

Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет _____ Комп'ютерних наук _____
(повна назва)
Кафедра _____ Штучного інтелекту _____
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти _____ перший (бакалаврський) _____

Розробка вебзастосунку для підготовки випускників шкіл до складання
Національно Мультидисциплінарного іспиту із підсистемою перевірки
результатів тестів _____
(тема)

Виконав:
здобувач _____ четвертого _____ року навчання,
групи _____ ІТШ-22-1 _____

_____ Олесь ХАМІНОВА _____
(власне ім'я, прізвище)

Спеціальність 122 Комп'ютерні науки _____

(код і повна назва спеціальності)
Тип програми _____ освітньо-професійна _____

Освітня програма _____ Штучний інтелект _____

(повна назва освітньої програми)

Керівник _____ ст.викл. Ігор БІБІЧКОВ _____
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри ШІ

(підпис)

_____ Лариса ЧАЛА _____
(власне ім'я, прізвище)

2026 р.

Харківський національний університет радіоелектроніки

Факультет _____ Комп'ютерних наук _____

Кафедра _____ Штучного інтелекту _____

Рівень вищої освіти _____ перший (бакалаврський) _____

Спеціальність _____ 122 Комп'ютерні науки _____

(код і повна назва)

Тип програми _____ освітньо-професійна _____

Освітня програма _____ Штучний інтелект _____

(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____

(підпис)

« _____ » _____ 20 ____ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві _____ Хаміновій Олесі Олександрівні _____

(прізвище, ім'я, по батькові)

1. Тема роботи _____ Розробка вебзастосунку для підготовки випускників шкіл до складання Національно Мультидисциплінарного іспиту із підсистемою перевірки результатів тестів _____

затверджена наказом університету від 5 червня 2026 р. № 488Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 9 липня 2026 р.

3. Вихідні дані до роботи _____ Дослідницькі матеріали про структуру та методику формування завдань Національного мультидисциплінарного тесту, технічна документація платформи .NET 8 та фреймворку ASP.NET Core MVC, документація об'єктно-реляційного перетворювача Entity Framework Core та СУБД SQLite, специфікації програмного інтерфейсу Anthropic Claude API, документація бібліотек Microsoft Playwright, AngleSharp та Polly, дані інтернет-джерел щодо підготовки випускників. _____

4. Перелік питань, що потрібно опрацювати в роботі _____

1) Аналіз предметної галузі та постановка задачі _____

2) Теоретичні дослідження _____

3) Проєктування та розробка моделі _____

4) Проєктування та розробка кінцевого застосунку _____

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / термін виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	08.06.2026	виконано
2	Аналіз предметної галузі та існуючих аналогів	08.06.2026 – 09.06.2026	виконано
3	Постановка задачі та обґрунтування технологій	10.06.2026 – 12.06.2026	виконано
4	Теоретичні дослідження та аналіз методів розробки	13.06.2026 – 14.06.2026	виконано
5	Проектування загальної архітектури та моделі даних	15.06.2026 – 18.06.2026	виконано
6	Програмна реалізація вебзастосунку та модулів ІІІ	19.06.2026 – 20.06.2026	виконано
7	Тестування системи та оцінювання її функціональності	21.06.2026 – 22.06.2026	виконано
8	Оформлення пояснювальної записки та додатків	23.06.2026 – 24.06.2026	виконано
9	Створення презентації та ілюстративних матеріалів	25.06.2026 – 26.06.2026	виконано
10	Захист перед ЕК	09.07.2026	виконано

Дата видачі завдання 8 червня 20 26 р.

Здобувач _____
(підпис)

Керівник роботи _____
(підпис)

ст.викл. Ігор БІБІЧКОВ
(посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка: 114 с., 34 рис., 14 табл., 3 дод., 22 джерела.

АНАЛІТИКА ПРОГРЕСУ, ВЕБЗАСТОСУНОК, ВЕБСКРАПІНГ, НАЦІОНАЛЬНИЙ МУЛЬТИДИСЦИПЛІНАРНИЙ ТЕСТ, ПЕРСОНАЛІЗОВАНЕ НАВЧАННЯ, ТЕСТУВАННЯ, ШТУЧНИЙ ІНТЕЛЕКТ, ASP.NET CORE, ENTITY FRAMEWORK CORE.

Об'єкт дослідження – процес самостійної підготовки випускників шкіл до Національного мультидисциплінарного тесту засобами вебтехнологій.

Предмет дослідження – методи та засоби реалізації вебзастосунку з підсистемою автоматизованої перевірки результатів тестування, зокрема аналізу відповідей відкритого типу засобами штучного інтелекту.

Мета роботи – розробка вебзастосунку для підготовки випускників шкіл до складання НМТ, що забезпечує доступ до банку реальних тестових завдань, перевірку результатів із розгорнутим зворотним зв'язком та аналіз відкритих відповідей засобами штучного інтелекту.

Методи дослідження – у роботі використано методи порівняльного аналізу програмних аналогів, методи проєктування клієнт-серверних застосунків на основі архітектурного шаблону MVC, методи автоматизованого збору даних із вебресурсів засобами браузерної автоматизації, методи інтеграції зовнішніх API для оцінювання текстових відповідей.

Проведено аналіз існуючих платформ підготовки до НМТ, за результатами якого виявлено відсутність комплексного рішення з розгорнутим зворотним зв'язком, аналітикою прогресу та плануванням підготовки.

ABSTRACT

Bachelor's thesis contains: 114 p., 34 fig., 14 tabl., 3 ann., 22 references.

ARTIFICIAL INTELLIGENCE, ASP.NET CORE, ENTITY FRAMEWORK CORE, NATIONAL MULTIDISCIPLINARY TEST, PERSONALIZED LEARNING, PROGRESS ANALYTICS, TESTING, WEB APPLICATION, WEB SCRAPING.

The object of the study is the process of independent preparation of school graduates for the National Multidisciplinary Test using web technologies.

The subject of the study is methods and tools for implementing a web application with an automated test result verification subsystem, including analysis of open-ended responses using artificial intelligence.

The goal of the work is to develop a web application for preparing school graduates for the NMT, providing access to a bank of real test tasks, result verification with detailed feedback, and analysis of open-ended responses using artificial intelligence.

Research methods – the work employs comparative analysis methods for existing software analogues, client-server application design methods based on the MVC architectural pattern, automated web data collection methods using browser automation, external API integration methods for evaluating text-based responses.

An analysis of existing NMT preparation platforms was conducted, revealing the absence of a comprehensive solution with detailed feedback, progress analytics, and preparation planning.

ЗМІСТ

Вступ.....	11
1 Аналіз предметної галузі та постановка задачі	13
1.1 Опис предметної галузі	13
1.2 Аналіз існуючих програмних аналогів.....	15
1.3 Вибір технологій розробки	18
1.4 Постановка задачі дослідження	21
2 Теоретичні дослідження	24
2.1 Архітектурні підходи до розробки	24
2.2 Методи автоматизованого збору даних	27
2.3 Методи перевірки відповідей	32
2.4 Оцінювання відповідей засобами ШІ.....	34
2.5 Методи аналітики прогресу	37
2.6 Метод формування розкладу	40
2.7 Методи захисту даних.....	43
3 Проєктування та розробка моделі	45
3.1 Загальна архітектура системи.....	45
3.2 Модель даних застосунку	48
3.3 Структура класів та сервісів	51
3.4 Проєктування сценаріїв.....	53
3.5 Модель збору завдань	56
3.6 Модель оцінювання ШІ.....	59
3.7 Модель аналітики та планування	62
4 Проєктування та розробка кінцевого застосунку	65
4.1 Реалізація рівня даних	65
4.2 Реалізація бізнес-логіки	69
4.3 Реалізація модуля скрапінгу	73
4.4 Інтеграція API штучного інтелекту	76
4.5 Реалізація вебінтерфейсу	79

