування звітності у форматі, потрібному куратору від підприємства.

Висновок про необхідність власної розробки

Проведений критичний аналіз літературних джерел та наявних програмних продуктів дає підстави стверджувати, що ринок не пропонує готового рішення, яке б комплексно й без надлишкових витрат на адаптацію відповідало вимогам автоматизації виробничої практики в умовах конкретного промислового підприємства. Універсальні LMS або занадто прості (Google Classroom), або надмірно складні в супроводі та потребують доопрацювання під кожну специфічну вимогу (Moodle), або орієнтовані на інший сегмент і закриті для глибокої інтеграції (eTutorium). Це обґрунтовує доцільність розроблення власної спеціалізованої онлайн-платформи, побудованої на сучасному відкритому технологічному стеку, яка враховує особливості процесу виробничої практики, інтегрується з інформаційним середовищем підприємства та не потребує ліцензійних відрахувань. Постановці конкретних цілей, завдань і вимог до такого програмного забезпечення присвячено наступні підрозділи цього розділу.

1.2 Актуальність теми та характеристика предметної області

Поняття виробничої практики в системі вищої освіти

Виробнича практика — це обов'язковий компонент освітньо-професійної програми, метою якого є формування у здобувача вищої освіти практичних умінь і навичок, закріплення та поглиблення теоретичних знань,

набутих під час аудиторного навчання, а також ознайомлення з реальними умовами майбутньої професійної діяльності. Згідно із Законом України «Про вищу освіту», практична підготовка є невід'ємною складовою процесу підготовки фахівців і здійснюється на базах практики — підприємствах, в установах та організаціях, що відповідають профілю спеціальності [8]. Організація практики регламентується відповідним положенням і передбачає укладання договорів між закладом вищої освіти та підприємством, призначення керівників практики від кафедри й від підприємства, видачу індивідуальних завдань, ведення щоденника практики та підготовку звітної документації [9].

Для промислових підприємств виробнича практика має додаткове значення: вона є інструментом профорієнтації, добору й попередньої адаптації потенційних працівників. Великі підприємства, зокрема ПАТ «АрселорМіттал Кривий Ріг», реалізують власні корпоративні програми практики й стажування, у межах яких студент опановує не лише фахові, а й специфічні для підприємства компетентності, ознайомлюється з технологічними процесами, правилами охорони праці та внутрішніми стандартами. Масштаб таких програм — десятки й сотні практикантів одночасно — робить ручне адміністрування процесу складним і трудомістким.

Характеристика поточного стану процесу («Як є»)

Аналіз типової організації виробничої практики на підприємстві показує, що супровід процесу здебільшого здійснюється засобами, не призначеними для цієї мети. Облік практикантів та їхнього прогресу ведеться у таблицях Microsoft Excel, комунікація між координатором практики, наставниками та студентами відбувається через електронну пошту й месенджери, а навчальні матеріали передаються у вигляді окремих файлів або паперових роздруківок. Підтвердження проходження практики

оформлюється паперовими довідками й сертифікатами, які підписуються та зберігаються вручну.

Така організація має низку суттєвих недоліків:

а) відсутність централізованого обліку — дані про студентів, програми практики та результати розпорошені між кількома таблицями й поштовими скриньками, що ускладнює пошук і призводить до дублювання та помилок;

б) ручне відстеження прогресу — координатор змушений власноруч зводити інформацію про те, які завдання виконав кожен практикант, через що відставання виявляються із запізненням;

в) відсутність автоматичних нагадувань — студенти пропускають терміни виконання завдань, оскільки система не повідомляє їх про наближення дедлайнів;

г) ручна перевірка знань — тестування й оцінювання здійснюються «вручну», що збільшує навантаження на наставників і вносить суб'єктивність;

ґ) трудомістке формування звітності — підготовка зведених звітів про проходження практики потребує значних витрат часу й не підтримує оперативного аналізу;

д) ризики паперового документообігу — паперові сертифікати легко втратити чи підробити, а їх перевірка на справжність є непростою.

Сукупно ці проблеми знижують прозорість процесу, збільшують адміністративне навантаження й погіршують якість супроводу практиканта. Це підтверджує висновки досліджень, у яких цифровізація управління практикою розглядається як спосіб підвищення ефективності, прозорості та підзвітності за одночасного зменшення обсягу паперової роботи й адміністративних затримок [2].

Модель процесу «Як має бути»

Для усунення зазначених недоліків процес проходження виробничої практики було перепроєктовано із застосуванням нотації моделювання бізнес-процесів BPMN. Цільову модель «Як має бути» побудовано у вигляді процесу із чотирма доріжками (lanes), що відповідають ролям учасників: HR-адміністратор підприємства, студент-практикант, куратор (наставник) та автоматизована система AMP. Ключова відмінність цільової моделі полягає у перенесенні рутинних операцій на автоматизовану систему. Графічну модель цільового процесу наведено на рисунку 1.1.

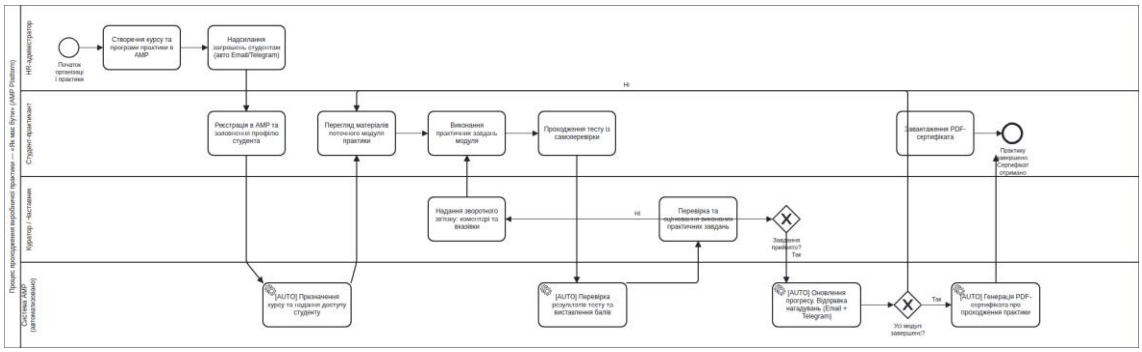

Рисунок 1.1 – Модель процесу проходження виробничої практики «Як має бути»

Процес розгортається у такій послідовності. HR-адміністратор створює в системі курс та програму практики, після чого система автоматично надсилає студентам запрошення електронною поштою та через Telegram. Студент реєструється у платформі й заповнює профіль, а система автоматично призначає йому відповідний курс і надає доступ до першого модуля. Далі студент переглядає матеріали поточного модуля, виконує практичні завдання та проходить тест для самоперевірки. Перевірка результатів тесту й виставлення балів виконуються системою автоматично, без участі людини.

Виконані практичні завдання надходять куратору на перевірку. На цьому етапі передбачено шлюз прийняття рішення «Завдання прийнято?»: у разі негативного результату куратор надає зворотний зв'язок із коментарями

та вказівками, і студент доопрацьовує завдання; у разі позитивного — система оновлює прогрес студента й розсилає відповідні нагадування. Після цього спрацьовує шлюз «Усі модулі завершено?»: якщо залишилися незавершені модулі, студент переходить до наступного, повторюючи цикл; якщо ж усі модулі пройдено, система автоматично генерує PDF-сертифікат про проходження практики. Студент завантажує сертифікат, на чому процес завершується.

Зіставлення моделей «Як є» та «Як має бути» демонструє, що в цільовому процесі автоматизуються саме ті операції, які раніше виконувалися вручну: призначення курсу й надання доступу, перевірка тестів, відстеження прогресу, розсилання нагадувань і формування підтвердного документа. За людиною (куратором) залишається лише змістовна частина — оцінювання практичних завдань і надання зворотного зв'язку, де експертне судження є необхідним. Такий розподіл обов'язків зменшує адміністративне навантаження, підвищує прозорість і оперативність процесу та усуває більшість недоліків, притаманних поточному стану.

Отже, актуальність теми зумовлена потребою промислових підприємств у спеціалізованому інструменті автоматизації виробничої практики, який реалізує описану цільову модель. Конкретні цілі, завдання та вимоги до такого програмного забезпечення сформульовано в наступному підрозділі.

1.3 Цілі, завдання та вимоги до програмного забезпечення

На підставі проведеного аналізу предметної області, наявних програмних рішень та виявлених недоліків ручної організації виробничої практики сформульовано мету та завдання розробки, а також визначено вимоги до програмного забезпечення.

Метою розробки є створення онлайн-платформи AMP (ArcelorMittal Practice) — вебзастосунку, що автоматизує організацію та супровід проходження виробничої практики студентів на промисловому підприємстві, забезпечуючи структуроване подання навчального контенту, автоматичну перевірку знань, відстеження прогресу практикантів та документальне підтвердження результатів практики.

Для досягнення поставленої мети необхідно розв'язати такі завдання:

– проаналізувати предметну область, наявні системи управління навчанням і платформи супроводу практики, обґрунтувати доцільність власної розробки; – спроєктувати архітектуру програмної системи та логічну модель бази даних, що відповідають визначеним вимогам; – реалізувати механізм реєстрації, автентифікації та авторизації користувачів із розмежуванням прав доступу за ролями; – розробити підсистему управління навчальним контентом (курси, модулі, завдання, тестові питання) з послідовним розблокуванням матеріалів; – реалізувати автоматичну перевірку результатів тестування та відстеження прогресу проходження практики; – реалізувати автоматичну генерацію PDF-сертифіката з можливістю його публічної верифікації за унікальним ідентифікатором; – розробити підсистему сповіщень і нагадувань через електронну пошту та Telegram; – реалізувати аналітичний інструментарій для куратора з можливістю експорту звітності у форматі Excel; – провести тестування програмного забезпечення та підтвердити його працездатність.

Функціональні вимоги

Функціональні вимоги визначають перелік дій, які система має надавати своїм користувачам. Для їх формалізації застосовано діаграму прецедентів (Use Case Diagram) мови UML, що відображає взаємодію акторів із системою. У межах платформи виокремлено чотири актори:

– Студент — основний користувач, що проходить практику: реєструється, заповнює профіль, опановує навчальний контент, складає

тести та отримує сертифікат; – Куратор — представник підприємства або викладач, що відстежує прогрес практикантів, переглядає аналітику та формує звітність; – Адміністратор — користувач, що наповнює систему навчальним контентом та керує довідковими даними; – Гість — неавтентифікований користувач, якому доступна лише публічна верифікація сертифіката за його ідентифікатором.

Узагальнену модель взаємодії акторів із системою наведено на рисунку 1.2.

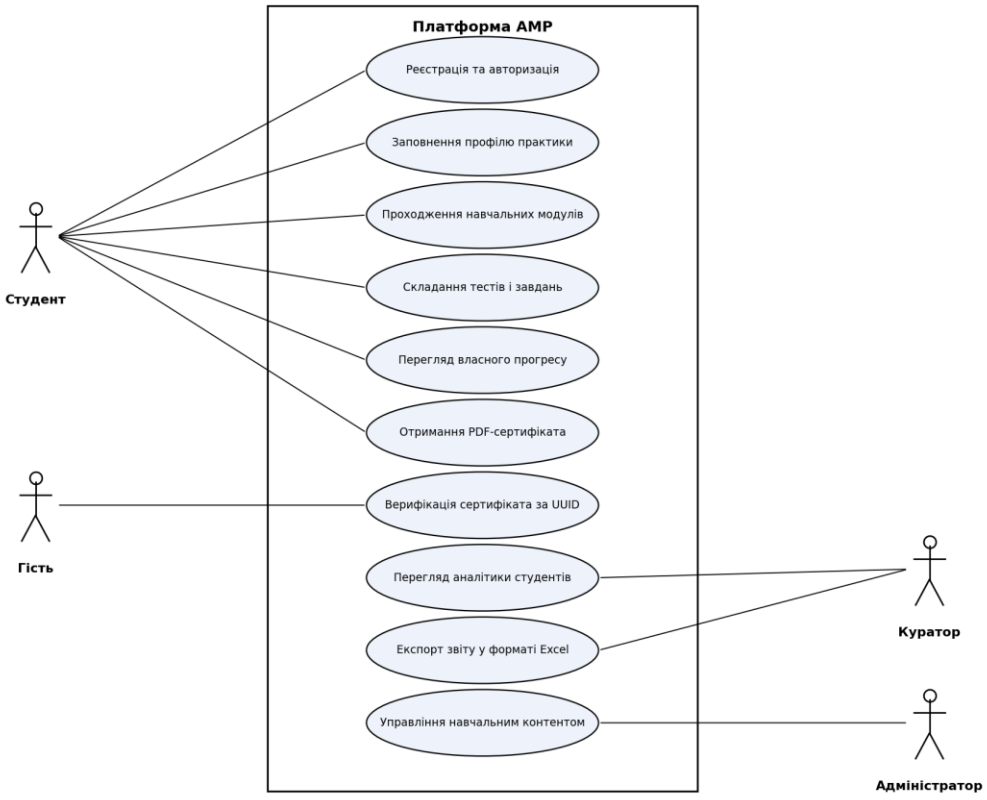

Рисунок 1.2 – Діаграма прецедентів платформи AMP

Деталізований перелік функціональних вимог за акторами наведено нижче.

Для актора «Студент» система повинна забезпечувати:

– реєстрацію нового облікового запису та авторизацію за допомогою електронної пошти й пароля, а також відновлення доступу через токен скидання пароля; – створення та заповнення профілю практики із

зазначенням закладу освіти, спеціальності, виду практики й рівня освіти, на підставі яких система автоматично призначає відповідний курс; – перегляд структури призначеного курсу, його модулів і завдань із послідовним розблокуванням контенту в міру проходження; – опрацювання навчальних матеріалів модуля, виконання практичних завдань та складання тестів із автоматичною перевіркою відповідей; – перегляд статистики власного прогресу — пройдених модулів, набраних балів і кількості використаних спроб; – завантаження автоматично згенерованого PDF-сертифіката після успішного завершення всіх модулів курсу; – отримання сповіщень і нагадувань електронною поштою та через Telegram, а також керування налаштуваннями нагадувань.

Для актора «Куратор» система повинна забезпечувати:

– перегляд списку студентів-практикантів та їхніх профілів; – отримання агрегованої аналітики щодо прогресу проходження практики групою студентів; – формування й вивантаження звіту про результати практики у форматі Excel.

Для актора «Адміністратор» система повинна забезпечувати:

– управління навчальним контентом — створення та редагування курсів, модулів, завдань і тестових питань; – ведення довідкових даних (заклади освіти, спеціальності, види практики, рівні освіти); – завантаження файлів навчальних матеріалів.

Для актора «Гість» система повинна забезпечувати верифікацію автентичності сертифіката за унікальним ідентифікатором (UUID) без потреби в авторизації.

Нефункціональні вимоги

Нефункціональні вимоги характеризують якісні властивості системи та обмеження, що накладаються на її реалізацію й експлуатацію.

Продуктивність. Час обробки типового запиту на серверній частині не повинен перевищувати 500 мс за умови штатного навантаження.

Досягнення цього показника забезпечується застосуванням асинхронної моделі обробки запитів (FastAPI, asyncpg), кешуванням довідкових і часто запитуваних даних у Redis, а також індексуванням ключових полів бази даних.

Безпека. Автентифікація користувачів реалізується за технологією JWT (JSON Web Token); паролі зберігаються виключно у вигляді криптографічних хешів. Доступ до захищених ресурсів розмежовується за ролями користувачів. Персональні дані, що підлягають захисту, зберігаються в зашифрованому вигляді, а ідентифікатори ключових сутностей формуються як UUID, що ускладнює їх передбачення. Кількість спроб проходження тесту обмежується (TASK_TRY_LIMIT = 3) із встановленням періоду блокування (TASK_TRY_TIMEOUT_MIN = 1440 хв) для запобігання зловживанням.

Масштабованість. Серверну частину побудовано за модульним принципом із поділом на незалежні процеси (REST API, Telegram-бот, планувальник завдань, обробник подій), що дає змогу масштабувати окремі компоненти незалежно. Контейнеризація на основі Docker та використання Redis як спільної черги повідомлень між сервісами забезпечують можливість горизонтального масштабування під зростання навантаження.

Адаптивність. Клієнтську частину реалізовано як односторінковий застосунок (SPA) з адаптивним інтерфейсом, що коректно відображається на пристроях із різною роздільною здатністю екрана — від настільних комп'ютерів до мобільних пристроїв.