4.6 Забезпечення безпеки.....	84
4.7 Тестування застосунку	87
Висновки.....	89
Перелік джерел посилання	91
Додаток А Таблиці	93
Додаток Б Лістинги	100
Додаток В Відомість кваліфікаційної роботи.....	114

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

БД – база даних;

ЗНО – зовнішнє незалежне оцінювання;

НМТ – національний мультидисциплінарний тест;

СУБД – система керування базами даних;

ШІ – штучний інтелект;

AJAX – Asynchronous JavaScript and XML – підхід до побудови інтерактивних користувацьких інтерфейсів вебзастосунків, що полягає в асинхронному обміні даними браузера з вебсервером у фоновому режимі;

Alpine.js – мінімалістичний клієнтський JavaScript-фреймворк для реактивного керування поведінкою елементів інтерфейсу без повного перезавантаження сторінок;

AngleSharp – сторонній .NET-інструмент (бібліотека) для швидкого та повного синтаксичного аналізу (парсер) HTML5-документів на основі об'єктної моделі DOM;

Anthropic Claude API – хмарний інтерфейс доступу до великих мовних моделей сімейства Claude від компанії Anthropic, задіяний для змістовного оцінювання відповідей учнів;

API – програмний інтерфейс застосунку, набір визначень, протоколів та інструментів для взаємодії між різними програмними компонентами;

BLL – Business Logic Layer – рівень бізнес-логіки програмної системи;

Bootstrap – популярний безкоштовний CSS-фреймворк із відкритим кодом для адаптивної верстки клієнтських інтерфейсів;

CSRF – Cross-Site Request Forgery – міжсайтна підробка запиту, вид атак на вебсистеми, що полягає в примусовому виконанні дій від імені автентифікованого користувача;

CSS – Cascading Style Sheets – каскадні таблиці стилів, мова опису зовнішнього вигляду HTML-документа;

DOM – Document Object Model – об'єктна модель документа, програмний інтерфейс для HTML та XML документів, що подає структуру тексту у вигляді деревоподібної моделі;

DTO – Data Transfer Object – об'єкт передавання даних, шаблон проєктування, що використовується для транспортування даних між рівнями системи;

EF Core – Entity Framework Core – сучасний об'єктно-реляційний перетворювач (ORM) від компанії Microsoft для платформи .NET;

FullCalendar.js – інтерактивний JavaScript-компонент (бібліотека) для візуалізації календаря та планування подій або розкладу занять;

Headless browser – безголовий браузер, веббраузер без графічного інтерфейсу користувача, який керується програмно для автоматизованого збору даних (скрапінгу);

HTML – HyperText Markup Language – стандартизована мова розмітки гіпертексту, що використовується для створення вебсторінок;

HTTP – HyperText Transfer Protocol – протокол передавання гіпертексту, прикладний протокол для обміну даними в мережі;

HTTPS – HyperText Transfer Protocol Secure – розширення протоколу HTTP з підтримкою шифрування для забезпечення безпеки передавання даних;

JSON – JavaScript Object Notation – текстовий формат обміну даними, заснований на JavaScript, що легко зчитується людьми та парситься програмами;

LINQ – Language Integrated Query – проєкт компанії Microsoft, що додає синтаксис запитів мовами C# або VB.NET для взаємодії з різними типами даних;

Microsoft Playwright – сучасна бібліотека автоматизації та керування фоновими браузером (Chromium, WebKit, Firefox) через єдиний програмний інтерфейс;

MVC – Model-View-Controller – «Модель-Подання-Контролер», архітектурний шаблон, що розділяє вебзастосунок на три взаємопов'язані компоненти для ізоляції бізнес-логіки від інтерфейсу;

ORM – Object-Relational Mapping – об'єктно-реляційне відображення (перетворення), технологія, що зв'язує бази даних із концепціями об'єктно-орієнтованих мов програмування;

Polly – .NET бібліотека для забезпечення стійкості застосунків та обробки тимчасових мережових помилок за допомогою політик повторень (Retry) та автоматичних пауз;

Razor Views – вбудований у платформу ASP.NET Core двигун шаблонізації вебсторінок, що дозволяє поєднувати HTML-розмітку із кодом на мові C#;

SQL – Structured Query Language – мова структурованих запитів, що використовується для взаємодії з реляційними базами даних;

SQLite – компактна, вбудована реляційна система керування базами даних, яка зберігає всю інформацію в одному локальному файлі без потреби у виділеному сервері;

URL – Uniform Resource Locator – єдиний вказівник курсу, стандартизована адреса вебресурсу в мережі Інтернет.

ВСТУП

Сучасна система освіти в Україні переживає системну трансформацію, яка охоплює не лише зміст навчальних програм, а й підходи до оцінювання результатів навчання. Одним із ключових інструментів такого оцінювання є Національний мультидисциплінарний тест (НМТ) – стандартизована форма перевірки знань, уведена для вступу до закладів вищої освіти. На відміну від традиційного зовнішнього незалежного оцінювання, НМТ орієнтований на перевірку здатності абітурієнтів застосовувати знання в міждисциплінарному контексті, що ставить підвищені вимоги як до учнів, так і до якості їхньої підготовки.

Аналіз поточного стану ринку освітніх послуг свідчить про зростання попиту на цифрові інструменти самопідготовки. Значна частина випускників шкіл вдається до онлайн-платформ для опрацювання тестових завдань, проте більшість наявних рішень не враховують специфіку НМТ, мають застарілий інтерфейс або не надають детального аналізу допущених помилок. Серед найпоширеніших недоліків існуючих застосунків – відсутність персоналізованого зворотного зв'язку, обмежений банк завдань, а також неможливість відстежувати динаміку прогресу здобувача впродовж тривалого часу.

Актуальність дослідження зумовлена кількома чинниками:

- щорічно в Україні десятки тисяч абітурієнтів складають НМТ, і рівень їхньої підготовки безпосередньо визначає можливість вступу до бажаного закладу вищої освіти;
- розповсюдження засобів дистанційного навчання, особливо в умовах воєнного часу, суттєво підвищило значущість якісних онлайн-інструментів;
- швидкий розвиток клієнт-серверних технологій і фреймворків для розробки вебзастосунків відкриває можливості для створення сучасних, масштабованих і зручних у використанні рішень.

Об'єктом дослідження є процес підготовки випускників шкіл до складання Національного мультидисциплінарного тесту засобами вебтехнологій.

Предметом дослідження є методи та інструменти розробки вебзастосунку з підсистемою автоматизованої перевірки результатів тестування, що забезпечує зворотний зв'язок і персоналізований аналіз помилок.

Метою кваліфікаційної роботи є проєктування та розробка вебзастосунку для підготовки випускників шкіл до складання НМТ, який включає підсистему перевірки результатів тестів із можливістю аналізу допущених помилок і відстеження динаміки навчального прогресу.

Для досягнення поставленої мети в роботі вирішуються такі задачі:

- аналіз існуючих програмних рішень для підготовки до стандартизованих тестів та виявлення їхніх переваг і недоліків;
- дослідження структури та змісту НМТ з метою формування репрезентативного банку тестових завдань;
- вибір і обґрунтування стеку технологій для розробки клієнтської та серверної частин застосунку;
- проєктування архітектури вебзастосунку та бази даних;
- розробка підсистеми перевірки результатів тестів із формуванням детального аналізу помилок;
- тестування розробленого застосунку та оцінювання його функціональності.

Практична цінність роботи полягає в тому, що розроблений вебзастосунок може бути впроваджений у навчальний процес загальноосвітніх шкіл та використовуватись індивідуально для самопідготовки. Підсистема перевірки результатів дозволяє не лише фіксувати правильні й неправильні відповіді, а й формувати профіль слабких тематичних зон конкретного користувача, тим самим сприяючи цілеспрямованому й ефективному опрацюванню навчального матеріалу.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Опис предметної галузі

Система вступу до закладів вищої освіти в Україні зазнала суттєвих змін протягом останніх років. З 2022 року основним інструментом оцінювання готовності абітурієнтів став Національний мультидисциплінарний тест – єдина стандартизована процедура, що прийшла на зміну зовнішньому незалежному оцінюванню в його традиційній формі. НМТ охоплює кілька дисциплін в межах одного тестування, перевіряє не лише предметні знання, а й здатність застосовувати їх у нових, нестандартних ситуаціях. Це принципово змінює вимоги до підготовки випускників і робить необхідними нові підходи до організації самостійної роботи учнів.

Предметна галузь даної кваліфікаційної роботи охоплює два взаємопов'язані напрями: організацію процесу підготовки до стандартизованих тестів і програмну реалізацію інструментів такої підготовки у вигляді вебзастосунку. Перший напрям стосується педагогічних і методичних аспектів: структури тесту, типів завдань, принципів оцінювання та закономірностей у рівні складності. Другий – технічної реалізації системи, яка забезпечує зручний доступ до навчального контенту, імітує умови реального тестування і надає користувачу інформацію про результати.

НМТ складається з чотирьох блоків: українська мова, математика, історія України та один предмет за вибором. Завдання тесту структуровані за рівнями складності й передбачають різні формати відповідей – від вибору однієї правильної відповіді до завдань з розгорнутою відповіддю. Така різноманітність форматів вимагає від підготовчої платформи гнучкої системи введення та перевірки відповідей, яка враховує специфіку кожного типу завдань.

Електронні системи тестування як клас програмного забезпечення мають власну усталену архітектуру. Їхніми ключовими компонентами є:

- банк завдань із механізмом вибірки;
- інтерфейс проходження тесту;
- підсистема перевірки та збереження результатів;
- модуль аналітики й звітності.

У контексті підготовки до НМТ особливого значення набуває підсистема перевірки результатів, оскільки вона виконує не лише технічну функцію зіставлення відповідей, а й педагогічну – надає здобувачеві розгорнутий зворотний зв'язок, що допомагає зрозуміти причину помилки, а не лише сам факт її наявності.

Вебзастосунок, як форма реалізації такої системи, має очевидні переваги перед настільними або мобільними програмами. Він не потребує встановлення, доступний з будь-якого пристрою через браузер, легко оновлюється та масштабується. Клієнт-серверна архітектура дозволяє зберігати дані користувачів централізовано, що уможливорює відстеження прогресу між сесіями та формування персональної статистики. Сучасні фреймворки для розробки фронтенду і бекенду суттєво прискорюють розробку та забезпечують відповідність застосунку актуальним стандартам якості й безпеки.

Важливим аспектом предметної галузі є також поведінка кінцевого користувача – випускника школи. Така аудиторія характеризується різним рівнем цифрової грамотності, необхідністю частих коротких сесій роботи (а не тривалих безперервних занять) і підвищеною чутливістю до зручності інтерфейсу. Тому при проєктуванні застосунку принципового значення набувають рішення щодо навігації, візуального оформлення та швидкості відгуку системи. Незручний або повільний інтерфейс безпосередньо знижує мотивацію користувача до регулярної підготовки, що суперечить основній меті застосунку.

Таким чином, предметна галузь даної роботи є міждисциплінарною: вона поєднує елементи педагогіки, інформаційних технологій і проєктування програмного забезпечення. Ефективне рішення у цій галузі вимагає врахування як технічних вимог до надійності та продуктивності системи, так і методичних вимог до організації навчального процесу й надання зворотного зв'язку.

1.2 Аналіз існуючих програмних аналогів

З метою визначення напрямів розробки проведено аналіз програмних рішень, що функціонують у сегменті підготовки до стандартизованого тестування. Розглянуто як вітчизняні платформи, орієнтовані безпосередньо на формат НМТ/ЗНО, так і зарубіжні системи адаптивного навчання загального призначення.

Платформа Prometheus (prometheus.org.ua) є найбільшою вітчизняною освітньою платформою відкритого доступу. Вона пропонує структуровані відеокурси з базовими тестами самоперевірки. Проте банк завдань прив'язаний до конкретних курсів, а не до реальних варіантів іспиту. Відсутня можливість проходження повноцінного тестового сеансу за структурою НМТ, а зворотний зв'язок обмежується відображенням правильної відповіді без пояснень.

Ресурс zno.osvita.ua містить архів завдань реальних іспитів ЗНО та НМТ в розрізі предметів і років. Завдання систематизовані та доступні для перегляду, однак платформа фактично є статичним архівом: відсутній персональний кабінет, збереження результатів, аналітика прогресу та будь-який механізм зворотного зв'язку. Ресурс придатний лише для ручного перегляду матеріалу.

Платформа «На урок» (naurok.com.ua) надає вчителям інструменти для створення тестів, які учні можуть проходити за посиланням. Однак вона орієнтована передусім на шкільний навчальний процес під керівництвом

вчителя, а не на самостійну підготовку. Персональна статистика учня між окремими сесіями не накопичується, а формат завдань не відтворює структуру реального НМТ.

Серед зарубіжних аналогів розглянуто Khan Academy – платформу з розгалуженою системою адаптивного навчання та детальною аналітикою прогресу учня. Незважаючи на високу педагогічну якість, платформа не підтримує українськомовний контент і не містить завдань формату НМТ. Quizlet (quizlet.com) дозволяє створювати власні набори карток і тести, проте є універсальним інструментом без прив'язки до конкретного іспиту та без аналізу відповідей. Результати порівняльного аналізу представлено у таблиці 1.1.

Таблиця 1.1 – Порівняльний аналіз існуючих платформ

Критерій	Prometheus	zno.osvita.ua	«На урок»	Khan Academy
Банк завдань НМТ	Частково	Так	Ні	Ні
Персональний кабінет	Так	Ні	Частково	Так
Збереження результатів	Так	Ні	Ні	Так
Аналітика прогресу	Базова	Ні	Ні	Розширена
Розгорнутий зворотний зв'язок	Ні	Ні	Ні	Так
Аналіз відкритих відповідей	Ні	Ні	Ні	Ні
Планування підготовки	Ні	Ні	Ні	Ні
Україномовний контент НМТ	Так	Так	Так	Ні

Для наочного представлення результатів порівняльного аналізу на рисунку 1.1 побудовано радарну діаграму, що відображає восьми-критеріальний профіль кожної з розглянутих платформ. Вісі діаграми

відповідають функціональним критеріям оцінювання, значення на кожній осі приймають одне з трьох рівнів: 0 – функція відсутня, 0,5 – реалізована частково, 1 – реалізована повністю. З діаграми видно, що жодна з платформ не формує рівномірно заповненого полігону: Prometheus має прогалини у зворотному зв'язку та плануванні, zno.osvita.ua – майже у всіх критеріях, окрім наявності завдань НМТ, Khan Academy – у підтримці україномовного контенту та аналізі відкритих відповідей. Це наочно підтверджує відсутність комплексного рішення на ринку та обґрунтовує необхідність розробки спеціалізованого застосунку.