Надійність і зручність. Система має забезпечувати цілісність даних за рахунок транзакційності операцій та каскадних обмежень цілісності в базі даних, а також інтуїтивно зрозумілий інтерфейс, що не потребує спеціальної підготовки користувача. Узагальнення нефункціональних вимог наведено в таблиці 1.2.

Таблиця 1.2 – Нефункціональні вимоги до програмного забезпечення

Категорія	Вимога	Засіб реалізації
Продуктивність	Час відповіді сервера ≤ 500 мс	Асинхронна обробка, кешування Redis, індекси БД
Безпека	Автентифікація, розмежування доступу, захист даних	JWT, хешування паролів, шифрування полів, UUID
Масштабованість	Незалежне масштабування компонентів	Модульна архітектура, Docker, черга Redis
Адаптивність	Коректне відображення на різних пристроях	Адаптивний інтерфейс SPA (Vue 3)
Надійність	Цілісність та узгодженість даних	Транзакції, каскадні обмеження цілісності

Визначені функціональні та нефункціональні вимоги є вихідними даними для подальшого проєктування архітектури програмної системи та бази даних, що розглядається у другому розділі.

Висновки до розділу 1

У першому розділі проаналізовано предметну область автоматизації виробничої практики, критично оцінено наукові джерела та наявні програмні рішення. Порівняльний аналіз платформ Google Classroom, Moodle та eTutorium (табл. 1.1) показав, що жодна з універсальних систем управління навчанням не покриває повного переліку вимог до супроводу виробничої практики на промисловому підприємстві без надмірних витрат на адаптацію, що обґрунтовує доцільність власної спеціалізованої розробки. Характеристика поточного стану процесу виявила ключові недоліки ручної організації практики — відсутність централізованого обліку, ручне відстеження прогресу й перевірку знань, відсутність автоматичних нагадувань та ризики паперового документообігу, — для усунення яких побудовано цільову модель процесу «Як має бути» у нотації BPMN (рис. 1.1). На основі проведеного аналізу сформульовано мету й завдання роботи та визначено функціональні (рис. 1.2) і нефункціональні (табл. 1.2) вимоги до платформи AMP. Визначені вимоги є вихідними даними для

проєктування архітектури програмної системи та бази даних, якому присвячено наступний розділ.

2 ПРОЄКТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

2.1 Архітектура програмного забезпечення

Проєктування архітектури є першим і визначальним етапом розробки програмної системи, на якому ухвалюються рішення щодо загальної організації застосунку, розподілу відповідальності між його складниками та способів їхньої взаємодії. Від обґрунтованості цих рішень залежать такі характеристики системи, як масштабованість, надійність, зручність супроводу та можливість подальшого розвитку. У цьому підрозділі розглянуто загальну архітектуру онлайн-платформи AMP (ArcelorMittal Practice), її декомпозицію на окремі служби, а також наведено діаграми компонентів та розгортання, що ілюструють статичну структуру системи й розподіл її складників між обчислювальними вузлами.

Загальна архітектура системи

Для платформи AMP обрано класичну трирівневу клієнт-серверну архітектуру, що передбачає поділ системи на три логічні рівні: рівень подання, рівень бізнес-логіки та рівень даних. Такий поділ забезпечує слабку зв'язаність складників, спрощує тестування та супровід, а також дозволяє незалежно розвивати й масштабувати кожен із рівнів.

Рівень подання реалізовано у вигляді односторінкового вебзастосунку (SPA, Single Page Application) на основі фреймворку Vue 3. Клієнтська частина виконується у браузері користувача й відповідає виключно за відображення інтерфейсу та взаємодію з користувачем; вона не містить бізнес-логіки й отримує всі дані від сервера через прикладний програмний інтерфейс. Додатковим каналом подання виступає Telegram-бот, через який користувач отримує сповіщення та нагадування.

Рівень бізнес-логіки становить серверна частина, що реалізує всю прикладну логіку платформи: автентифікацію та авторизацію користувачів,

управління навчальним контентом, перевірку тестів, відстеження прогресу, генерацію сертифікатів і розсилання нагадувань. Серверна частина не має власного графічного інтерфейсу й надає функціональність через REST API, що уможливлює її використання різними клієнтами.

Рівень даних утворюють система керування базами даних PostgreSQL 17, яка є основним надійним сховищем для всіх сутностей платформи, та сховище Redis, що використовується для кешування й як брокер повідомлень між службами серверної частини. Відокремлення рівня даних дозволяє централізовано керувати збереженням інформації та забезпечує її цілісність незалежно від кількості клієнтів.

Взаємодія між клієнтським і серверним рівнями відбувається за протоколом HTTPS у форматі REST-запитів; стан автентифікації передається у вигляді токена JWT (JSON Web Token) у заголовку запиту, що відповідає принципу відсутності збереження стану сесії на сервері (stateless) і спрощує горизонтальне масштабування.

Мікросервісна декомпозиція серверної частини

Хоча платформа AMP розгортається як єдиний застосунок, її серверна частина декомпонована на чотири незалежні служби (процеси), кожна з яких має власну точку входу й вузьку зону відповідальності. Такий підхід поєднує переваги сервіс-орієнтованої архітектури — поділ відповідальності та незалежність життєвих циклів процесів — із простотою розгортання моноблочного застосунку. Усі служби спільно використовують єдину кодову базу (моделі даних, схеми, конфігурацію), що усуває дублювання й гарантує узгодженість контрактів даних між процесами.

Серверна частина складається з таких служб:

– служба REST API (`main_api.py`) — основний процес, побудований на фреймворку FastAPI та запущений ASGI-сервером Uvicorn. Вона обробляє всі HTTP-запити клієнтського застосунку, реалізує бізнес-логіку та взаємодіє з базою даних через асинхронний ORM SQLAlchemy;

– Telegram-бот (`main_bot.py`) — процес на основі бібліотеки pyTelegramBotAPI, що працює в режимі довготривалого опитування (long polling) і відповідає за двосторонню взаємодію з користувачами через месенджер Telegram, зокрема за прив'язку облікових записів та доставлення сповіщень;

– планувальник (`main_scheduler.py`) — процес на основі бібліотеки APScheduler, що виконує заплановані за розкладом (cron) завдання, насамперед формування й розсилання нагадувань студентам про необхідність продовжити проходження практики;

– обробник подій (`main_pubsub.py`) — процес-передплатник, що прослуховує канали повідомлень Redis Pub/Sub і асинхронно опрацьовує події, опубліковані іншими службами (надсилання повідомлень у Telegram та запис сповіщень). Завдяки цьому тривалі або зовнішні операції винесено з основного потоку обробки HTTP-запитів, що зменшує час відповіді API.

Взаємодія між службами організована за двома механізмами. Спільний доступ до даних здійснюється через базу даних PostgreSQL, яка виступає єдиним джерелом правди для всіх процесів. Асинхронний обмін подіями реалізовано через механізм публікації-передплати (Publish/Subscribe) на базі Redis: служба API публікує подію в канал (наприклад, «надіслати сповіщення»), а обробник подій незалежно її отримує й виконує. Така слабка зв'язаність дозволяє додавати нові типи обробників без зміни коду, що публікує події.

Логічну структуру платформи та зв'язки між її складниками подано на діаграмі компонентів (рис. 2.1).

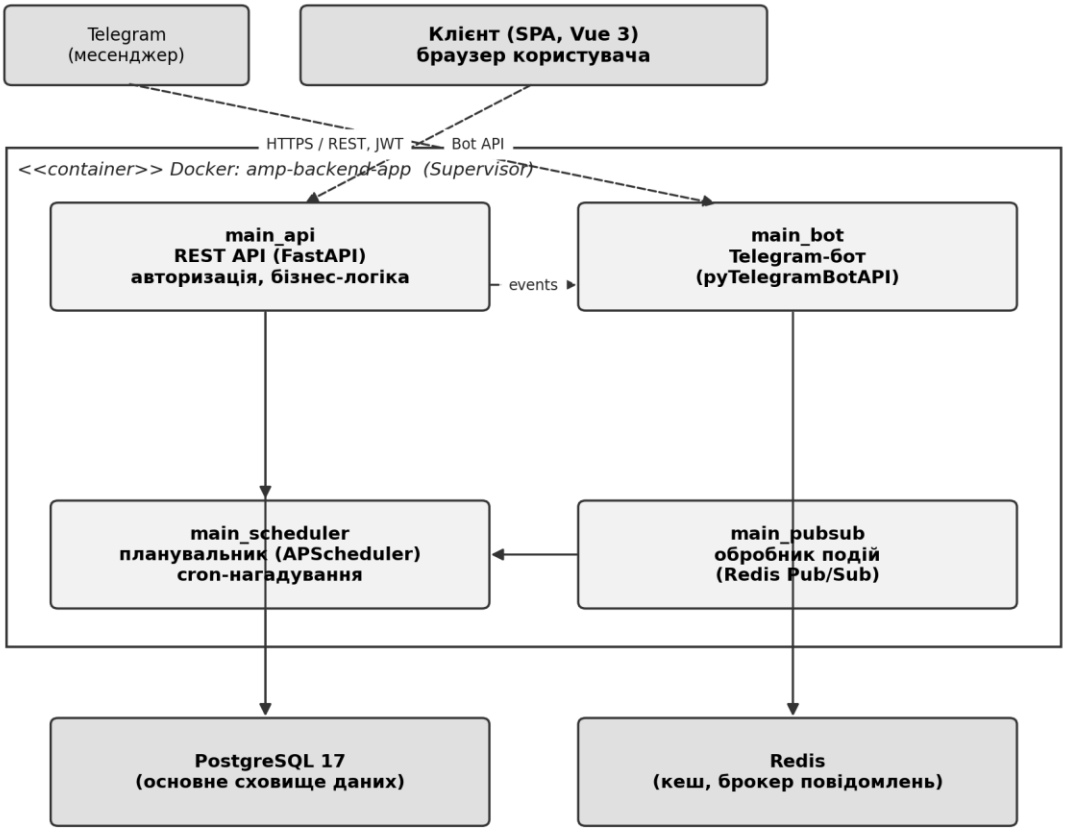

Рисунок 2.1 – Діаграма компонентів платформи AMP

Як видно з діаграми, клієнтський застосунок і Telegram взаємодіють із системою лише через відповідні служби серверної частини, а доступ до сховищ даних інкапсульовано всередині серверного рівня. Це забезпечує контрольованість усіх операцій із даними та єдину точку застосування правил безпеки.

Контейнеризація та розгортання

Для забезпечення відтворюваності середовища, ізоляції залежностей та спрощення розгортання платформу контейнеризовано засобами Docker, а оркестрування контейнерів виконується за допомогою Docker Compose. Система розгортається у вигляді трьох контейнерів-служб: контейнера застосунку, контейнера бази даних PostgreSQL та контейнера Redis.

Особливістю розгортання є те, що всі чотири процеси серверної частини (API, бот, планувальник, обробник подій) виконуються в межах

одного контейнера застосунку під керуванням процес-менеджера Supervisor. Конфігурація Supervisor описує кожен процес як окрему програму з увімкненими параметрами автозапуску (`autostart`) та автоматичного перезапуску в разі збою (`autorestart`), завдяки чому аварійне завершення однієї служби не впливає на роботу інших і вона автоматично відновлюється. Такий підхід обрано як компроміс між чистотою мікросервісної ізоляції та обмеженістю апаратних ресурсів цільового сервера: він зменшує накладні витрати на окремі контейнери, зберігаючи при цьому незалежність життєвих циклів процесів.

Контейнер бази даних PostgreSQL зберігає дані у змонтованому томі (volume), що забезпечує їхню персистентність між перезапусками контейнера. У робочому середовищі параметри роботи PostgreSQL додатково налаштовано під обмежені ресурси сервера (обсяг спільних буферів, максимальна кількість з'єднань тощо), а для кожного контейнера задано ліміти використання процесора та оперативної пам'яті. Для контролю готовності служб у конфігурації визначено перевірки життєздатності (healthcheck), а запуск контейнера застосунку відкладається до моменту, коли база даних і Redis стають доступними (`depends_on` з умовою `service_healthy`).

Фізичний розподіл складників системи між контейнерами та вузлами, а також канали їхньої взаємодії наведено на діаграмі розгортання (рис. 2.2).

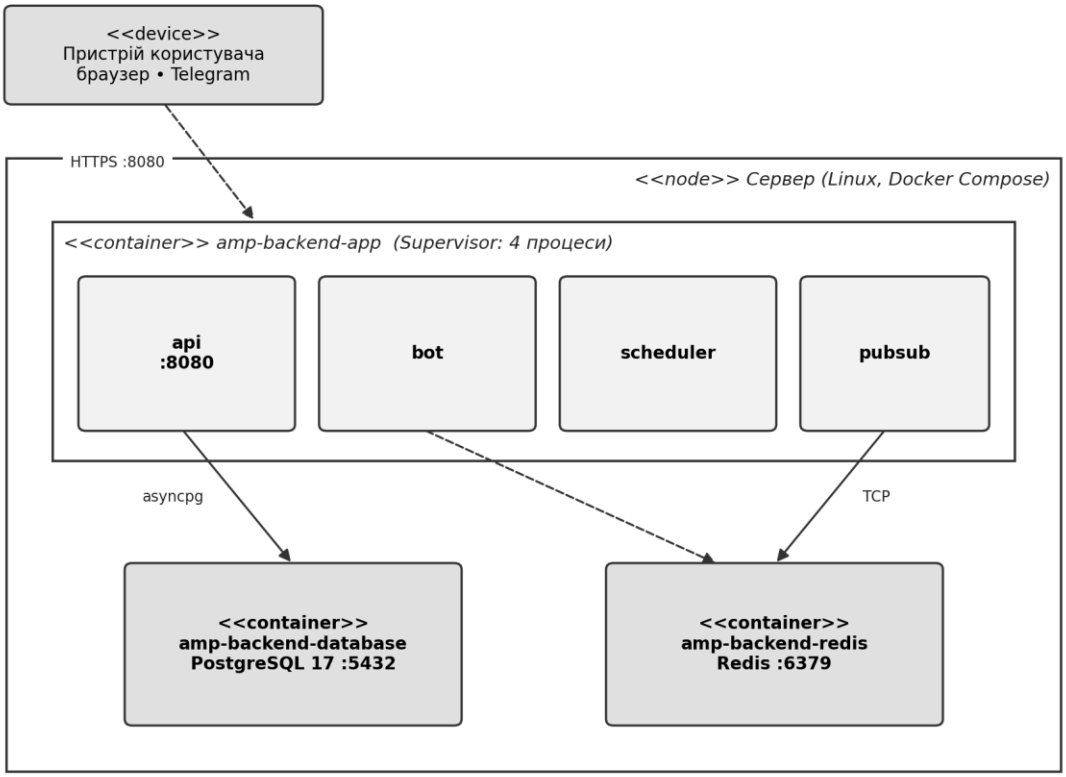

Рисунок 2.2 – Діаграма розгортання платформи AMP

Зовні доступним є лише порт служби API (8080), через який проксі-сервер спрямовує запити користувачів; контейнери бази даних і Redis у робочому середовищі ізольовано у внутрішній мережі Docker і недоступні безпосередньо ззовні, що підвищує рівень захищеності системи. Запропонована архітектура є водночас достатньо простою для розгортання на одному сервері й відкритою до подальшого масштабування: за зростання навантаження окремі служби або сховища можуть бути винесені в окремі контейнери чи на окремі вузли без зміни прикладної логіки.

2.2 Розробка бази даних

База даних є центральним сховищем усіх відомостей платформи AMP: облікових записів студентів та адміністраторів, навчального контенту, результатів проходження тестів, виданих сертифікатів та налаштувань сповіщень. Коректне проєктування структури даних безпосередньо впливає на цілісність інформації, продуктивність запитів та зручність подальшого

розвитку системи. У цьому підрозділі обґрунтовано вибір системи керування базами даних, описано концептуальну схему та структуру таблиць платформи, а також розглянуто ключові архітектурні рішення, прийняті при побудові схеми.

Вибір системи керування базами даних

Для збереження даних платформи AMP обрано PostgreSQL версії 17 — відкриту об'єктно-реляційну систему керування базами даних, яка поєднує потужну реалізацію стандарту SQL, розширені типи даних та доведену надійність у виробничих середовищах [10].

Основними критеріями вибору PostgreSQL стали такі його властивості.

– Повна підтримка ACID-властивостей (атомарність, узгодженість, ізольованість, довговічність) гарантує цілісність даних при паралельних операціях — зокрема, під час одночасного відстеження прогресу кількох студентів та генерації сертифікатів.