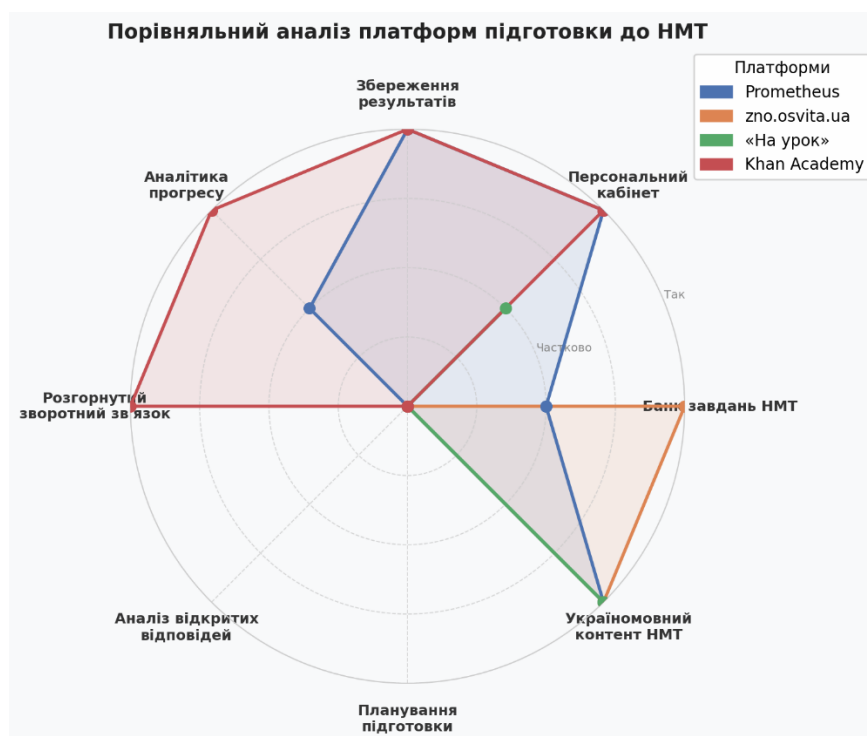


Рисунок 1.1 – Радарна діаграма порівняння платформ підготовки до НМТ за функціональними критеріями

Аналіз виявив спільні недоліки розглянутих рішень:

– аналітика прогресу або відсутня, або не відображає слабкі теми у розрізі конкретного предмету;

– жодна вітчизняна платформа не надає функціонал планування підготовки, інтегрований із результатами тестування.

Виявлені недоліки обумовлюють актуальність розробки спеціалізованого вебзастосунку, що поєднує банк реальних завдань НМТ, підсистему перевірки результатів з аналізом відповідей засобами штучного інтелекту, персональну аналітику та інструмент планування підготовки.

1.3 Вибір технологій розробки

Вибір технологічного стеку здійснено з урахуванням функціональних вимог до застосунку, зокрема необхідності реалізації клієнт-серверної архітектури, роботи з реляційною базою даних, інтеграції зовнішнього API штучного інтелекту та автоматизованого збору навчального контенту.

На рисунку 1.2 представлено загальну архітектуру розробленої системи.

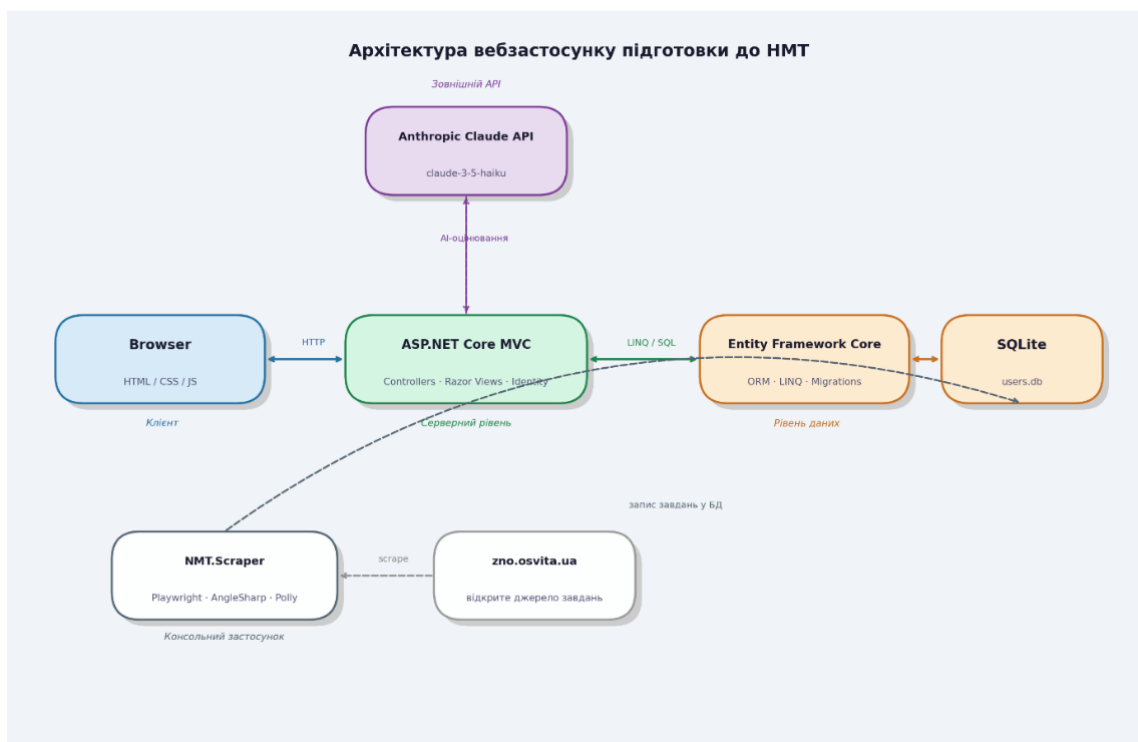


Рисунок 1.2 – Загальна архітектура вебзастосунку підготовки до НМТ

Серверна частина. Як основний фреймворк для реалізації серверної частини обрано ASP.NET Core 8 на платформі .NET 8. Серед альтернатив розглядалися Django (Python), Laravel (PHP) та Spring Boot (Java). ASP.NET Core відзначається вбудованою підтримкою впровадження залежностей, вбудованим middleware-конвеєром і строгою типізацією мови C#, що знижує ймовірність помилок на етапі компіляції. Підхід MVC (Model-View-Controller) обрано як такий, що забезпечує чітке розмежування рівнів представлення, бізнес-логіки та доступу до даних і добре масштабується зі збільшенням кількості функціональних модулів.

Доступ до даних. Для взаємодії з базою даних застосовано Entity Framework Core 8 – об'єктно-реляційний перетворювач (ORM), що входить до екосистеми .NET. EF Core забезпечує підтримку міграцій схеми бази даних, роботу з LINQ-запитами та автоматичне відстеження змін сутностей. Як СУБД обрано SQLite, оскільки застосунок не передбачає одночасної роботи великої кількості користувачів у конкурентному режимі, а файловий характер SQLite спрощує розгортання без встановлення окремого сервера бази даних.

Автентифікація та авторизація. Управління обліковими записами реалізовано засобами ASP.NET Core Identity – вбудованої підсистеми .NET, що надає готові механізми реєстрації, входу, хешування паролів і захисту від CSRF-атак. Застосування Identity дозволило уникнути розробки власних механізмів безпеки і скористатися рішенням, що регулярно оновлюється та проходить аудит безпеки.

Збір навчального контенту. Для автоматизованого збору завдань із відкритого ресурсу zno.osvita.ua розроблено окремий консольний застосунок NMT.Scraper. У ньому застосовано бібліотеку Microsoft Playwright 1.40 для керування браузером та відтворення поведінки реального користувача, AngleSharp 1.1.2 для синтаксичного розбору HTML-структури сторінок і Polly 8.3 для реалізації політики повторних спроб при

тимчасових мережеских помилках. Винесення скрапера в окремий проєкт дозволило ізолювати логіку збору даних від основного вебзастосунку.

Інтеграція штучного інтелекту. Для аналізу відповідей відкритого типу застосовано Anthropic Claude API з моделлю claude-3-5-haiku. Обрана модель забезпечує прийнятний баланс між швидкістю генерації відповіді та якістю оцінювання текстових пояснень учня. API-ключ зберігається поза репозиторієм у механізмі User Secrets платформи .NET, що виключає його потрапляння до системи контролю версій. Передбачено режим MockAiService для роботи застосунку без ключа в умовах розробки.

Клієнтська частина. Інтерфейс користувача реалізовано на основі Razor Views – шаблонізатора, інтегрованого в ASP.NET Core MVC. Базову верстку та типові елементи інтерфейсу побудовано засобами CSS-фреймворку Bootstrap 5, поверх якого реалізовано власну дизайн-систему на CSS-змінних – для підтримки світлої й темної тем і єдиного візуального стилю. Для модуля планування підготовки застосовано бібліотеку FullCalendar.js, що надає готовий інтерактивний компонент календаря з підтримкою подій. У таблиці А.1 (додаток А) узагальнено обрані технології та їх призначення в межах проєкту.

На рисунку 1.3 представлено технологічний стек застосунку у вигляді шарової діаграми. Нижній шар Data містить Entity Framework Core як ORM та SQLite як файлову СУБД. Над ним розташований шар Backend на базі ASP.NET Core MVC, що включає підсистему автентифікації Identity та бібліотеку Polly для обробки мережеских помилок. Верхній шар Frontend реалізовано через Razor Views, власну CSS дизайн-систему та бібліотеку FullCalendar.js. Окремий вертикальний блок External охоплює зовнішні залежності – Anthropic Claude API, Microsoft Playwright та AngleSharp – які взаємодіють із системою на рівні Backend. Така багаторівнева організація забезпечує слабку зв'язність між компонентами та чітке розмежування відповідальності, що спрощує супровід і подальше масштабування застосунку.

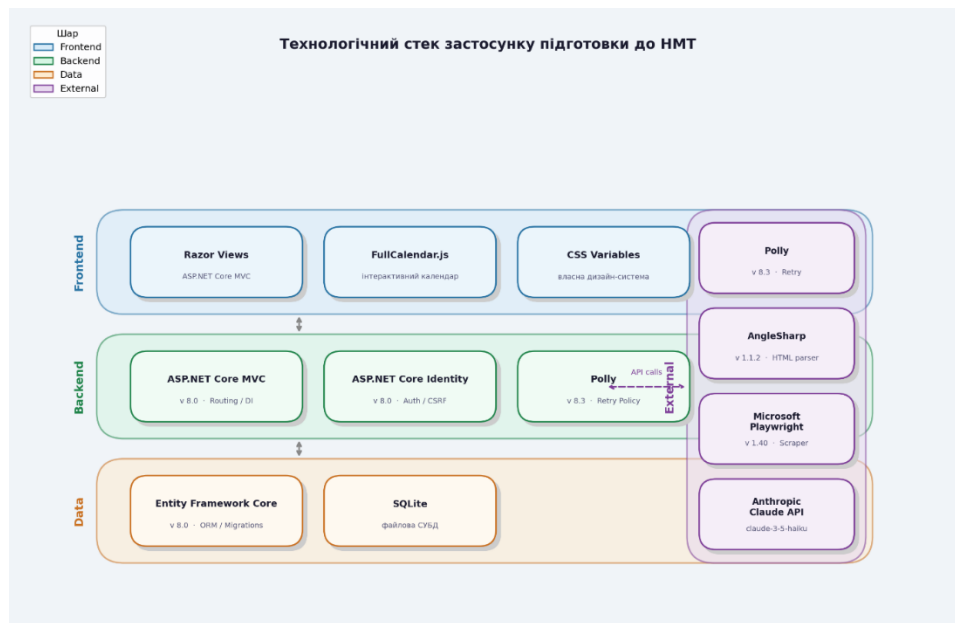


Рисунок 1.3 – Шарова діаграма технологічного стеку застосунку

1.4 Постановка задачі дослідження

На підставі аналізу предметної галузі та виявлених недоліків існуючих програмних аналогів сформульовано мету, об'єкт і предмет дослідження, визначено перелік задач та вимоги до розроблюваного застосунку.

Об'єкт дослідження – процес самостійної підготовки випускників шкіл до Національного мультидисциплінарного тесту засобами вебтехнологій.

Предмет дослідження – методи та засоби реалізації вебзастосунку з підсистемою автоматизованої перевірки результатів тестування, зокрема аналізу відповідей відкритого типу.

Мета роботи – розробка вебзастосунку для підготовки випускників шкіл до складання НМТ, що забезпечує доступ до банку реальних тестових завдань, перевірку результатів із розгорнутим зворотним зв'язком та аналіз відкритих відповідей засобами штучного інтелекту.

Для досягнення поставленої мети визначено такі задачі:

- провести аналіз предметної галузі та виконати огляд існуючих програмних аналогів;
- обґрунтувати вибір технологічного стеку та спроектувати архітектуру застосунку;
- розробити модуль автоматизованого збору навчального контенту з відкритих джерел;
- реалізувати підсистему тестування з підтримкою завдань вибіркового та відкритого типу;
- інтегрувати зовнішній API штучного інтелекту для оцінювання відповідей відкритого типу;
- розробити модуль персональної аналітики прогресу користувача;
- реалізувати модуль планування підготовки на основі результатів тестування;
- провести тестування та оцінити відповідність застосунку визначеним вимогам.

Функціональні вимоги до застосунку визначено на основі виявлених потреб цільової аудиторії та прогалин існуючих рішень:

- реєстрація користувача з вибором чотирьох предметів НМТ, які він планує складати;
- банк завдань, сформований на основі реальних варіантів іспитів, з підтримкою питань вибіркового та відкритого типу;
- режим тестування з відображенням питань по одному, фіксацією часу відповіді та відображенням пояснення після кожної відповіді;
- автоматизована перевірка відповідей вибіркового типу та AI-оцінювання відповідей відкритого типу із виставленням балу від 0 до 100 і текстовим коментарем;
- персональний дашборд із відображенням загального прогресу, динаміки результатів за останні 30 днів та переліку тем, що потребують додаткового опрацювання;

- модуль планування підготовки на основі інтерактивного календаря з можливістю ручного додавання завдань та автоматичного формування розкладу;

- перегляд повної історії тестових сесій із деталізацією результатів кожної.

Нефункціональні вимоги визначають обмеження та якісні характеристики системи:

- безпека (автентифікація на основі ASP.NET Core Identity, захист від CSRF-атак на всіх формах і AJAX-запитах, зберігання API-ключів поза репозиторієм);

- відмовостійкість (наявність резервного режиму MockAiService, що забезпечує повноцінну роботу застосунку за відсутності ключа зовнішнього API);

- кросбраузерність (коректне відображення інтерфейсу в актуальних версіях основних браузерів);

- простота розгортання (використання SQLite як файлової бази даних виключає необхідність встановлення окремого сервера СУБД);

- зручність використання (адаптивна верстка на основі Bootstrap гарантує коректне відображення інтерфейсу як на стаціонарних, так і на мобільних пристроях).

Сформульовані функціональні та нефункціональні вимоги є основою для проєктування архітектури застосунку. Вони визначають не лише перелік можливостей, які має реалізувати система, а й якісні характеристики, яким повинне відповідати підсумкове рішення. Обрана архітектура та технологічний стек, що детально розглядаються в наступному розділі, безпосередньо впливають із наведених вимог і покликані забезпечити їхнє виконання.

2 ТЕОРЕТИЧНІ ДОСЛІДЖЕННЯ

2.1 Архітектурні підходи до розробки

Вибір архітектурного підходу передувє безпосередній реалізації програмної системи й визначає її подальші властивості – зручність супроводу, придатність до тестування та здатність до розширення. Для вебзастосунку підготовки до НМТ це рішення ускладнене тим, що система об'єднує кілька різнорідних підсистем: банк тестових завдань, модуль проходження тестів, інтеграцію зовнішнього сервісу штучного інтелекту та аналітику навчального прогресу. Кожна з них має власну логіку опрацювання й відмінні вимоги до даних, тому архітектура повинна насамперед забезпечити розмежування відповідальностей між складниками системи та послабити зв'язність між ними.