– Нативна підтримка типу UUID і вбудована функція `gen_random_uuid()` (розширення pgcrypto) дозволяють генерувати унікальні ідентифікатори безпосередньо на рівні сервера бази даних, що використовується у платформі для публічних посилань на сертифікати.

– Підтримка масивів через тип `ARRAY` дозволила зберігати допустимі роки навчання (`study_years`) у таблиці `course_types` як єдине поле без нормалізації у допоміжну таблицю.

– Асинхронний драйвер asyncpg та підтримка з боку SQLAlchemy 2.0 [11] у режимі `async` забезпечили можливість реалізації неблокуючих операцій з базою даних у FastAPI-застосунку, що є критично важливим для продуктивності ASGI-сервера.

– Зрілий інструментарій міграцій Alembic дозволяє контролювати еволюцію схеми за допомогою версійованих скриптів DDL; для платформи AMP розроблено 38 міграцій, кожна з яких є атомарною та оборотньою.

Взаємодія із СУБД реалізована через ORM SQLAlchemy 2.0 в декларативному стилі: кожній таблиці відповідає окремий клас-модель, поля задаються анотаціями типів через `Mapped[T]`, а зв'язки між таблицями описані за допомогою `relationship()`. Такий підхід дозволяє писати типобезпечний код і уникати ручного формування SQL-запитів у більшості випадків.

Концептуальна схема бази даних

База даних платформи AMP налічує 28 таблиць, які можна розподілити за функціональними групами: користувачі та профілі, навчальний контент, відстеження прогресу, сервісні таблиці та довідники. Концептуальна схема, що відображає основні сутності та зв'язки між ними, наведена на рисунку 2.3.

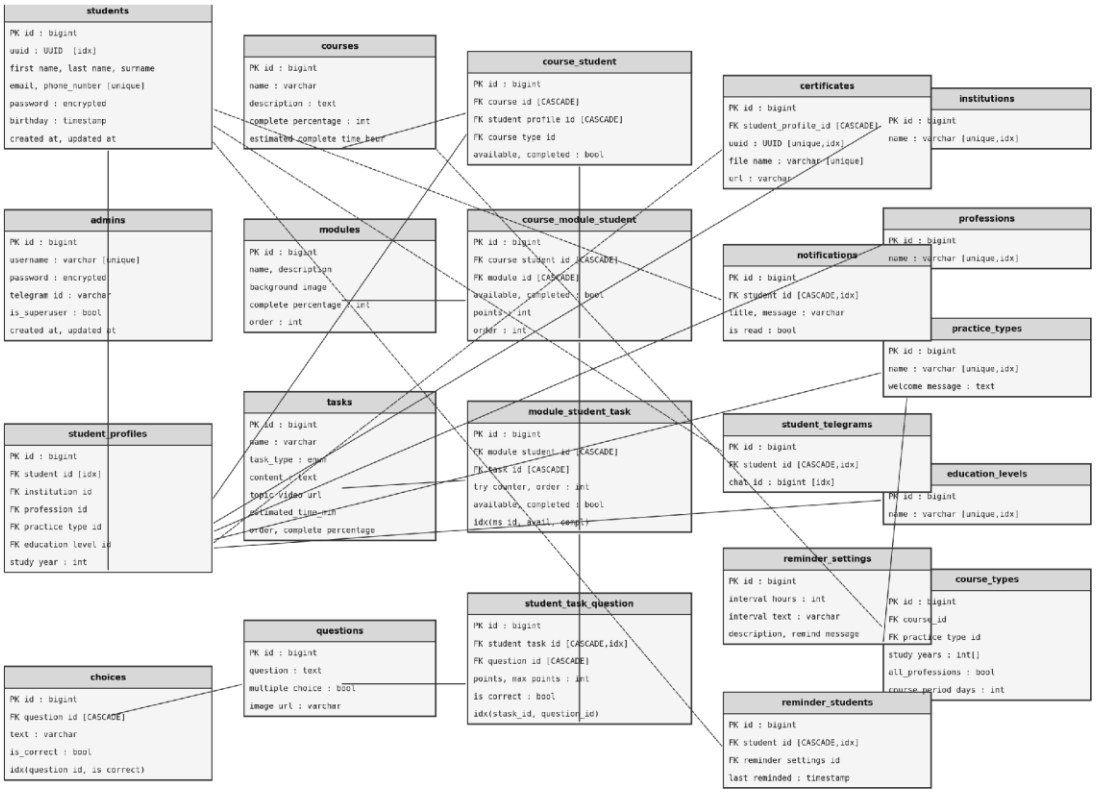

Рисунок 2.3 – Концептуальна схема бази даних платформи AMP

Центральними сутностями є `students` та `student_profiles`: перша зберігає автентифікаційні дані та особисту інформацію студента, друга —

академічний контекст проходження практики. Між ними існує зв'язок «один до багатьох», що дозволяє студенту мати кілька профілів для різних видів практики. Навчальний контент організовано в ієрархію курс → модуль → завдання → питання → варіанти відповіді. Для відстеження прогресу кожного студента ця ієрархія дублюється в проміжних таблицях із суфіксом `_student`, що зберігають стан виконання та набрані бали.

Структура таблиць бази даних

У таблиці 2.1 наведено перелік усіх таблиць платформи AMP з їхнім призначенням.

Таблиця 2.1 – Перелік таблиць бази даних платформи AMP

Група	**Таблиця**	**Призначення**
Користувачі	students	Облікові записи студентів (автентифікація, персональні дані)
Користувачі	admins	Облікові записи адміністраторів
Профілі	student_profiles	Профілі студента для кожного виду практики
Контент	courses	Навчальні курси (назва, опис, параметри)
Контент	modules	Модулі курсу (розділи з навчальним матеріалом)
Контент	tasks	Завдання модуля (відеолекція або тест)
Контент	questions	Тестові питання
Контент	choices	Варіанти відповіді на питання
Прогрес	course_student	Запис про зарахування студента на курс та загальний стан
Прогрес	course_module_student	Прогрес студента в межах конкретного модуля
Прогрес	module_student_task	Прогрес студента за кожним завданням модуля
Прогрес	student_task_question	Збережені відповіді та бали за питаннями тесту
Сервісні	certificates	Видані PDF-сертифікати з UUID для верифікації
Сервісні	notifications	Внутрішні сповіщення для студентів
Сервісні	student_telegrams	Зв'язка облікового запису студента з Telegram chat_id
Сервісні	telegram_messages	Архів повідомлень Telegram-бота

Сервісні	reminder_settings	Налаштування шаблонів нагадувань
Сервісні	reminder_students	Підписки студентів на нагадування
Довідники	institutions	Навчальні заклади студентів
Довідники	professions	Спеціальності / посади на підприємстві
Довідники	practice_types	Типи практики (виробнича, переддипломна тощо)
Довідники	education_levels	Освітні рівні (бакалавр, магістр тощо)
Довідники	course_types	Конфігурація курсу під конкретний тип практики
Зв'язкові	course_module	Зв'язок «курс — модуль» (M:N)
Зв'язкові	module_task	Зв'язок «модуль — завдання» (M:N)
Зв'язкові	task_question	Зв'язок «завдання — питання» (M:N)
Зв'язкові	education_level_course_type	Зв'язок «освітній рівень — тип курсу» (M:N)
Зв'язкові	profession_course_type	Зв'язок «спеціальність — тип курсу» (M:N)

Розглянемо детальніше кожну групу таблиць.

Група «Користувачі» містить таблицю `students`, яка є основним реєстром студентів платформи. У ній зберігаються прізвище, ім'я та по батькові, адреса електронної пошти, номер телефону (обидва поля є унікальними та необов'язковими — студент може зареєструватись або за email, або за номером телефону), дата народження, захешований (зашифрований) пароль та UUID для публічних операцій. Таблиця `admins` має схожу структуру, але додатково містить поле `username` (унікальне), прапорець `is_superuser` та необов'язковий ідентифікатор Telegram.

Таблиця `student_profiles` реалізує концепцію «профілю практики»: один студент може мати кілька профілів — по одному для кожного разу, коли він проходить практику (з різним роком навчання, видом практики чи спеціальністю). Кожен профіль пов'язаний із довідниками `institutions`, `professions`, `practice_types` та `education_levels` через зовнішні ключі.

Група «Навчальний контент» побудована за ієрархічним принципом. Таблиця `courses` зберігає загальні параметри курсу: назву, опис, мінімальний відсоток виконання для завершення (`complete_percentage`) та оцінений час проходження в годинах. Таблиця `modules` описує структурні

розділи курсу та містить поле `order` для визначення порядку відображення; зображення-обкладинка модуля зберігається у файловій системі, а в БД зберігається лише шлях. Таблиця `tasks` представляє окреме завдання: поле `task_type` (тип `enum`) визначає, чи є завдання відеолекцією, чи тестом; поля `content` (текстовий матеріал) та `topic_video_url` (посилання на відео) є необов'язковими. Таблиця `questions` описує тестове питання, яке може бути з одним або кількома правильними варіантами відповіді (`multiple_choice`); зображення до питання також необов'язкове. Таблиця `choices` зберігає варіанти відповіді з прапорцем правильності.

Зв'язки між таблицями контенту реалізовано через проміжні таблиці `course_module`, `module_task` та `task_question` за схемою «багато до багатьох», що дозволяє повторно використовувати один і той самий модуль у різних курсах або одне питання — у різних тестах.

Група «Відстеження прогресу» є найважливішою з точки зору бізнес-логіки. Таблиця `course_student` фіксує зарахування студента (через профіль) на курс і зберігає посилання на відповідний тип курсу. Поля `available` та `completed` дозволяють адміністратору призупиняти доступ до курсу або фіксувати його завершення. Таблиця `course_module_student` відстежує стан кожного модуля для конкретного студента: поле `points` зберігає сукупний бал за всі завдання модуля, `available` визначає, чи розблоковано модуль для студента. Таблиця `module_student_task` зберігає стан окремого завдання: лічильник спроб (`try_counter`), прапорці виконання та доступності. Таблиця `student_task_question` є детальним журналом відповідей: для кожного питання зафіксовано набрані бали, максимальний бал і правильність відповіді.

Така чотирирівнева структура відстеження (курс → модуль → завдання → питання) дозволяє реалізувати механізм послідовного розблокування контенту: наступний модуль стає доступним лише після

успішного завершення попереднього; до завдань у модулі застосовується аналогічне правило.

Сервісні таблиці забезпечують роботу додаткових функцій. Таблиця `certificates` зберігає запис про виданий PDF-сертифікат: UUID є унікальним ідентифікатором для публічної верифікації (будь-яка особа може перевірити справжність сертифіката за URL), `file_name` — унікальне ім'я файлу у сховищі, `url` — повна адреса для завантаження. Таблиця `notifications` є журналом внутрішньосистемних сповіщень із прапорцем прочитання. Таблиця `student_telegrams` зберігає зв'язку між обліковим записом студента та його Telegram `chat_id` (тип `BigInteger`); наявність цього запису свідчить про те, що студент підключив Telegram-сповіщення. Таблиця `telegram_messages` архівує листування через бот. Таблиці `reminder_settings` та `reminder_students` реалізують систему нагадувань: перша описує шаблон нагадування (інтервал у годинах, текст повідомлення), друга — реєструє конкретних студентів на отримання нагадувань із фіксацією часу останнього надсилання (`last_reminded`).

Довідникові таблиці (`institutions`, `professions`, `practice_types`, `education_levels`) мають спрощену структуру: унікальне текстове ім'я та автоматично генерований первинний ключ. Вони заповнюються адміністратором і використовуються при реєстрації студентів та налаштуванні курсів. Таблиця `course_types` є конфігуратором автоматичного зарахування: вона зв'язує курс із типом практики, переліком допустимих спеціальностей, освітніх рівнів і років навчання. Завдяки цьому система може автоматично призначати студентові відповідний курс на основі його профілю.

Архітектурні рішення при побудові схеми

При розробці схеми бази даних прийнято ряд архітектурних рішень, що підвищують цілісність даних, продуктивність запитів та рівень безпеки.

Базовий клас-модель. Усі 28 таблиць успадковують абстрактну модель `BaseModel`, яка визначає три поля, спільні для кожної сутності: `id` (первинний ключ типу `BigInteger` з автоінкрементом та індексом), `created_at` та `updated_at` (мітки часу у часовому поясі UTC, що встановлюються автоматично сервером бази даних через `server_default`). Таке успадкування усуває дублювання коду та гарантує єдину конвенцію ідентифікації записів у всій схемі.

UUID-ідентифікатори. У таблицях `students` та `certificates` поряд із внутрішнім автоінкрементним `id` передбачено поле `uuid` типу `UUID`, що заповнюється функцією `gen_random_uuid()` PostgreSQL. UUID використовується лише у зовнішніх контекстах: URL для верифікації сертифіката та ідентифікація студента в публічних операціях, де небажано розкривати послідовний числовий ідентифікатор. Поле `uuid` оголошено унікальним і проіндексованим для ефективного пошуку.

Шифрування чутливих полів. Паролі студентів і адміністраторів зберігаються у зашифрованому вигляді через кастомний тип `encrypted_str`, реалізований за допомогою `StringEncryptedType` бібліотеки sqlalchemy-utils з алгоритмом AES (режим PKCS5). Ключ шифрування зчитується з конфігурації (`SECRET_KEY`) і не фіксується у коді. Таке рішення є усвідомленим архітектурним вибором для корпоративної системи з централізованим управлінням обліковими записами.

Складені індекси. Для оптимізації найбільш частих запитів визначено три складені індекси:

– `ix_choice_search (question_id, is_correct)` у таблиці `choices` — прискорює вибірку правильних варіантів відповіді для конкретного питання при автоматичній перевірці тесту;

– `ix_mod_stud_task_search (module_student_id, available, completed)` у таблиці `module_student_task` — оптимізує фільтрацію завдань за станом (доступні, незавершені) для відображення прогресу студента;

– `ix_stq_task_question (student_task_id, question_id)` у таблиці `student_task_question` — прискорює пошук збережених відповідей при завантаженні детальної статистики.

Каскадне видалення. Для підтримки цілісності даних при видаленні записів широко застосовується механізм каскадного видалення (`ON DELETE CASCADE`). Ланцюжок каскадів побудований відповідно до семантичної ієрархії: видалення облікового запису студента автоматично знищує всі його профілі, видалення профілю — запис про зарахування на курс і сертифікат, видалення зарахування — записи про прогрес у модулях, видалення прогресу в модулі — стани завдань, видалення стану завдання — збережені відповіді. Аналогічно, видалення навчального елемента (питання, завдання, модуля) каскадно поширюється на пов'язані записи прогресу. Така архітектура виключає «осиротілі» записи та спрощує операції очищення даних.

Еволюція схеми через міграції. Усі зміни схеми бази даних виконуються виключно через міграції Alembic, кожна з яких описує атомарну зміну DDL та містить функції `upgrade()` і `downgrade()`. За час розробки платформи AMP створено 38 міграцій — від початкового ініціалізаційного скрипта до дрібних коригувань полів і додавання нових індексів. Використання міграцій забезпечує відтворюваність схеми в будь-якому середовищі (розробницькому, тестовому, виробничому) і дозволяє відкочувати зміни за потреби.

Таким чином, спроєктована схема бази даних забезпечує надійне сховище для всіх функцій платформи: ієрархічна структура контенту та прогресу відображає реальний процес проходження практики, складені індекси оптимізують критичні запити, а механізми UUID та шифрування підвищують рівень безпеки системи.

2.3 Проєктування API та інтерфейсу

Взаємодія між клієнтською та серверною частинами платформи AMP побудована за архітектурним стилем REST (Representational State Transfer). Сервер надає програмний інтерфейс прикладного програмування (API), який клієнтський застосунок викликає через протокол HTTP, обмінюючись даними у форматі JSON. Такий підхід забезпечує слабку зв'язаність компонентів: серверна частина не зберігає стан сеансу клієнта, а кожний запит містить усю інформацію, необхідну для його обробки, що відповідає принципу відсутності стану (stateless).

Проєктування REST API

Усі кінцеві точки програмного інтерфейсу мають спільний префікс /api/v1, що визначає версію API та дозволяє у майбутньому розвивати інтерфейс без порушення сумісності з наявними клієнтами. Запити та відповіді структуровано відповідно до принципів RESTful: ресурси визначаються URL-адресою, а характер дії над ресурсом задається HTTP-методом (GET — отримання даних, POST — створення або виконання операції, PATCH — часткове оновлення). Стан результату обробки повертається стандартним кодом стану HTTP (200, 201, 400, 401, 404 тощо).