Основою для застосунків такого класу слугує клієнт-серверна модель, у якій обчислювальне навантаження розподілене між двома сторонами взаємодії. Роль клієнта виконує браузер, що формує запити та відображає одержаний результат, тоді як сервер опрацьовує ці запити, виконує бізнес-логіку й звертається до сховища даних. Завдяки такому поділу дані зберігаються централізовано, робота системи не залежить від обчислювальних можливостей пристрою користувача, а оновлення не потребують втручання на боці клієнта. Для розглядуваної предметної галузі перелічені властивості є визначальними, оскільки відстеження динаміки результатів здобувача між окремими навчальними сесіями вимагає постійного зберігання стану на сервері.

Для організації серверної частини застосовано шаблон проєктування «Модель-Подання-Контролер» (Model-View-Controller, MVC). Він поділяє застосунок на три групи компонентів із чітко окресленими обов'язками: модель відповідає за дані та правила предметної галузі, подання формує інтерфейс користувача, а контролер приймає запити

й узгоджує роботу двох попередніх складників [1]. Принцип взаємодії компонентів за цим шаблоном показано на рисунку 2.1.

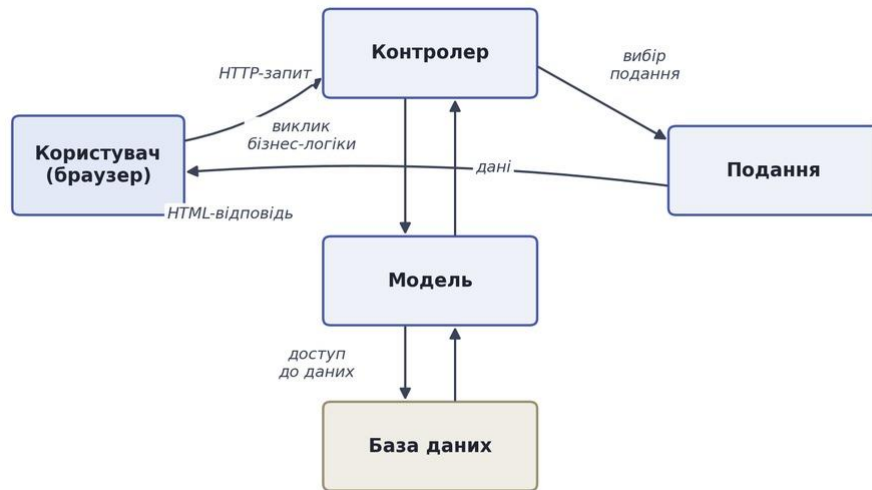


Рисунок 2.1 – Принцип роботи шаблону «Модель-Подання-Контролер»

Обробку запиту виконують у визначеній послідовності. Запит користувача надходить до контролера, який звертається до моделі для виконання потрібної бізнес-логіки, модель за потреби читає або змінює дані у сховищі та повертає результат. Після цього контролер обирає відповідне подання, що формує підсумкову HTML-сторінку для користувача. Такий поділ дає змогу змінювати інтерфейс незалежно від логіки опрацювання даних і навпаки, що спрощує супровід застосунку зі зростанням кількості функціональних модулів.

Поряд із поділом за шаблоном MVC у системі застосовано багаторівневу організацію коду, за якої компоненти згруповано в горизонтальні рівні відповідно до їхнього призначення. Кожен рівень взаємодіє лише із суміжним і не звертається безпосередньо до віддалених від нього рівнів. Структуру такої організації наведено на рисунку 2.2.

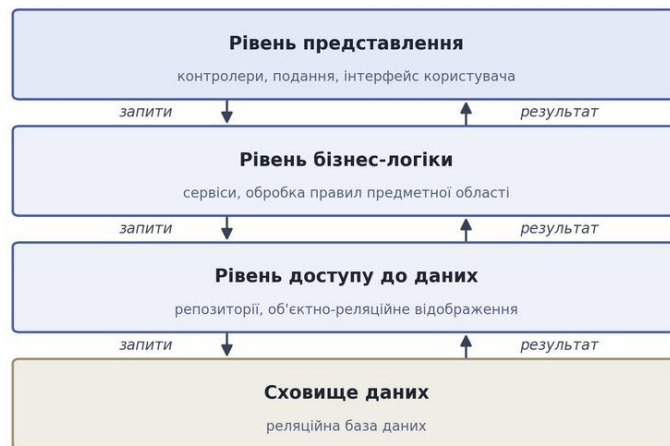


Рисунок 2.2 – Багаторівнева організація програмного коду

Верхній рівень представлення містить контролери та подання й відповідає за взаємодію з користувачем. Середній рівень бізнес-логіки об'єднує сервіси, що реалізують правила предметної галузі – формування тестової сесії, перевірку відповідей, обчислення показників прогресу. Нижній рівень доступу до даних інкапсулює роботу зі сховищем через об'єктно-реляційне відображення. У межах проєкту цей поділ відтворено у структурі спільної бібліотеки, де рівень даних, бізнес-логіку та інфраструктурні складники розміщено в окремих просторах імен. Такий підхід знижує зв'язність між рівнями: зміни у способі зберігання даних не зачіпають бізнес-логіку, доки залишається незмінним контракт взаємодії між ними.

Для відокремлення бізнес-логіки від конкретної технології доступу до даних застосовано патерн «Репозиторій». Він уводить проміжний шар абстракції, що подає роботу зі сховищем як набір методів для отримання та збереження сутностей, приховуючи деталі формування запитів [4]. Завдяки цьому сервіси оперують абстракціями репозиторіїв, а не безпосередньо засобами об'єктно-реляційного перетворювача, що спрощує заміну реалізації сховища та створює умови для модульного тестування бізнес-логіки з підставними реалізаціями репозиторіїв.

Зв'язування компонентів між собою реалізовано за принципом інверсії керування через механізм упровадження залежностей. Замість того щоб створювати залежності самостійно, компонент одержує їх ззовні у готовому вигляді, а за їх побудову та час життя відповідає вбудований контейнер служб платформи [6]. Такий підхід послаблює зв'язність між класами, полегшує заміну реалізацій і централізує налаштування системи. Зокрема, саме на рівні контейнера служб застосунок обирає основну або резервну реалізацію сервісу штучного інтелекту залежно від наявності ключа доступу до зовнішнього API.

Вибір конкретної платформи для серверної частини здійснено за результатами порівняння кількох поширених фреймворків веброзробки – ASP.NET Core, Django, Laravel і Spring Boot. Зіставлення проведено за критеріями системи типізації, наявності вбудованих засобів упровадження залежностей, рівня продуктивності, складу екосистеми та порога входу. Узагальнені результати наведено в таблиці A.2 (додаток A).

Зіставлення показало, що ASP.NET Core поєднує статичну типізацію, вбудовану підтримку впровадження залежностей і високий рівень продуктивності за помірного порога входу, тоді як решта розглянутих рішень поступаються принаймні за одним із цих критеріїв. З огляду на це для реалізації застосунку обрано поєднання клієнт-серверної моделі, шаблону MVC, багаторівневої організації коду, патерну «Репозиторій» та впровадження залежностей на платформі ASP.NET Core. Конкретне відображення цих підходів у структурі розроблюваної системи розглянуто в розділі 3.

2.2 Методи автоматизованого збору даних

Формування банку тестових завдань потребує значного обсягу структурованих даних – умов завдань, варіантів відповідей, правильних відповідей і тематичної належності. Ручне внесення такого матеріалу є

трудомістким і не забезпечує потрібної повноти, тому для наповнення сховища обрано метод автоматизованого збору даних із відкритого вебресурсу. Застосування цього методу пов'язане з низкою труднощів: вміст сторінок нерідко формується динамічно засобами клієнтських сценаріїв, з'єднання з вебсервером є нестабільним, а повторні запуски можуть призводити до дублювання вже зібраних завдань. Тому метод збору має передбачати засоби опрацювання динамічного вмісту, відновлення після тимчасових збоїв та усунення повторів.