Кінцеві точки логічно згруповано за функціональними доменами (модулями), кожному з яких відповідає окремий маршрутизатор (router) у термінах фреймворку FastAPI: автентифікація (auth), студент (student), профіль студента (student-profile), куратор (teacher), навчальні заклади (institution), сертифікати (certificate), сповіщення (notification), Telegram, файли (files). Документація API генерується автоматично у форматі OpenAPI (Swagger) на основі анотацій типів та схем Pydantic, що спрощує тестування та інтеграцію.

Авторизація запитів реалізована за технологією JSON Web Token (JWT). Після успішного входу (POST /api/v1/auth/login/) сервер повертає

підписаний токен доступу з терміном дії 1 доба. Клієнт зберігає токен і додає його у заголовок Authorization кожного захищеного запиту; сервер перевіряє підпис і термін дії токена та визначає особу користувача без звернення до сховища сеансів. Для відновлення доступу передбачено окремий короткоживучий токен скидання пароля з терміном дії 15 хвилин, що надсилається на електронну пошту користувача.

Перелік основних кінцевих точок API згруповано за модулями та наведено у таблиці 2.2.

Таблиця 2.2 – Перелік основних кінцевих точок REST API платформи AMP

Модуль	**Метод і шлях**	**Призначення**
Auth	POST /auth/sign_up/	Реєстрація студента
Auth	POST /auth/sign_up/admin/	Реєстрація адміністратора
Auth	POST /auth/login/	Автентифікація, видача JWT
Auth	POST /auth/password-reset/request/	Запит на скидання пароля
Auth	POST /auth/password-reset/new-password/	Встановлення нового пароля
Student	GET /student/me	Дані поточного користувача
Student-profile	POST /student-profile/create	Створення профілю проходження практики
Student-profile	GET /student-profile/{id}/course	Отримання курсу профілю
Student-profile	GET /student-profile/{id}/course/module/{mid}	Отримання модуля курсу
Student-profile	POST /student-profile/{id}/course/update-availability	Оновлення доступності контенту
Student-profile	POST /student-profile/{id}/course/module/{mid}/task/{tid}	Здача завдання (тесту)
Student-profile	GET /student-profile/{id}/statistic	Статистика проходження курсу

Student-profile	GET /student-profile/{id}/statistic/module/{mid}/task/{tid}	Статистика виконання завдання
Teacher	GET /teacher/student-profiles/list	Перелік профілів студентів
Teacher	POST /teacher/student-profiles/stats	Зведена статистика за профілями
Teacher	POST /teacher/student-profiles/stats/generate-excel-file	Експорт статистики у файл Excel
Institution	GET /institution/list	Перелік навчальних закладів
Institution	GET /institution/profession/list	Перелік професій
Institution	GET /institution/practice_type/list	Перелік видів практики
Institution	GET /institution/education_level/list	Перелік освітніх рівнів
Certificate	GET /certificate/validate/{uuid}	Верифікація сертифіката за UUID
Notification	GET /notification/logs	Перелік сповіщень користувача
Notification	PATCH /notification/is_read	Позначення сповіщень прочитаними
Telegram	GET /telegram/connect-link	Посилання прив’язки Telegram
File	POST /files/upload	Завантаження файлу
File	GET /files/{filename}	Отримання завантаженого файлу
File	GET /files/static/certificates/{filename}	Отримання файлу сертифіката
Base	GET /version	Версія програмного забезпечення

Структура переданих і отриманих даних формалізована за допомогою схем (DTO — Data Transfer Object) на базі бібліотеки Pydantic, розміщених у каталозі schemas. Кожна схема визначає набір полів, їхні типи та правила валідації, завдяки чому некоректні запити відхиляються ще до досягнення бізнес-логіки із поверненням зрозумілого повідомлення про помилку. Окремо винесено схему посторінкового виведення (pagination), що

застосовується до переліків значного обсягу — наприклад, списку профілів студентів для куратора.

Проєктування інтерфейсу користувача

Клієнтська частина платформи реалізована як односторінковий застосунок (SPA — Single Page Application): браузер один раз завантажує застосунок, після чого переходи між сторінками відбуваються без перезавантаження, а дані динамічно підвантажуються через REST API. Це забезпечує плавну взаємодію, наближену до настільних застосунків, та знижує навантаження на сервер.

Навігаційну структуру застосунку (карту сторінок, sitemap) утворюють дванадцять екранів, кожному з яких відповідає окремий маршрут клієнтського маршрутизатора. Карту навігації наведено у таблиці 2.3.

Таблиця 2.3 – Карта навігації односторінкового застосунку платформи AMP

Сторінка (компонент)	**Маршрут**	**Призначення**	**Доступ**
HomeView	/	Головна сторінка, вхід у систему	публічний
SignupView	/signup	Реєстрація студента	публічний
ForgotPasswordView	/forgot-password	Запит на відновлення пароля	публічний
ResetPasswordView	/reset-password	Встановлення нового пароля за токеном	публічний
ChangePasswordView	/change-password	Зміна пароля	публічний
CreateProfile	/create-profile	Створення профілю практики	автентифікований
StudentProfileView	/profile	Перегляд та редагування профілю	автентифікований

StudentCourseView	/course	Перегляд курсу та модулів	автентифікований
StudentModuleView	/course/:profileId/module/:moduleId	Проходження модуля та тестування	автентифікований
TeacherStatsView	/teacher-stats	Аналітичний дашборд куратора	куратор
CertificateValidationView	/certificate-validation	Публічна верифікація сертифіката	публічний
AboutProjectView	/project	Інформація про платформу	публічний

Доступ до сторінок, що вимагають автентифікації, обмежено за допомогою навігаційної охорони (route guard): перед кожним переходом проміжний обробник перевіряє ознаку requiresAuth маршруту та наявність дійсного токена у сховищі стану і за відсутності автентифікації перенаправляє користувача на головну сторінку. Для скорочення обсягу початкового завантаження компоненти сторінок підключаються через відкладене завантаження (lazy loading) — код кожної сторінки вивантажується окремим фрагментом лише за першого переходу на неї.

Компонентна архітектура клієнтської частини

Клієнтський застосунок побудовано на фреймворку Vue 3 із використанням Composition API; збирання та локальний сервер розробки забезпечує інструмент Vite. Архітектуру організовано за трьома основними рівнями: рівень подання (компоненти та сторінки), рівень керування станом (сховища) та рівень доступу до даних (HTTP-клієнт і модулі API).

Інтерфейс декомпозовано на близько двадцяти п’яти багаторазово використовуваних компонентів (каталог components) — таких як поля введення, селектори, кнопки, таблиці, індикатор прогресу, картки модулів, тестове завдання, центр сповіщень. Сторінки (каталог views) компонують ці елементи у завершені екрани відповідно до карти навігації.

Керування глобальним станом застосунку реалізовано за допомогою бібліотеки Pinia. Сховище auth зберігає токен доступу, відомості про користувача та обраний профіль практики, надаючи похідні значення (getters) — зокрема ознаку автентифікованості та доступність сертифіката; токен зберігається у локальному сховищі браузера (localStorage), що дозволяє відновити сеанс після перезавантаження сторінки. Сховище notification відповідає за стан сповіщень користувача.

Звернення до серверного API інкапсульовано в окремому модулі на базі бібліотеки Axios. Для HTTP-клієнта налаштовано базову URL-адресу та перехоплювачі (interceptors): перехоплювач запиту автоматично додає токен авторизації із сховища auth до заголовка кожного захищеного звернення, а перехоплювач відповіді централізовано обробляє результати та помилки. Виклики API згруповано за доменами у відповідні модулі (автентифікація, студент, профіль, куратор, навчальні заклади, сертифікати, статистика, Telegram), що відображає структуру серверних маршрутизаторів і спрощує супровід коду.

Спроєктований програмний інтерфейс та архітектура клієнтської частини утворюють завершену основу для реалізації платформи, особливості якої розглянуто у третьому розділі.

Висновки до розділу 2

У другому розділі спроєктовано програмну систему платформи АМР на основі вимог, визначених у першому розділі. Обґрунтовано вибір трирівневої клієнт-серверної архітектури з декомпозицією серверної частини на чотири незалежні служби (REST API, Telegram-бот, планувальник та обробник подій), що поєднує поділ відповідальності з простотою розгортання, а статичну структуру й розподіл складників системи відображено на діаграмах компонентів (рис. 2.1) та розгортання (рис. 2.2). Спроєктовано схему бази даних із 28 таблиць під керуванням

PostgreSQL 17 (рис. 2.3, табл. 2.1), у якій ієрархічна структура контенту та прогресу відтворює реальний процес проходження практики, а механізми UUID-ідентифікації, шифрування чутливих полів, складених індексів і каскадного видалення забезпечують безпеку, продуктивність та цілісність даних. Розроблено REST API з єдиним префіксом /api/v1 і JWT-авторизацією (табл. 2.2), а також компонентну архітектуру односторінкового застосунку на Vue 3 із картою навігації з дванадцяти екранів (табл. 2.3). Отримані проєктні рішення є безпосередньою основою для програмної реалізації платформи, якій присвячено наступний розділ.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Аналіз та обґрунтування обраних програмних засобів

Вибір програмних засобів для реалізації платформи AMP здійснювався з урахуванням функціональних і нефункціональних вимог, сформульованих у підрозділі 1.3, та архітектурних рішень, прийнятих у розділі 2. Ключовими критеріями добору були: підтримка асинхронної обробки запитів задля досягнення цільового часу відгуку (≤500 мс), зрілість і стабільність екосистеми, наявність документації, відкритий програмний код (open source) та відсутність ліцензійних відрахувань, а також сумісність компонентів між собою. Перевагу надавали технологіям, що мають широке промислове застосування та активну спільноту розробників, оскільки це знижує ризики супроводу системи у довгостроковій перспективі.

Програмну систему реалізовано за клієнт-серверною архітектурою, тому набір засобів природно поділяється на дві групи: технології серверної частини, що відповідають за бізнес-логіку, роботу з даними та інтеграції, і технології клієнтської частини, що забезпечують взаємодію з користувачем у вебпереглядачі. Розглянемо кожну з груп окремо.

Програмні засоби серверної частини

Серверну частину реалізовано мовою програмування Python — інтерпретованою високорівневою мовою із суворою динамічною типізацією. Python обрано через високу швидкість розробки, лаконічний синтаксис та одну з найбагатших серед сучасних мов екосистему бібліотек для веброзробки, роботи з базами даних, обробки PDF-документів та інтеграції зі сторонніми сервісами. Окрему роль відіграє підтримка асинхронного програмування (async/await), що дозволяє ефективно обслуговувати численні одночасні мережеві запити без блокування потоків виконання.

Основою серверної частини є вебфреймворк FastAPI, побудований на специфікації ASGI та бібліотеках Starlette (вебрівень) і Pydantic (валідація даних). FastAPI обрано завдяки нативній підтримці асинхронності, автоматичній генерації документації API у форматі OpenAPI (Swagger) на основі анотацій типів, а також високій продуктивності, співмірній із фреймворками компільованих мов. Доступ до бази даних реалізовано через об'єктно-реляційне відображення (ORM) SQLAlchemy 2.0 у поєднанні з асинхронним драйвером asyncpg, що дає змогу описувати схему даних декларативно у вигляді класів Python та виконувати запити у неблокувальному режимі. Версіонування схеми бази даних забезпечує інструмент Alembic, за допомогою якого створено 38 міграцій, що документують еволюцію структури сховища.

Перелік основних програмних засобів серверної частини з версіями та обґрунтуванням наведено у таблиці 3.1.

Таблиця 3.1 – Програмні засоби серверної частини платформи AMP

Технологія	**Версія**	**Призначення та обґрунтування вибору**
Python	3.12.4	Основна мова серверної частини; висока швидкість розробки, багата екосистема, підтримка асинхронності
FastAPI	0.115.8	Асинхронний вебфреймворк; автогенерація OpenAPI, висока продуктивність, інтеграція з Pydantic
Starlette	0.45.3	ASGI-основа FastAPI; маршрутизація, проміжне програмне забезпечення (middleware)
Pydantic	2.10.6	Валідація та серіалізація даних; декларативні схеми запитів і відповідей (DTO)
SQLAlchemy	2.0.38	Асинхронний ORM; декларативні моделі, типобезпечні запити до бази даних
asyncpg	0.30.0	Високопродуктивний асинхронний драйвер PostgreSQL для SQLAlchemy

PostgreSQL	17	Реляційна СУБД; надійність, підтримка UUID, JSON, розширень та транзакцій
Alembic	1.14.1	Версіонування схеми бази даних; 38 інкрементних міграцій
Redis	5.2.1	Сховище «ключ–значення» у пам'яті; кешування та обмін повідомленнями (Pub/Sub) між сервісами
APScheduler	3.11.0	Планувальник задач; виконання cron-завдань розсилки нагадувань
ReportLab	4.4.0	Програмна генерація PDF-сертифікатів про проходження практики
pyTelegramBotAPI	4.26.0	Реалізація Telegram-бота для надсилання сповіщень студентам
PyJWT	2.10.1	Формування та перевірка підписаних токенів доступу JWT
cryptography	44.0.1	Криптографічні примітиви; шифрування конфіденційних полів бази даних
aiosmtplib	4.0.1	Асинхронне надсилання повідомлень електронної пошти (SMTP)
openpyxl	3.1.2	Формування аналітичних звітів куратора у форматі Excel (XLSX)
boto3	1.37.21	Взаємодія з об'єктним сховищем (S3-сумісним) для зберігання файлів
SQLAdmin	0.20.1	Адміністративна панель для керування навчальним контентом
Uvicorn	0.35.0	ASGI-сервер для запуску застосунку FastAPI
Docker, Supervisor	—	Контейнеризація та керування 4 процесами (API, бот, планувальник, обробник подій) в одному контейнері

Як сховище даних обрано об'єктно-реляційну систему керування базами даних PostgreSQL версії 17. Її перевагами є висока надійність, відповідність вимогам ACID, вбудована підтримка типу UUID та функції gen_random_uuid() для генерації унікальних ідентифікаторів, а також розвинений механізм індексів та зовнішніх ключів із каскадними операціями. Допоміжне сховище Redis застосовується одночасно у двох ролях: як кеш для часто запитуваних даних і як механізм обміну повідомленнями (Pub/Sub) між незалежними процесами серверної частини.

Окрему групу засобів становлять бібліотеки інтеграцій та фонової обробки. Планувальник APScheduler виконує періодичні завдання розсилки нагадувань; бібліотека ReportLab формує PDF-сертифікати з унікальним ідентифікатором для подальшої верифікації; pyTelegramBotAPI забезпечує роботу Telegram-бота; aiosmtplib відповідає за надсилання повідомлень електронною поштою, а openpyxl — за вивантаження аналітичних звітів куратора у форматі Excel. Контейнеризацію виконано засобами Docker, а керування чотирма процесами серверної частини в межах єдиного контейнера покладено на Supervisor.

Програмні засоби клієнтської частини

Клієнтську частину реалізовано як односторінковий застосунок (Single Page Application, SPA) на основі прогресивного фреймворку Vue.js версії 3. Vue обрано через невисокий поріг входження, реактивну систему оновлення інтерфейсу та підхід Composition API, що дозволяє структурувати логіку компонентів за функціональним призначенням. Складання та локальну розробку забезпечує інструмент Vite, який завдяки використанню нативних ES-модулів надає миттєве гаряче перезавантаження модулів (HMR) та швидку оптимізовану збірку для промислового середовища.

Маршрутизацію між сторінками застосунку реалізовано бібліотекою Vue Router із застосуванням навігаційних охоронців (route guards) для розмежування доступу за ролями користувачів. Централізоване керування станом застосунку (зокрема стан автентифікації та сповіщень) забезпечує бібліотека Pinia. Обмін даними із серверною частиною здійснюється HTTP-клієнтом Axios, для якого налаштовано перехоплювачі запитів і відповідей (interceptors): вони автоматично додають токен доступу JWT до заголовків та обробляють відповіді про закінчення терміну авторизації. Перелік основних програмних засобів клієнтської частини наведено у таблиці 3.2.

Таблиця 3.2 – Програмні засоби клієнтської частини платформи AMP

Технологія	Версія	Призначення та обґрунтування вибору
Vue.js	3.5.13	Прогресивний фреймворк для побудови інтерфейсу; Composition API, реактивність
Vite	6.0.11	Інструмент складання та локальної розробки; швидка збірка, гаряче перезавантаження (HMR)
Vue Router	4.5.0	Клієнтська маршрутизація SPA; навігаційні охоронці для розмежування доступу
Pinia	2.3.1	Централізоване керування станом застосунку (state management)
Axios	1.7.9	HTTP-клієнт; перехоплювачі для автоматичного додавання токена JWT
VueUse	12.5.0	Набір композиційних утиліт для типових задач інтерфейсу
Vue Motion	2.2.6	Декларативні анімації та переходи елементів інтерфейсу
ESLint	9.18.0	Статичний аналіз коду; контроль якості та єдиного стилю
Prettier	3.4.2	Автоматичне форматування вихідного коду
Node.js	18	Середовище виконання інструментів складання клієнтської частини

Дібраний комплекс програмних засобів є цілісним і взаємоузгодженим: усі компоненти належать до зрілих відкритих екосистем, активно підтримуються та широко застосовуються у промисловій веброзробці. Серверні технології Python і FastAPI у поєднанні з асинхронним доступом до PostgreSQL забезпечують виконання нефункціональних вимог щодо продуктивності, а клієнтський стек Vue 3, Vite та Pinia — побудову адаптивного інтерфейсу з високою чутливістю. Спільним для обох частин є використання токенів JWT для авторизації та формату JSON для обміну даними, що мінімізує накладні витрати на інтеграцію клієнта й сервера.

3.2 Особливості реалізації серверної частини

Серверну частину платформи AMP реалізовано мовою Python 3.12.4 на основі асинхронного вебфреймворку FastAPI відповідно до архітектурних рішень, прийнятих у розділі 2. У цьому підрозділі розглянуто

організацію вихідного коду проєкту та найважливіші з погляду бізнес-логіки механізми: авторизацію на основі JWT, послідовне розблокування навчального контенту, обмеження кількості спроб проходження завдань, генерацію сертифікатів у форматі PDF та автоматичну розсилку нагадувань. Для ілюстрації наведено ключові фрагменти вихідного коду.

Структура проєкту серверної частини

Вихідний код серверної частини організовано за принципом поділу відповідальності: кожний каталог об'єднує модулі зі спорідненим призначенням. Така структура спрощує супровід системи та повторне використання коду між чотирма процесами застосунку (REST API, Telegram-бот, планувальник, обробник подій), що описані у підрозділі 2.1. Загальну організацію каталогу `src/` наведено нижче.

```
src/
  main_api.py        точка входу REST API (FastAPI)
  main_bot.py        точка входу Telegram-бота
  main_scheduler.py  точка входу планувальника (APScheduler)
  main_pubsub.py     точка входу обробника подій (Redis
Pub/Sub)
  api/        роутери, залежності (depends), модель-менеджери
  database/   ORM-сутності, менеджери, переліки (enums), сесії
  schemas/    Pydantic-схеми запитів і відповідей (DTO)
  common/     сервіси бізнес-логіки, клієнти (Redis), пошта
  scheduler/  визначення та конфігурація запланованих задач
  bot/        обробники, клавіатури та сховище Telegram-бота
  admin/      адміністративна панель (SQLAdmin)
  configs/    налаштування за середовищами (base/dev/prod)
```

Бізнес-логіка винесена із роутерів у сервісні класи каталогу `common/services`, що дає змогу повторно використовувати її як в обробниках HTTP-запитів, так і у фонових задачах планувальника. Доступ до бази даних інкапсульовано у класах-менеджерах (каталог `database/entities`), які надають типізовані методи вибірки та оновлення сутностей і приховують деталі формування SQL-запитів засобами SQLAlchemy.

Реалізація авторизації на основі JWT

Автентифікацію користувачів реалізовано за допомогою токенів JWT (JSON Web Token), що відповідає принципу відсутності стану (stateless) серверної частини. Після успішного входу клієнт отримує підписаний токен і додає його до заголовка `Authorization` кожного наступного запиту. Формування та перевірка підпису токена зосереджені у двох функціях, наведених у лістингу 3.1: `create_access_token` кодує корисне навантаження разом із часом завершення дії (`exp`) за алгоритмом HS256, а `decode_access_token` перевіряє підпис і повертає розкодовані дані.

```
def create_access_token(
    data: dict, expires_delta: timedelta, token_type: str |
None = None
) -> str:
    to_encode = data.copy()
    expire = datetime.now(UTC) + expires_delta
    extra_data = {"exp": expire}
    if token_type:
        extra_data["token_type"] = token_type
    to_encode.update(extra_data)
    return jwt.encode(
        to_encode, settings.SECRET_KEY,
algorithm=settings.JWT_ENCODE_ALGORITHM
    )
```

Платформа використовує два типи токенів, що різняться полем `token_type` та терміном дії: токен доступу `student-token` із терміном дії одна доба, який видається після реєстрації та входу, і токен `password-reset-token` із терміном дії 15 хвилин, що супроводжує процедуру відновлення пароля. Перевірку токена для захищених кінцевих точок винесено у залежність (dependency) FastAPI `authorize_student`: вона отримує токен із заголовка через `APIKeyHeader`, розкодовує його, знаходить відповідного студента у базі даних і повертає помилку HTTP 403, якщо термін дії токена сплив (`jwt.ExpiredSignatureError`) або підпис недійсний (`jwt.InvalidTokenError`). Завдяки механізму впровадження залежностей FastAPI ця перевірка автоматично виконується перед кожним захищеним обробником, а до нього потрапляє вже автентифікований об'єкт студента.

Послідовне розблокування навчального контенту

Однією з ключових вимог до платформи є послідовне проходження навчального матеріалу: студент отримує доступ до наступного модуля лише після завершення попереднього, а в межах модуля — до наступного завдання лише після виконання поточного. Цю логіку реалізовано у сервісі `StudentCourseAvailabilityService`, який після кожного надсилання відповіді перераховує доступність модулів і завдань. Алгоритм розблокування модулів наведено у лістингу 3.2: сервіс у циклі шукає останній доступний незавершений модуль і перший недоступний; якщо доступного модуля ще немає, він відкриває перший, а якщо поточний доступний модуль уже завершено — відкриває наступний.

```
async def _update_module_availability(self) -> None:
    """Make modules available based on completion status."""
    while True:
        uncompleted_available = await
self.s_course_module_manager.last_uncompleted_available(
            self.s_course
        )
        uncompleted_unavailable = (
            await
self.s_course_module_manager.last_uncompleted_unavailable(self
.s_course)
        )
        if not uncompleted_unavailable:
            break
        if not uncompleted_available:
            await
self.s_course_module_manager.toggle_available(uncompleted_unav
ailable)
            continue
        if uncompleted_available.completed:
            await
self.s_course_module_manager.toggle_available(uncompleted_unav
ailable)
        else:
            break
```

Аналогічний алгоритм застосовується до завдань усередині модуля у методі `update_task_availability`. Метод `submit_task` сервісу `StudentCourseService` виконує повний цикл обробки відповіді: перевіряє

відповіді студента, перераховує бали модуля, оновлює статус завершення завдання та викликає перерахунок доступності. Якщо після цього курс позначається як завершений, автоматично запускається генерація сертифіката, описана нижче.

Обмеження кількості спроб проходження завдань

Задля запобігання багаторазовому вгадуванню правильних відповідей у платформі діє обмеження на кількість спроб проходження завдання. Відповідні константи задано у налаштуваннях: `TASK_TRY_LIMIT` = 3 спроби та `TASK_TRY_TIMEOUT_MIN` = 1440 хвилин (24 години). Лічильник спроб зберігається в полі `try_counter` запису завдання студента, а факт блокування — у сховищі Redis із автоматичним терміном дії (TTL), що дорівнює періоду очікування. Метод `check_task_cooldown`, наведений у лістингу 3.3, після вичерпання ліміту спроб встановлює блокування у Redis і обнуляє лічильник.

```
async def check_task_cooldown(self, student_task:
ModuleStudentTaskModel) -> None:
    # Check if try limit has been reached
    if student_task.try_counter >= settings.TASK_TRY_LIMIT:
        # Block task and reset counter
        await task_cooldown_client.block(
            self.s_course.student_profile_id,
            student_task.task_id,
        )
        await self.s_module_task_manager.update(student_task,
try_counter=0)
```

Блокування реалізовано клієнтом `TaskCooldownClient` за допомогою команди Redis `SETEX`, яка записує ключ виду `try-timeout:{profile_id}:{task_id}` із заданим терміном дії; після його завершення ключ автоматично видаляється, і доступ до завдання відновлюється без додаткових дій з боку сервера. Перевірку наявності блокування винесено в окрему залежність FastAPI `task_try_checker`: перед прийманням нової відповіді вона звертається до Redis і, якщо завдання заблоковано, повертає код HTTP 429 (Too Many Requests) разом із кількістю

секунд до розблокування (`cooldown_seconds`). Це значення клієнтська частина використовує для відображення таймера зворотного відліку.

Генерація сертифіката у форматі PDF

Після успішного завершення курсу платформа автоматично формує іменний сертифікат у форматі PDF. Генерацію реалізовано у сервісі `CertificateService` на основі бібліотеки ReportLab та допоміжного класу `PDFWriter`. Сертифікат будується методом накладання: заздалегідь підготовлений PDF-шаблон (`template_certificate.pdf`) зчитується бібліотекою PyPDF2, а поверх нього засобами ReportLab наноситься динамічний текстовий шар — прізвище та ім'я студента, відсоток проходження курсу, його тривалість у годинах, назва, спеціальність, дата видачі та унікальний ідентифікатор. Для забезпечення глобальної унікальності та можливості подальшої верифікації кожному сертифікату присвоюється ідентифікатор UUID (`uuid1`), який також виводиться на документі. Послідовність формування сертифіката за профілем студента наведено у лістингу 3.4.

```
async def generate_certificate_by_student_profile(
    self,
    student_profile: StudentProfileModel,
) -> tuple[CertificatePath, uuid1, FileName]:
    uuid = uuid1()
    file_name = (

f"{student_profile.practice_type.name}__{student_profile.stude
nt.initials}-{uuid}.pdf"
    )
    file_name = file_name.replace(" ", "-")
    student_course = await
student_profile.awaitable_attrs.course
    course_completed_percentage = (
        await
self.calculate_service.calculate_course_completed_percentage(s
tudent_course)
    )
    return (
        await self.generate_certificate(file_name=file_name,
uuid=uuid, ...),
        uuid,
```

```
        file_name,
    )
```

Метод `db_generate_certificate` зберігає згенерований файл та запис про сертифікат у базі даних, попередньо видаляючи попередній сертифікат того самого студента, якщо він існував. На документі друкується посилання на публічну сторінку перевірки (`/certificate-validation`), де за ідентифікатором UUID будь-яка зацікавлена особа (наприклад, роботодавець) може пересвідчитися в автентичності сертифіката через відкриту кінцеву точку `GET /api/v1/certificate/validate/{uuid}`. Це унеможливлює підробку документа без доступу до бази даних платформи.

Реалізація нагадувань та сповіщень

Для своєчасного інформування студентів про наближення кінцевого терміну проходження курсу та про необхідність продовжити навчання у платформі передбачено підсистему запланованих задач на основі бібліотеки APScheduler. Планувальник запускається окремим процесом (`main_scheduler.py`) і реєструє набір задач за конфігурацією `client_manager_tasks`. Усі три задачі — `StudentCourseTimeoutNotification` (нагадування за 15, 10, 3 та 1 день до завершення курсу), `ExpiredCourseValidation` (позначення курсів, термін яких сплив) та `ReminderNotification` (періодичні нагадування за налаштуваннями студента) — виконуються за тригером `cron` щодоби о 00:00. Реєстрацію задач у планувальнику наведено у лістингу 3.5.

```
def init_scheduler_tasks(scheduler: AsyncIOScheduler,
scheduled_tasks: tuple[Task]) -> None:
    for task in scheduled_tasks:
        scheduler.add_job(
            func=task.task_class().execute,
            kwargs=task.task_kwargs,
            trigger=task.scheduler_trigger,
            **task.scheduler_kwargs,
        )
```

Самі задачі не надсилають повідомлення безпосередньо, а делегують це сервісу `StudentNotificationService`, який публікує подію у Redis Pub/Sub.

Окремий процес-обробник подій (`main_pubsub.py`) отримує таку подію і доставляє сповіщення доступними каналами: зберігає його в базі даних для відображення в інтерфейсі, надсилає електронною поштою та, якщо студент під'єднав свій акаунт, дублює повідомленням у Telegram-боті. Такий поділ на публікацію та обробку подій робить розсилку асинхронною й відмовостійкою: тимчасова недоступність поштового сервера чи Telegram API не блокує основну бізнес-логіку застосунку.

3.3 Особливості реалізації клієнтської частини

Клієнтську частину платформи AMP реалізовано у вигляді односторінкового застосунку (SPA) на основі фреймворку Vue 3.5.13 із застосуванням композиційного API (Composition API) та синтаксису `<script setup>`. Збірку проєкту виконує інструмент Vite 6.0.11, керування станом — бібліотека Pinia 2.3.1, маршрутизацію — Vue Router 4, а взаємодію із серверною частиною — HTTP-клієнт Axios. У цьому підрозділі розглянуто організацію вихідного коду клієнтської частини та найважливіші механізми її реалізації: централізовану взаємодію з REST API через перехоплювачі Axios, керування станом автентифікації засобами Pinia, захист маршрутів, інтерактивний навчальний модуль із таймером блокування спроб, аналітичну панель викладача з експортом у формат Excel та публічну сторінку верифікації сертифіката.

Структура проєкту клієнтської частини

Вихідний код клієнтської частини організовано за функціональним призначенням модулів усередині каталогу `src/`. Така структура відокремлює подання (views) від багаторазово використовуваних елементів інтерфейсу (components) і централізує доступ до серверних служб у каталозі `api/`. Загальну організацію наведено нижче.

```
src/
  main.js        точка входу: створення застосунку, реєстрація
Pinia, router
```

```
  App.vue        кореневий компонент
  views/         сторінки-подання (12 екранів SPA)
  components/    багаторазові компоненти інтерфейсу (~25)
  stores/        сховища стану Pinia (auth, notification)
  router/        визначення маршрутів та проміжне ПЗ
(middleware)
  api/           модулі звернень до REST API (за доменами)
  utils/         утиліти: екземпляр Axios, константи, хелпери
  assets/        стилі та статичні ресурси
```

Усі звернення до серверної частини згруповано в каталозі `api/` за предметними доменами (`apiAuth.js`, `apiStudentProfile.js`, `apiTeacher.js`, `apiCertificate.js` тощо). Кожний такий модуль експортує функції, що формують HTTP-запит через єдиний попередньо налаштований екземпляр Axios, описаний нижче. Завдяки цьому подання не працюють із мережевим рівнем безпосередньо, а викликають іменовані функції-обгортки, що спрощує супровід і повторне використання коду.

Централізована взаємодія з API та перехоплювачі Axios

Усі мережеві запити проходять через єдиний екземпляр Axios, створений у модулі `utils/api.js` із базовою адресою `VITE_APP_API_URL` (зчитується зі змінних середовища Vite) і часом очікування 50 секунд. До цього екземпляра під'єднано перехоплювач запитів (request interceptor), який перед надсиланням кожного запиту звертається до сховища автентифікації та, якщо користувач увійшов у систему, додає токен JWT до заголовка `Authorization`. Це звільняє кожний окремий виклик API від потреби вручну передавати токен. Реалізацію перехоплювача наведено у лістингу 3.6.

```
api.interceptors.request.use(
    (config) => {
        const authStore = useAuthStore();
        if (authStore.isAuthenticated) {
            config.headers.Authorization =
`${authStore.getToken}`;
        }
        return config;
    },
    (error) => Promise.reject(error)
);
```

Окрім перехоплювача запитів, до екземпляра під'єднано й перехоплювач відповідей (response interceptor), що пропускає успішні відповіді та централізовано перенаправляє помилки у проміжок `Promise.reject` для подальшого оброблення у викличному коді. Такий поділ відповідальності забезпечує єдину точку конфігурації мережевого рівня: зміна базової адреси, заголовків чи логіки авторизації виконується в одному місці й автоматично застосовується до всіх ~25 кінцевих точок, описаних у підрозділі 2.3.