У практиці вебскрапінгу розрізняють два основні підходи до отримання вмісту сторінок. Перший ґрунтується на надсиланні статичних HTTP-запитів із подальшим розбором одержаного HTML-документа, він є швидким і невибагливим до ресурсів, проте не відтворює вмісту, що формується клієнтськими сценаріями вже після завантаження сторінки. Другий підхід полягає в керуванні повноцінним браузером, який виконує ці сценарії та відображає сторінку так само, як у реального користувача. Такий спосіб опрацьовує динамічний вміст, однак потребує більше обчислювальних ресурсів і часу. Зіставлення підходів за основними критеріями наведено в таблиці 2.1.

Таблиця 2.1 – Порівняння підходів до отримання вмісту сторінок

Критерій	Статичні HTTP-запити	Браузерна автоматизація
Обробка JavaScript	Відсутня	Повна
Швидкість збору	Висока	Нижча
Споживання ресурсів	Низьке	Високе
Імітація дій користувача	Обмежена	Повна

Продовження таблиці 2.1

Критерій	Статичні HTTP-запити	Браузерна автоматизація
Стійкість до динамічного вмісту	Низька	Висока
Складність налаштування	Низька	Вища

Оскільки сторінки обраного джерела формують перелік завдань динамічно, для збору застосовано підхід на основі браузерної автоматизації. Керування браузером здійснюють у фоновому (headless) режимі, що дає змогу виконувати збір без графічного інтерфейсу [9]. Після повного завантаження сторінки її розмітку передають синтаксичному аналізатору, який будує об'єктну модель документа та надає засоби вибірки потрібних елементів за селекторами [14]. Поєднання цих засобів утворює гібридну схему: браузер відповідає за відтворення вмісту, а аналізатор – за вилучення структурованих даних.

Процес збору організовано як послідовний конвеєр, що повторюється для кожної сторінки в заданому діапазоні. На кожній ітерації браузер переходить за сформованою адресою, очікує на повне завантаження динамічного вмісту, після чого розмітку сторінки передають аналізатору для вилучення цільових елементів. Отримані дані проходять етап нормалізації та перевірки на повноту, що дозволяє відсіяти некоректні або порожні записи ще до збереження. Для підвищення стійкості збору між запитами витримують коротку паузу, а тимчасові мережеві помилки обробляють за допомогою політики повторних спроб, що запобігає втраті даних через нестабільність з'єднання. Узагальнену послідовність його етапів показано на рисунку 2.3.



Рисунок 2.3 – Узагальнена схема конвеєра автоматизованого збору

Робота конвеєра розпочинається із запуску браузера та переходу на чергову сторінку джерела з очікуванням завершення завантаження її вмісту. Далі перевіряють, чи містить сторінка завдання: за їх відсутності сторінку пропускають. У разі наявності завдань розмітку розбирають, вилучені дані нормалізують до єдиного внутрішнього формату, після чого відбирають лише нові завдання й зберігають їх у сховищі. По завершенні опрацювання система переходить до наступної сторінки, доки не буде вичерпано заданий діапазон.

Звернення до зовнішнього ресурсу супроводжуються тимчасовими збоями – короткочасною недоступністю сервера, перевищенням часу очікування чи розривами з'єднання. Для підвищення стійкості збору застосовано політику повторних спроб з експоненційно зростаючою затримкою [13]: після кожної невдалої спроби пауза перед наступною подвоюється, що зменшує навантаження на сервер і підвищує ймовірність

успішного відновлення. Тривалість затримки перед k -ю спробою визначають за формулою (2.1).

$$t_k = b \cdot 2^k, \quad (2.1)$$

де b – базова тривалість затримки, с;

k – порядковий номер повторної спроби;

N – гранична кількість повторних спроб.

Експоненційне зростання паузи запобігає повторному перевантаженню джерела однотипними запитами в умовах його тимчасової недоступності, тоді як обмеження граничної кількості спроб не дає процесу збору зациклитися на недосяжній сторінці.

Повторні запуски збору не повинні створювати дублікатів у сховищі. Для цього кожному завданню присвоюють зовнішній ідентифікатор, що однозначно визначає його в межах джерела й поєднує предмет, номер тесту та порядковий номер завдання. Перед збереженням перевіряють наявність завдання з таким ідентифікатором, а вже відомі завдання пропускають. Додатковою гарантією слугує обмеження унікальності на рівні бази даних, яке унеможлиблює внесення двох записів з однаковим ідентифікатором навіть за одночасного опрацювання.

Щоб уникнути надмірного навантаження на вебсервер джерела та наблизити поведінку збирача до дій реального користувача, між послідовними запитами витримують паузу випадкової тривалості в заданих межах. Рівномірний розподіл звернень у часі знижує ризик блокування з боку ресурсу й узгоджується з вимогами коректного використання відкритих джерел даних.

Отже, метод автоматизованого збору поєднує браузерну автоматизацію для опрацювання динамічного вмісту, синтаксичний розбір розмітки для вилучення структурованих даних, політику повторних спроб з експоненційною затримкою для стійкості до збоїв, дедуплікацію за

зовнішнім ідентифікатором та обмеження частоти звернень. Спроектовану на основі цього методу модель модуля збору розглянуто в розділі 3, а її програмну реалізацію – у розділі 4.

2.3 Методи перевірки відповідей

Завдання поділяють на два типи – вибіркові, що передбачають вибір однієї правильної відповіді із запропонованих варіантів, і відкриті, які вимагають самостійного формулювання розгорнутої відповіді. Перевірка результатів виконує одночасно технічну функцію зіставлення відповіді з правильною та педагогічну – фіксацію того, наскільки засвоєно матеріал. Метод перевірки має враховувати специфіку кожного типу завдань і зводити їх результати до єдиного підсумкового показника проходження тесту.

Перевірку вибірових завдань виконують зіставленням обраного користувачем варіанта з еталонною відповіддю. Перед порівнянням позначення варіанта нормалізують – приводять до єдиного вигляду незалежно від регістру та можливих відмінностей у записі літер «А», «Б», «В», «Г». Якщо нормалізоване позначення збігається з еталонним, відповідь визнають правильною, інакше – неправильною. Такий спосіб є однозначним і не потребує додаткового опрацювання, тому результат для вибірових завдань визначають безпосередньо в момент відповіді.

Відкриті завдання у тесті чергують із вибіровими: розгорнутої відповіді потребує кожне третє питання. Оскільки текст вільної відповіді не можна перевірити простим порівнянням, його передають на оцінювання засобами штучного інтелекту, що формує бал розуміння за стобальною шкалою, метод цього оцінювання докладно розглянуто в підрозділі 2.5. Для зведення такої відповіді до бінарного результату застосовують два правила: відповідь вважають правильною, якщо її нормалізований текст точно збігається з еталонною відповіддю або якщо бал розуміння не нижчий за 60.

Загальну логіку перевірки окремої відповіді з розгалуженням за типом завдання показано на рисунку 2.4.