Керування станом засобами Pinia

Стан застосунку зосереджено у двох сховищах Pinia. Сховище автентифікації `auth` зберігає токен доступу, відомості про користувача та обраний профіль практики. Токен ініціалізується значенням із локального сховища браузера (`localStorage`), завдяки чому сеанс користувача зберігається між перезавантаженнями сторінки. Сховище надає гетер `isAuthenticated`, що повертає логічну ознаку наявності токена, та набір дій (actions) для входу, виходу та завантаження даних користувача. Ключові елементи сховища наведено у лістингу 3.7.

```
export const useAuthStore = defineStore('auth', {
  state: () => ({
    token: getTokenFromLocalStorage('student-token'),
    userInfo: null,
    profile: null,
  }),
  getters: {
    getToken: (state) => state.token,
    isAuthenticated: (state) => !!state.token,
    certificate: (state) => state.profile?.certificate,
  },
  actions: {
    login(token) {
      this.token = token
      localStorage.setItem('student-token', token)
    },
    logout() {
      this.token = null
      this.userInfo = null
      localStorage.removeItem('student-token')
```

```
    },
    async getUser() {
      if (this.isAuthenticated) {
        const response = await me()
        this.userInfo = response.data
        this.updateProfile()
        return this.userInfo
      }
      return null
    },
  },
})
```

Дія `login` зберігає одержаний від сервера токен як у стані сховища, так і в `localStorage`; дія `logout` очищає їх, завершуючи сеанс. Дія `getUser` звертається до кінцевої точки `GET /api/v1/student/me`, завантажує відомості про користувача та його профілі практики й обирає активний профіль. Друге сховище — `notification` — реалізовано у функціональному стилі (setup-store) і відповідає за відображення спливних повідомлень користувачеві: дія `showNotification` показує переданий текст і автоматично приховує його через п'ять секунд, ігноруючи повторні виклики протягом цього інтервалу, що запобігає накладанню сповіщень.

Маршрутизація та захист сторінок

Навігацію між дванадцятьма екранами SPA забезпечує бібліотека Vue Router із застосуванням лінивого завантаження компонентів (`() => import(...)`), завдяки чому код кожної сторінки завантажується лише за потреби. Захист сторінок, що потребують автентифікації, реалізовано глобальним проміжним обробником (middleware) через хук `beforeEach`: якщо маршрут позначено ознакою `meta.requiresAuth`, а користувач не автентифікований, навігацію перенаправляють на головну сторінку. Сторінки навчального курсу, модуля, профілю та створення профілю захищено саме цим механізмом. Натомість головна сторінка має зворотний охоронець `beforeEnter`, який автоматично перенаправляє вже автентифікованого користувача до сторінки курсу.

Реалізація навчального модуля

Центральним екраном застосунку є подання навчального модуля `StudentModuleView.vue`, у якому студент послідовно опановує теми та проходить тести. Ліворуч розташовано бічну панель зі списком завдань модуля, де стан кожного завдання позначається піктограмою: заблоковане (`LockIcon`), доступне (`CycleIcon`) або завершене (`CompletedIcon`). Доступність завдання визначається полем `available`, яке обчислюється серверною частиною за алгоритмом послідовного розблокування, описаним у підрозділі 3.2; перехід до недоступного завдання заблоковано на рівні інтерфейсу. Усі дані стану компонента оголошено як реактивні посилання (`ref`), що забезпечує автоматичне оновлення інтерфейсу при зміні даних.

Завдання типу `TEST` оброблюються окремо: студент обирає відповіді у компоненті `Test`, які накопичуються у форматі «ідентифікатор питання — масив обраних відповідей» і надсилаються на сервер дією `submitTest`. Після надсилання компонент перезавантажує дані модуля, статистику завдання та відомості користувача, після чого відображає результат — кількість набраних балів із прохідним порогом. У разі успішного проходження запускається анімація святкування (компонент `Fireworks`), а за завершення останнього модуля курсу користувачеві пропонується кнопка завантаження сертифіката. Завдання навчального (нетестового) типу зараховуються автоматично, коли студент прогортає їх вміст до кінця, що відстежує обробник події прокручування `handleScroll`.

Окрему увагу приділено реалізації обмеження кількості спроб. Константа `TRY_LIMIT`, що дорівнює трьом, відображає студентові кількість використаних спроб, а в разі їх вичерпання серверна частина повертає час до розблокування у полі `cooldown_seconds` (механізм блокування на боці сервера описано в підрозділі 3.2). На основі цього значення компонент запускає таймер зворотного відліку, який щосекунди оновлює текстове подання часу, що залишився, у форматі

«години:хвилини:секунди» та автоматично зупиняється після завершення інтервалу. Реалізацію таймера наведено у лістингу 3.8.

```
const addCooldownTimerInterval = (seconds) => {
  const targetTime = Date.now() + seconds * 1000

  if (cooldownTimerData.value.timer) {
    clearInterval(cooldownTimerData.value.timer)
  }

  cooldownTimerData.value.timer = setInterval(() => {
    const remaining = targetTime - Date.now()
    if (remaining <= 0) {
      clearInterval(cooldownTimerData.value.timer)
      cooldownTimerData.value.text = '00:00:00'
      return
    }
    const hrs = Math.floor((remaining / 3600000) %
24).toString().padStart(2, '0')
    const mins = Math.floor((remaining / 60000) %
60).toString().padStart(2, '0')
    const secs = Math.floor((remaining / 1000) %
60).toString().padStart(2, '0')
    cooldownTimerData.value.text = `${hrs}:${mins}:${secs}`
  }, 1000)
}
```

Таймер очищується у хуку життєвого циклу `onUnmounted` разом із відписуванням від подій вікна, що запобігає витокам пам'яті при переході між сторінками. Компонент також адаптується до мобільних пристроїв: за ширини екрана до 768 пікселів бічна панель приховується автоматично.

Аналітична панель викладача та експорт у формат Excel

Подання `TeacherStatsView.vue` надає викладачеві (куратору практики) засоби моніторингу успішності студентів. Робота з панеллю відбувається у два кроки. Спершу куратор обирає критерії вибірки — навчальний заклад, спеціальність, курс та вид практики — за якими кінцева точка `GET /api/v1/teacher/student-profiles/list` повертає перелік відповідних студентів. Після вибору конкретних студентів пакетний запит `POST /api/v1/teacher/student-profiles/stats` повертає їхню статистику, на основі якої динамічно формується таблиця: фіксовані стовпці (прізвище та ініціали,

спеціальність, навчальний заклад, курс) доповнюються стовпцями для кожного модуля з відсотком його завершення, підсумковим відсотком та посиланням на сертифікат.

Для формування звітності передбачено експорт статистики у файл Excel. Кінцева точка `POST /api/v1/teacher/student-profiles/stats/generate-excel-file` повертає бінарний вміст файла, тому відповідний запит виконується з параметром `responseType: 'blob'`. Одержані дані обгортаються в об'єкт `Blob` із MIME-типом електронної таблиці, після чого засобами браузера створюється тимчасове об'єктне посилання (object URL) та програмно ініціюється завантаження файла `students.xlsx`. Реалізацію експорту наведено у лістингу 3.9.

```
async function downloadExcel() {
  const response = await studentsStatsExcel(
    selectedStudentsValues.value.map((student) => student.id),
  )

  const blob = new Blob([response.data], {
    type: 'application/vnd.openxmlformats-
officedocument.spreadsheetml.sheet',
  })

  const url = window.URL.createObjectURL(blob)
  const link = document.createElement('a')
  link.href = url
  link.setAttribute('download', 'students.xlsx')
  document.body.appendChild(link)
  link.click()
  document.body.removeChild(link)
  window.URL.revokeObjectURL(url)
}
```

Після завершення завантаження тимчасове об'єктне посилання звільняється викликом `revokeObjectURL`, що запобігає накопиченню ресурсів у пам'яті браузера. Такий підхід дає змогу завантажувати згенеровані сервером звіти без перезавантаження сторінки та без зберігання проміжних файлів на сервері.

Верифікація сертифіката

Публічну перевірку автентичності сертифіката реалізовано в поданні `CertificateValidationView.vue`, доступ до якого не потребує автентифікації. Користувач (наприклад, представник підприємства-замовника) вводить унікальний номер сертифіката, надрукований на документі, після чого застосунок звертається до відкритої кінцевої точки `GET /api/v1/certificate/validate/{uuid}`. Якщо сервер повертає код стану 200, сертифікат вважається дійсним і користувачеві відображається відповідне підтвердження; в іншому разі — повідомлення про те, що сертифікат не знайдено. Завантаження самого документа реалізовано компонентом `CertificateButton`, який відкриває URL сертифіката в новій вкладці браузера. Такий механізм забезпечує перевірку автентичності виданих платформою сертифікатів без можливості їх підробки, оскільки достовірність гарантується наявністю відповідного запису з ідентифікатором UUID у базі даних платформи.

3.4 Методика роботи користувача

У цьому підрозділі наведено типові сценарії роботи з онлайн-платформою АМР для трьох ролей користувачів — студента, куратора (викладача) та адміністратора. Опис супроводжується копіями екрану діючого програмного забезпечення, що ілюструють ключові кроки кожного сценарію. Послідовність екранів відповідає реальному маршруту користувача, реалізованому засобами Vue Router (карта навігації наведена у підрозділі 2.3).

Сценарій роботи студента

Робота студента з платформою починається з головної сторінки, яка містить стислий опис призначення системи та заклик до реєстрації. Головна сторінка є публічною й доступна без авторизації; з неї користувач переходить до форми створення облікового запису (рис. 3.1).

Рисунок 3.1 – Головна сторінка платформи AMP

Для початку проходження виробничої практики студент реєструється в системі. Форма реєстрації (рис. 3.2) поєднує введення особистих даних (прізвище, ім'я, електронна пошта, номер телефону, пароль) із вибором параметрів навчального профілю — навчального закладу, спеціальності, курсу, рівня освіти та виду практики. Значення для випадних списків завантажуються з довідникових кінцевих точок `/api/v1/institution/*`, що гарантує узгодженість введених даних із наявними в базі сутностями. Саме

за комбінацією цих параметрів система автоматично визначає курс, який має пройти студент.

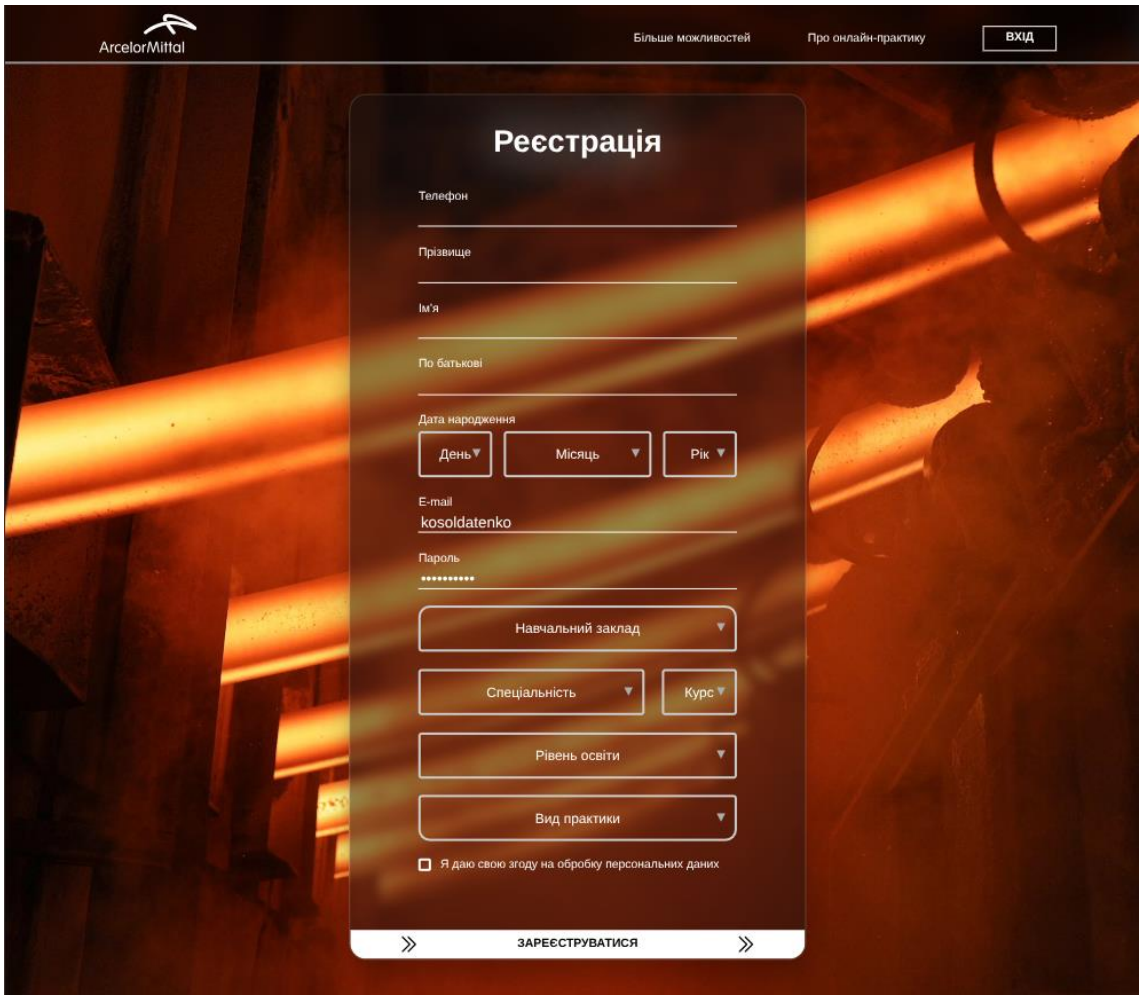

Рисунок 3.2 – Форма реєстрації студента

Після успішної реєстрації та авторизації студент отримує JWT-токен і перенаправляється на сторінку призначеного курсу (рис. 3.3). Сторінка курсу складається з двох частин: ліворуч розташовано перелік навчальних модулів, праворуч — блок «Мій результат» із поточним прогресом та, після завершення курсу, посиланням на згенерований сертифікат. Модулі розблоковуються послідовно: доступним є лише наступний модуль після завершення попереднього, що реалізує контрольований порядок проходження матеріалу.

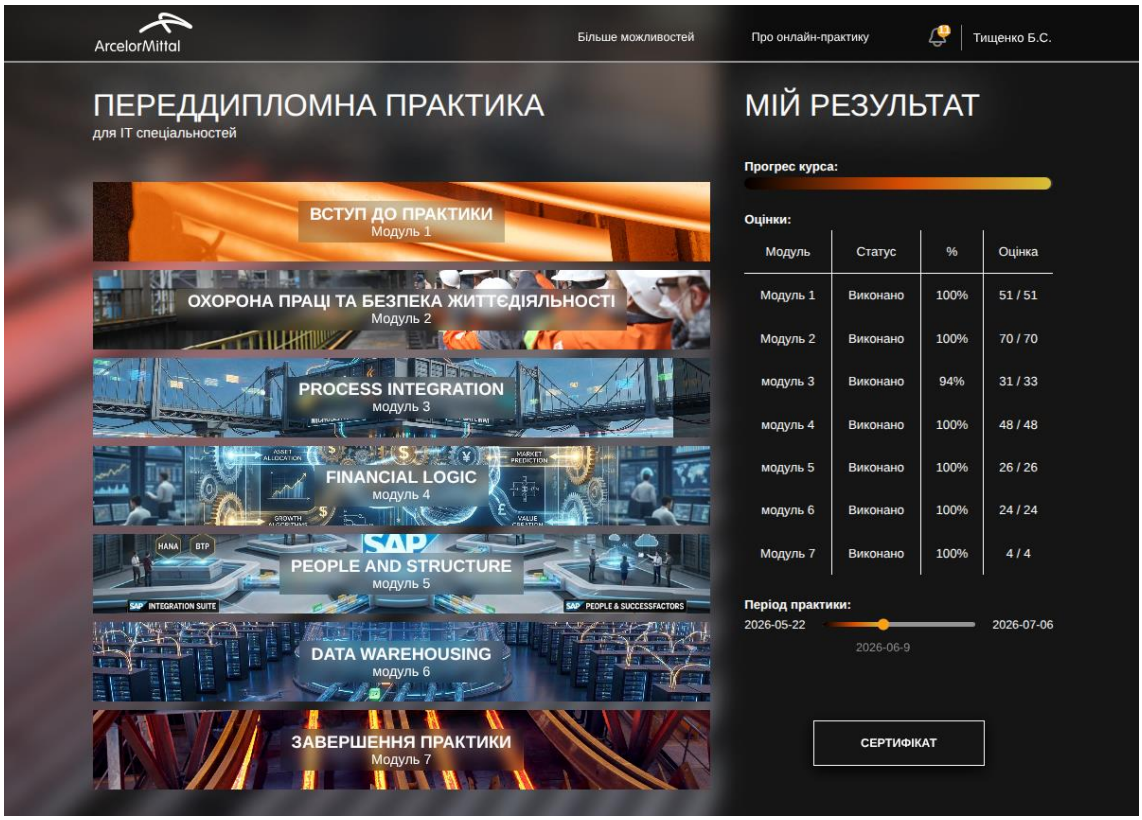

Рисунок 3.3 – Сторінка курсу зі списком модулів

Вибравши доступний модуль, студент переходить до екрана його проходження (рис. 3.4). Інтерфейс цього екрана побудовано навколо бічної панелі зі списком завдань модуля, де кожне завдання позначене піктограмою стану — заблоковане, доступне або виконане. Основна область відображає поточне завдання: навчальний матеріал або тестове запитання з варіантами відповідей. Бічну панель можна приховати, збільшивши робочу область. Відповіді студента надсилаються на кінцеву точку перевірки, а результат повертається миттєво.

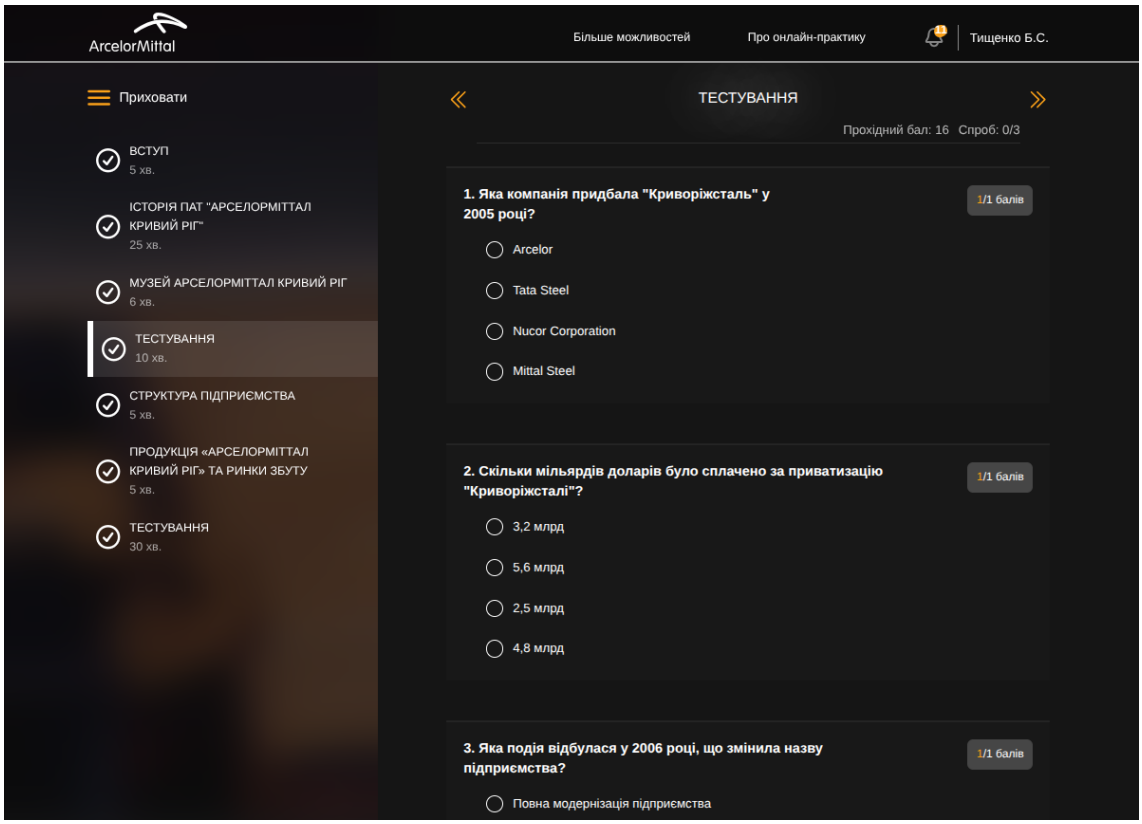

Рисунок 3.4 – Проходження навчального модуля з тестом

Кількість спроб проходження завдання обмежена (константа `TASK_TRY_LIMIT` = 3). У разі вичерпання спроб активується таймер очікування тривалістю `TASK_TRY_TIMEOUT_MIN` = 1440 хвилин (24 години), після чого спроби поновлюються. Це запобігає механічному добиранню правильних відповідей і стимулює свідоме опрацювання матеріалу. Успішне завершення модуля супроводжується анімацією та автоматичним розблокуванням наступного модуля.

Поточні результати студент відстежує в особистому кабінеті (рис. 3.5). Кабінет об'єднує контактні дані, обраний навчальний профіль, зведену статистику проходження курсу та, після його завершення, блок із сертифікатом. Тут само доступні дії зі зміни профілю та виходу із системи. Згенерований сертифікат у форматі PDF містить унікальний ідентифікатор UUID, за яким будь-яка стороння особа може перевірити його справжність.

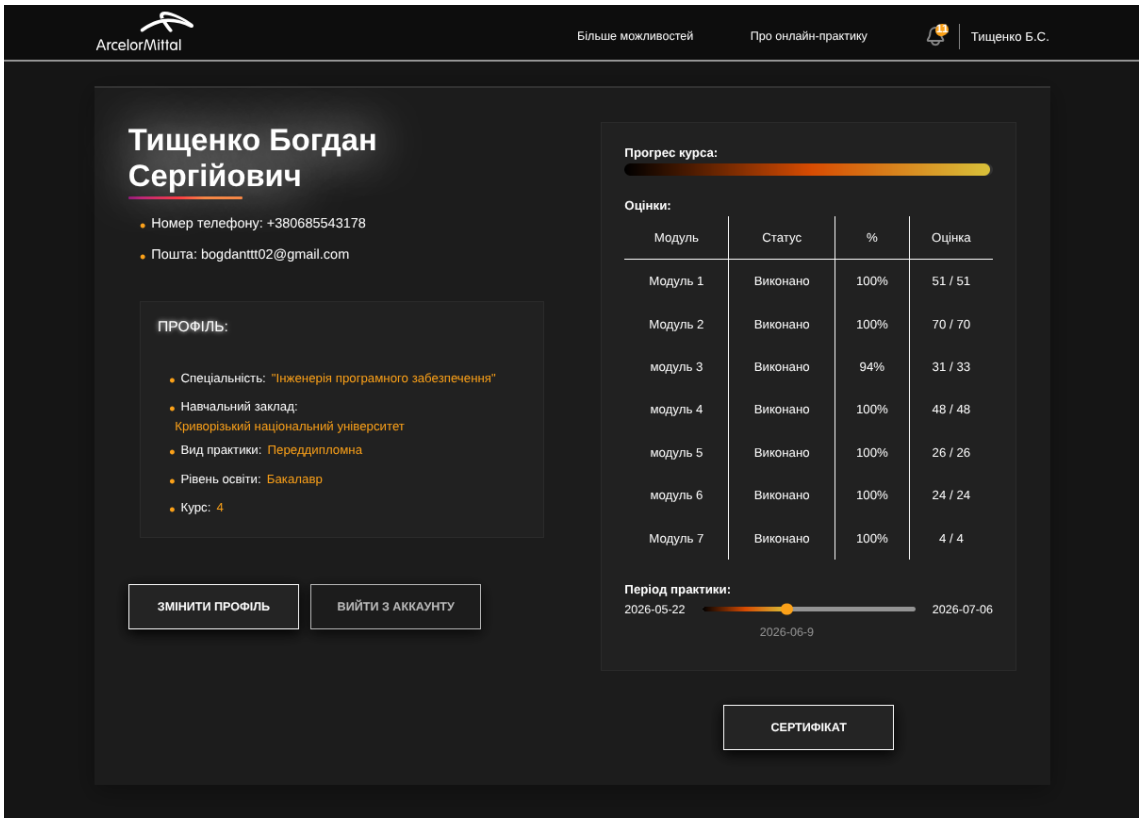

Рисунок 3.5 – Особистий кабінет студента зі статистикою та сертифікатом

Таким чином, для студента платформа реалізує повний замкнений маршрут: реєстрація → вибір профілю → проходження курсу за модулями → тестування → отримання сертифіката. Усі переходи між екранами захищені навігаційними застереженнями (route guards), які не допускають неавторизованого доступу до закритих сторінок.

Сценарій роботи куратора

Куратор (викладач) використовує аналітичний дашборд для контролю проходження практики групами студентів. Робота починається з форми відбору, де куратор задає параметри вибірки — навчальний заклад, спеціальність, курс та вид практики (рис. 3.6). За цими параметрами система формує перелік студентів, що відповідають критеріям, і виводить його у вигляді таблиці з показниками прогресу кожного студента та посиланнями на видані сертифікати.

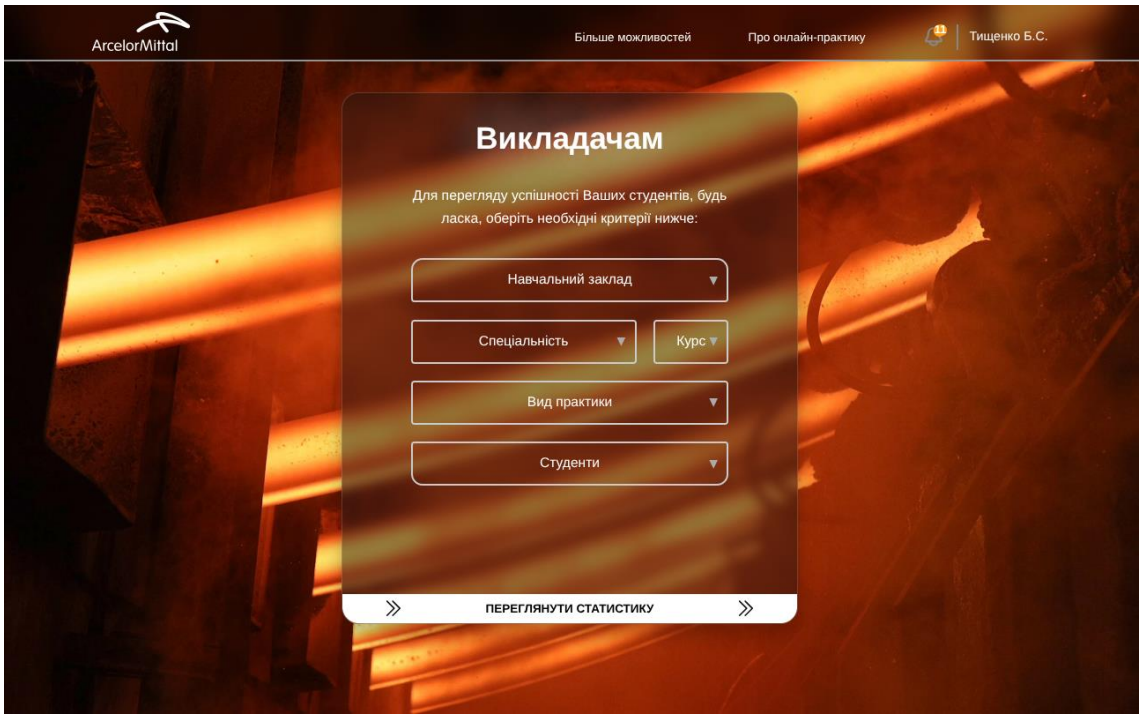

Рисунок 3.6 – Аналітичний дашборд куратора

Дані для таблиці завантажуються одним пакетним запитом до кінцевої точки `/api/v1/teacher/student-profiles/stats`, що мінімізує кількість звернень до сервера навіть для великих груп. Кнопка «Завантажити Excel» ініціює формування звіту у форматі XLSX: сервер віддає файл потоковою відповіддю (streaming response), а браузер зберігає його локально. Сформований звіт придатний для подальшого аналізу та документального оформлення підсумків практики поза межами платформи. Отже, куратор отримує наочний інструмент моніторингу без потреби ручного збирання даних.

Сценарій роботи адміністратора

Адміністратор відповідає за наповнення платформи навчальним контентом — курсами, модулями, завданнями та тестовими запитаннями, а також за супровід довідникових сутностей (навчальні заклади, спеціальності, види практики тощо). Обліковий запис адміністратора створюється через окрему захищену кінцеву точку

`/api/v1/auth/sign_up/admin/`, відокремлену від реєстрації студентів, що розмежовує рівні доступу в системі.

Керування контентом відбувається на рівні моделей бази даних, описаних у підрозділі 2.2: адміністратор формує ієрархію «курс → модуль → завдання → запитання → варіант відповіді» та визначає, для якої комбінації спеціальності й виду практики призначено кожен курс (зв'язки `profession_course_type`, `education_level_course_type`). Завдяки такій структурі один і той самий навчальний матеріал може повторно використовуватися для різних категорій студентів, а зміни в контенті негайно відображаються в інтерфейсі студентів і кураторів без перерозгортання застосунку.

Перевірку справжності сертифіката може виконати будь-який користувач — зокрема представник підприємства-замовника — на публічній сторінці верифікації (рис. 3.7). Достатньо ввести унікальний номер сертифіката (UUID), розміщений у нижній частині PDF-документа, і система звернеться до кінцевої точки `/api/v1/certificate/validate/{uuid}`, повернувши результат перевірки із зазначенням власника та статусу документа.

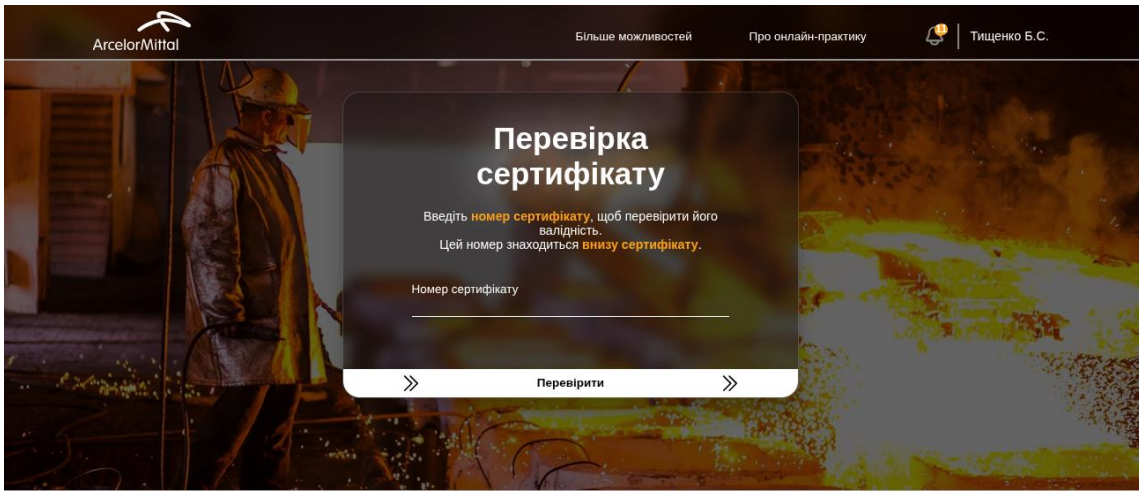

Рисунок 3.7 – Публічна сторінка верифікації сертифіката

Наведені сценарії демонструють, що інтерфейс платформи AMP розмежовує функції за ролями та забезпечує кожну категорію користувачів необхідним набором інструментів — від проходження навчального матеріалу студентом до моніторингу й верифікації результатів куратором та зовнішніми особами.

Висновки до розділу 3

У третьому розділі описано програмну реалізацію платформи AMP на основі проєктних рішень, прийнятих у попередньому розділі. Обґрунтовано вибір конкретних програмних засобів серверної та клієнтської частин із зазначенням версій і призначення кожного (табл. 3.1, 3.2): серверний стек Python 3.12.4 і FastAPI з асинхронним доступом до PostgreSQL 17 забезпечує виконання вимог щодо продуктивності, а клієнтський стек Vue 3, Vite та Pinia — побудову адаптивного односторінкового застосунку. Розглянуто організацію вихідного коду обох частин та ключові механізми реалізації, проілюстровані дев'ятьма лістингами коду: авторизацію на основі JWT, послідовне розблокування навчального контенту, обмеження кількості спроб через Redis, автоматичну генерацію PDF-сертифіката з UUID-верифікацією та підсистему нагадувань на боці сервера; перехоплювачі Axios, керування станом засобами Pinia, інтерактивний навчальний модуль із таймером блокування та експорт аналітики у формат Excel — на боці клієнта. Наведено методику роботи користувачів для трьох ролей — студента, куратора та адміністратора — із копіями екрана діючого програмного забезпечення (рис. 3.1–3.7). Отже, поставлені у першому розділі функціональні та нефункціональні вимоги до платформи повністю реалізовано в робочому програмному продукті.

ВИСНОВКИ

У кваліфікаційній роботі вирішено актуальне завдання автоматизації супроводу проходження виробничої практики студентів на промисловому підприємстві шляхом розроблення спеціалізованої онлайн-платформи AMP. За результатами виконаної роботи можна сформулювати такі висновки.

На основі критичного аналізу наукових джерел та наявних програмних рішень установлено, що поточна організація виробничої практики спирається переважно на ручні засоби (електронні таблиці, листування електронною поштою, паперовий документообіг) і характеризується відсутністю централізованого обліку, автоматичного відстеження прогресу, перевірки знань та своєчасних нагадувань. Порівняльний аналіз універсальних систем управління навчанням (Google Classroom, Moodle, eTutorium) показав, що жодна з них не покриває повного переліку вимог до супроводу виробничої практики на промисловому підприємстві без надмірних витрат на адаптацію, що обґрунтувало доцільність власної розробки. За підсумками аналізу сформульовано мету й завдання роботи, визначено функціональні та нефункціональні вимоги до програмного забезпечення.

На основі визначених вимог спроєктовано та реалізовано онлайн-платформу AMP, яка має трирівневу клієнт-серверну архітектуру з декомпозицією серверної частини на чотири незалежні служби: REST API, Telegram-бот, планувальник нагадувань та обробник подій. Серверну частину побудовано засобами Python 3.12.4 і FastAPI з асинхронним доступом до бази даних, клієнтську — у вигляді односторінкового застосунку на Vue 3, Vite та Pinia. Розроблено схему бази даних із 28 таблиць під керуванням PostgreSQL 17, що відтворює ієрархічну структуру навчального контенту та прогресу студента й використовує UUID-ідентифікацію, шифрування чутливих полів, складені індекси та каскадне

видалення. Реалізовано REST API з єдиним префіксом /api/v1, JWT-авторизацією та близько 25 кінцевими точками, а також клієнтський інтерфейс із дванадцяти екранів для трьох ролей користувачів — студента, куратора та адміністратора.

Програмна реалізація охоплює ключові механізми платформи: авторизацію на основі JWT, послідовне розблокування навчального контенту, обмеження кількості спроб проходження тестів засобами Redis, автоматичну генерацію PDF-сертифіката з UUID-верифікацією, підсистему нагадувань електронною поштою та через Telegram, а також експорт аналітичних даних у формат Excel для куратора. Працездатність розробленого програмного забезпечення підтверджено методикою роботи користувачів із копіями екрана діючої платформи.

Розроблена платформа АМР дозволяє автоматизувати повний цикл супроводу виробничої практики: реєстрацію та авторизацію учасників, проходження навчального контенту й тестів із автоматичною перевіркою, відстеження прогресу студентів у режимі реального часу, автоматичну видачу та публічну верифікацію сертифікатів, своєчасне нагадування про навчальні події та формування аналітичної звітності для кураторів. Упровадження платформи усуває недоліки ручної організації практики, знижує трудомісткість обліку й контролю та підвищує прозорість процесу для всіх його учасників. Результати роботи орієнтовано на застосування промисловими підприємствами з корпоративними програмами практики, зокрема ПАТ «АрселорМіттал Кривий Ріг».

Усі поставлені у роботі завдання виконано повністю, а мету кваліфікаційної роботи — розроблення онлайн-платформи для підтримки проходження виробничої практики — досягнуто. Перспективними напрямами подальшого розвитку платформи є розширення набору типів навчальних завдань, інтеграція з корпоративними інформаційними системами підприємств та впровадження мобільного застосунку.

ПЕРЕЛІК ПОСИЛАНЬ

1. Що таке LMS: типи, можливості та переваги систем управління навчанням [Електронний ресурс] // Cases.media. – 2023. – Режим доступу: https://cases.media. – Дата звернення: 15.04.2026.
2. Mensah R. O. Design and Implementation of a Web-Based Internship Management System / R. O. Mensah, S. Akrobotu // International Journal of Computer Applications. – 2021. – Vol. 174, № 12. – P. 1–8. – Режим доступу: https://www.researchgate.net. – Дата звернення: 16.04.2026.
3. Mobile-based Student Internship Monitoring System using Progress Tracking Algorithm [Electronic resource] // International Research Journal on Advanced Science Hub. – 2022. – Vol. 4, № 5. – P. 105–112. – Access mode: https://rspsciencehub.com. – Date of access: 16.04.2026.
4. Google Classroom [Electronic resource] : official site. – Access mode: https://classroom.google.com. – Date of access: 17.04.2026.
5. Moodle [Electronic resource] : official site and documentation / Moodle Pty Ltd. – Access mode: https://moodle.org. – Date of access: 17.04.2026.
6. Google Classroom vs Moodle: LMS comparison [Electronic resource] // SelectHub. – 2024. – Access mode: https://www.selecthub.com. – Date of access: 18.04.2026.
7. eTutorium LMS [Електронний ресурс] : офіційний сайт. – Режим доступу: https://etutorium.com/lms. – Дата звернення: 18.04.2026.
8. Про вищу освіту : Закон України від 01.07.2014 № 1556-VII (зі змінами) [Електронний ресурс] // Верховна Рада України. – Режим доступу: https://zakon.rada.gov.ua/laws/show/1556-18. – Дата звернення: 20.04.2026.
9. Положення про проведення практики студентів вищих навчальних закладів України : затв. наказом Міністерства освіти України від 08.04.1993 № 93 [Електронний ресурс] // Верховна Рада України. –

Режим доступу: https://zakon.rada.gov.ua/laws/show/z0035-93. – Дата звернення: 20.04.2026.

10. PostgreSQL 17 Documentation [Electronic resource] / The PostgreSQL Global Development Group. – 2024. – Access mode: https://www.postgresql.org/docs/17/. – Date of access: 05.05.2026.
11. SQLAlchemy 2.0 Documentation [Electronic resource] / M. Bayer. – Access mode: https://docs.sqlalchemy.org/en/20/. – Date of access: 05.05.2026.

Додаток А. Текст програми

У цьому додатку наведено ключові фрагменти вихідного коду програмної платформи AMP, що ілюструють реалізацію найважливіших механізмів системи. Оскільки розробка є комерційною, наведено лише показові фрагменти без конфіденційних даних (секретних ключів, паролів, рядків підключення); довгі допоміжні ділянки скорочено та позначено коментарем «# ...» або «// ...».

А.1 Генерація та перевірка JWT-токенів

Серверна частина застосовує токени JSON Web Token для автентифікації без збереження стану сесії. Функції наведено за файлом src/common/services/auth.py.

```
from datetime import UTC, datetime, timedelta
import jwt
from src.configs import settings


def create_access_token(
    data: dict, expires_delta: timedelta, token_type: str |
None = None
) -> str:
    to_encode = data.copy()
    expire = datetime.now(UTC) + expires_delta
    extra_data = {"exp": expire}
    if token_type:
        extra_data["token_type"] = token_type
    to_encode.update(extra_data)
    return jwt.encode(
        to_encode, settings.SECRET_KEY,
        algorithm=settings.JWT_ENCODE_ALGORITHM,
    )


def decode_access_token(token: str) -> dict:
    return jwt.decode(
        token, settings.SECRET_KEY,
        algorithms=[settings.JWT_ENCODE_ALGORITHM],
    )
```

А.2 Залежність авторизації студента

Перевірка токена винесена в залежність FastAPI, яка декодує токен, завантажує студента з бази даних і повертає відповідь 403 у разі простроченого чи недійсного токена (файл src/api/depends/student_auth.py).

```
jwt_header = APIKeyHeader(
    name="Authorization", auto_error=True, scheme_name="JWT"
)


async def authorize_student(
    session: DBSessionDep, jwt_token: str =
Security(jwt_header)
) -> StudentModel:
    try:
        student = await
StudentManager(session).get_obj_from_jwt(jwt_token)
    except jwt.ExpiredSignatureError as e:
        raise HTTPException(status_code=403, detail="Expired
token") from e
    except jwt.InvalidTokenError as e:
        raise HTTPException(status_code=403, detail="Invalid
token") from e
    if not student:
        raise HTTPException(
            status_code=403, detail="Could not validate
credentials"
        )
    return student


StudentAuthDep = Annotated[StudentModel,
Depends(authorize_student)]
```

А.3 Послідовне розблокування навчального контенту

Сервіс відкриває наступний модуль або завдання лише після завершення поточного, забезпечуючи лінійну траєкторію проходження курсу (файл src/common/services/student_course/student_course_availability.py).

```
async def update_availability(self) -> None:
    """Оновити доступність модулів і завдань за статусом
виконання."""
    await self._update_module_availability()
```

```
    module = await self.s_course_module_manager \
        .last_uncompleted_available(self.s_course)
    if not module:
        module = await self.s_course_module_manager \
            .last_available(self.s_course)
    if module:
        await self.update_task_availability(module)


async def _update_module_availability(self) -> None:
    """Відкрити наступний модуль після завершення
поточного."""
    manager = self.s_course_module_manager
    while True:
        available = await
manager.last_uncompleted_available(self.s_course)
        unavailable = await manager \
            .last_uncompleted_unavailable(self.s_course)
        if not unavailable:
            break
        if not available:
            await manager.toggle_available(unavailable)
            continue
        if available.completed:
            await manager.toggle_available(unavailable)
        else:
            break
```

A.4 Генерація PDF-сертифіката засобами ReportLab

Сертифікат формується накладанням текстового шару (ПІБ, курс, дата, UUID) на заздалегідь підготовлений PDF-шаблон. Нижче наведено спрощений варіант методу (без логіки перенесення довгих рядків) за файлом src/common/services/pdf_writer.py.

```
from reportlab.pdfgen import canvas
from reportlab.pdfbase import pdfmetrics
from reportlab.lib.pagesizes import landscape, A4
from reportlab.pdfbase.ttfonts import TTFont
from PyPDF2 import PdfReader, PdfWriter
import io


class PDFWriter:
    def __init__(self, file_service, fonts_path: dict[str,
str]) -> None:
        self.file_service = file_service
```

```
        self.fonts_path = fonts_path

    def write_by_template(
        self, template_path, text_elements,
        output_filename=None, page_size=landscape(A4),
    ) -> str:
        output_path =
self.file_service.get_file_path(output_filename)
        reader = PdfReader(template_path)
        writer = PdfWriter()
        template_page = reader.pages[0]

        # Текстовий шар поверх шаблону сертифіката
        packet = io.BytesIO()
        c = canvas.Canvas(packet, pagesize=page_size,
pageCompression=0)
        for config in text_elements.values():
            font_name = config.get("font", "Helvetica")
            if (font_name in self.fonts_path
                    and font_name not in pdfmetrics._fonts):
                pdfmetrics.registerFont(
                    TTFont(font_name,
self.fonts_path[font_name]))
            c.setFont(font_name, config.get("size", 12))
            if "color" in config:
                c.setFillColor(config["color"])
            c.drawString(config["x"], config["y"],
config["text"])
        c.save()

        # Об'єднання шаблону з текстовим шаром
        packet.seek(0)
        overlay = PdfReader(packet)
        template_page.merge_page(overlay.pages[0])
        writer.add_page(template_page)
        with open(output_path, "wb") as output_file:
            writer.write(output_file)
        return output_path
```

A.5 Планування нагадувань засобами APScheduler

Окремий процес-планувальник реєструє щоденні cron-задачі (нагадування, контроль строків проходження) у асинхронному планувальнику (файли src/scheduler/initializer.py та src/scheduler/task_configs.py).

```
from apscheduler.schedulers.asyncio import AsyncIOScheduler
from src.scheduler.task_configs import client_manager_tasks,
Task
```

```
def init_scheduler_tasks(
    scheduler: AsyncIOScheduler, scheduled_tasks: tuple[Task]
) -> None:
    for task in scheduled_tasks:
        scheduler.add_job(
            func=task.task_class().execute,
            kwargs=task.task_kwargs,
            trigger=task.scheduler_trigger,
            **task.scheduler_kwargs,
        )


async def init_scheduler() -> None:
    scheduler = AsyncIOScheduler()
    scheduler.start()
    init_scheduler_tasks(scheduler, client_manager_tasks)
```

Опис набору задач (щоденно о 00:00):

```
@dataclass
class Task:
    task_class: TaskABC
    task_kwargs: dict
    scheduler_trigger: Literal["interval", "date", "cron"]
    scheduler_kwargs: dict


client_manager_tasks = (
    Task(task_class=ReminderNotification, task_kwargs={},
         scheduler_trigger="cron",
         scheduler_kwargs={"hour": 0, "minute": 0}),
    # ... аналогічно StudentCourseTimeoutNotification,
    #     ExpiredCourseValidation
)
```

А.6 Перехоплювачі HTTP-запитів (Axios)

Клієнтська частина централізує роботу з API через єдиний екземпляр Axios; перехоплювач запиту автоматично додає JWT-токен до заголовка Authorization (файл src/utils/api.js).

```
import axios from "axios";
import { useAuthStore } from "@/stores/auth";

const apiUrl = import.meta.env.VITE_APP_API_URL;
const timeout = 50 * 1000; // 50 секунд

const api = axios.create({
```

```
    baseURL: apiUrl,
    timeout,
    headers: { "Content-Type": "application/json" },
});

api.interceptors.request.use(
    (config) => {
        const authStore = useAuthStore();
        if (authStore.isAuthenticated) {
            config.headers.Authorization =
`${authStore.getToken}`;
        }
        return config;
    },
    (error) => Promise.reject(error)
);

api.interceptors.response.use(
    (response) => response,
    (error) => Promise.reject(error)
);

export default api;
```

А.7 Сховище стану автентифікації (Pinia)

Сховище auth зберігає токен доступу та відомості про користувача, синхронізує токен із localStorage і надає похідні значення (ознака автентифікованості, доступність сертифіката). Наведено за файлом src/stores/auth.js (допоміжні дії скорочено).

```
import { defineStore } from 'pinia'
import { me } from '../api/apiStudent'
import { getTokenFromLocalStorage } from
'../utils/getTokenFromLocalStorage'

export const useAuthStore = defineStore('auth', {
  state: () => ({
    token: getTokenFromLocalStorage('student-token'),
    userInfo: null,
    profile: null,
  }),
  getters: {
    getToken: (state) => state.token,
    isAuthenticated: (state) => !!state.token,
    certificate: (state) => state.profile?.certificate,
  },
  actions: {
```

```
    login(token) {
      this.token = token
      localStorage.setItem('student-token', token)
    },
    logout() {
      this.token = null
      this.userInfo = null
      localStorage.removeItem('student-token')
    },
    async getUser() {
      if (this.isAuthenticated) {
        try {
          const response = await me()
          this.userInfo = response.data
          this.updateProfile()
          return this.userInfo
        } catch (error) {
          this.logout()
        }
      }
      return null
    },
    // ... getProfile / selectProfile / updateProfile
  },
})
```

А.8 Таймер блокування повторних спроб (StudentModuleView)

На екрані проходження модуля після надсилання тесту запускається таймер зворотного відліку до наступної дозволеної спроби (cooldown), який щосекунди оновлює відображуваний час (файл src/views/StudentModuleView.vue).

```
// Надсилання відповіді й оновлення стану модуля
const submit = async (taskId, answers) => {
  try {
    await courseSubmitTask(profileId, moduleId, taskId,
answers)
    await updateModuleData()
    await updateSelectedTask()
    await auth.getUser()
    await updateTestResult()
  } catch (error) {
    console.error('Error submitting test:', error)
  }
}

// Запуск зворотного відліку до наступної спроби
const addCooldownTimerInterval = (seconds) => {
```

```
  const targetTime = Date.now() + seconds * 1000
  if (cooldownTimerData.value.timer) {
    clearInterval(cooldownTimerData.value.timer)
  }
  cooldownTimerData.value.timer = setInterval(() => {
    const remaining = targetTime - Date.now()
    if (remaining <= 0) {
      clearInterval(cooldownTimerData.value.timer)
      cooldownTimerData.value.text = '00:00:00'
      return
    }
    const hrs = Math.floor((remaining / 3600000) % 24)
      .toString().padStart(2, '0')
    const mins = Math.floor((remaining / 60000) % 60)
      .toString().padStart(2, '0')
    const secs = Math.floor((remaining / 1000) % 60)
      .toString().padStart(2, '0')
    cooldownTimerData.value.text = `${hrs}:${mins}:${secs}`
  }, 1000)
}
```