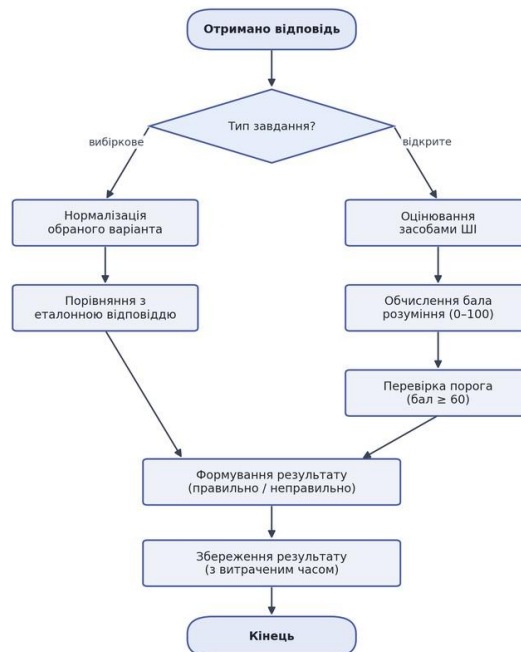


Рисунок 2.4 – Логіка перевірки відповіді за типом завдання

Після одержання відповіді спершу визначають тип завдання. Для вибіркового завдання виконують нормалізацію та порівняння з еталоном, для відкритого – оцінювання засобами штучного інтелекту з подальшою перевіркою порога. Обидві гілки сходяться на етапі формування результату відповіді, де встановлюють ознаку правильності, після чого результат зберігають разом із витраченим на відповідь часом.

Кожну відповідь фіксують окремим записом, що містить обраний варіант або текст відкритої відповіді, ознаку правильності, тип завдання та тривалість обмірковування. Збереження часу відповіді поряд із її змістом створює основу для подальшої аналітики, оскільки дає змогу враховувати не лише точність, а й темп роботи користувача.

Підсумковий результат сесії формують після завершення проходження тесту. Для цього підраховують кількість правильних

відповідей і загальну кількість опрацьованих завдань, обчислюють тривалість сесії та відсоток правильних відповідей за формулою (2.2).

$$P = \frac{n_c}{n} \cdot 100 \%, \quad (2.2)$$

де n_c – кількість правильних відповідей у сесії;

n – загальна кількість опрацьованих завдань у сесії.

Обчислений відсоток слугує узагальненою оцінкою результату сесії, на основі якої формується підсумкове повідомлення для користувача та поповнюються дані для аналітики прогресу.

Отже, метод перевірки опрацьовує вибірккові та відкриті завдання за різними правилами, але зводить їхні результати до спільної ознаки правильності та єдиного підсумкового показника сесії. Оцінювання відкритих відповідей засобами штучного інтелекту докладно розглянуто в підрозділі 2.5, а використання збережених результатів для аналітики – у підрозділі 2.6. Програмну реалізацію перевірки наведено в розділі 4.

2.4 Оцінювання відповідей засобами ШІ

Відповіді відкритого типу не можна перевірити прямим зіставленням рядків, оскільки правильна думка може бути сформульована багатьма способами. Завдання перевірки такої відповіді є задачею опрацювання природної мови: потрібно оцінити, наскільки зміст вільної відповіді відповідає очікуваному, а не збіг окремих слів. Для цього застосовано велику мовну модель, доступ до якої здійснюється через зовнішній програмний інтерфейс.

Велика мовна модель навчена на значних обсягах текстів і здатна оцінювати смислову відповідність вільного формулювання до еталонного змісту з урахуванням синонімії та перефразувань. Це є істотною перевагою над перевіркою за ключовими словами, яка не розпізнає правильну за

змістом, але інакше сформульовану відповідь. У роботі використано модель `claude-3-5-haiku`, що забезпечує прийнятний баланс між швидкістю генерації та якістю оцінювання текстових пояснень [8].

Якість оцінювання значною мірою залежить від того, як сформульовано звернення до моделі. Тому запит будують як структурований промпт, що містить роль і завдання для моделі, текст питання, еталонну відповідь, офіційне пояснення, відповідь учня та опис потрібного формату виводу. Склад вхідного промпту показано на рисунку 2.5.

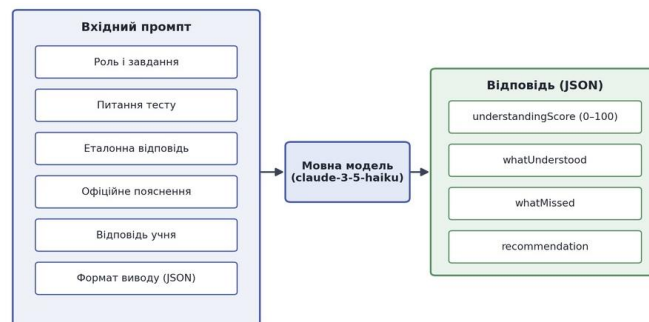


Рисунок 2.5 – Структура вхідного промпту та результату оцінювання

Щоб результат оцінювання можна було опрацювати програмно, модель інструктують повертати відповідь виключно у форматі JSON із чотирма полями: балом розуміння за стобальною шкалою, переліком засвоєного, переліком пропущеного та короткою рекомендацією щодо повторення. Одержаний текст очищають від можливого службового оформлення й розбирають у відповідну структуру даних, у разі помилки розбору оцінювання вважають невиконаним і відповідь опрацьовують без оцінки моделі.

Послідовність взаємодії сервісу оцінювання із зовнішнім інтерфейсом моделі показано на рисунку 2.6.

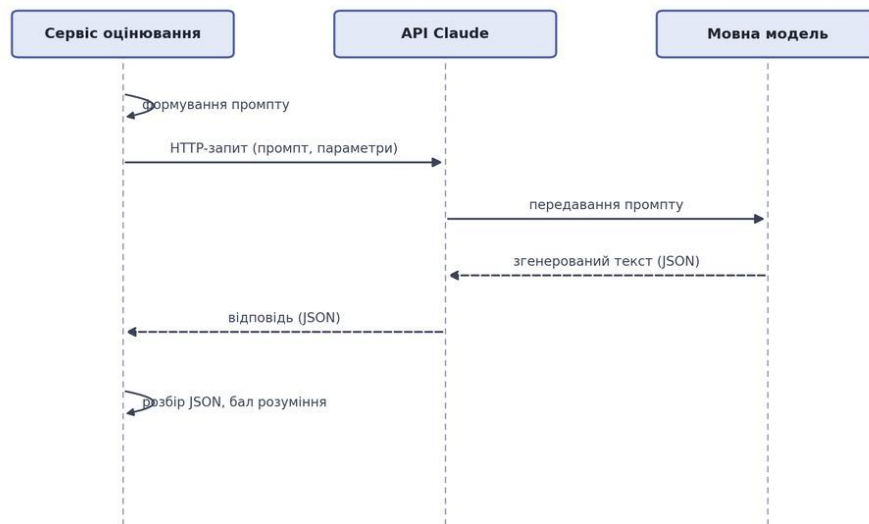


Рисунок 2.6 – Послідовність взаємодії із зовнішнім інтерфейсом моделі

Сервіс формує промпт і надсилає його у складі HTTP-запиту до інтерфейсу разом із параметрами генерації, зокрема обмеженням на обсяг відповіді. Інтерфейс передає промпт моделі, отримує згенерований текст і повертає його сервісу. На завершення сервіс розбирає одержаний JSON та вилучає бал розуміння й супровідні пояснення.

Бал розуміння змінюється в межах від 0 до 100 і відображає ступінь засвоєння матеріалу, проте для зведення результату до спільної з вибірковими завданнями ознаки правильності його зіставляють із пороговим значенням. Відповідь зараховують як правильну за виконання умови (2.3).

$$s \geq \theta, \quad (2.3)$$

де s – бал розуміння відповіді, виставлений мовною моделлю (0–100);

θ – пороговий бал зарахування відповіді ($\theta = 60$).

Якщо умову (2.3) виконано, відкрити відповідь зараховують як правильну, інакше – як неправильну. Поряд із цією бінарною ознакою

зберігають і сам бал розуміння разом із поясненнями, що дає змогу надати користувачеві розгорнутий зворотний зв'язок, а не лише факт правильності.

Окрім порогового критерію, передбачено швидку перевірку точного збігу. Якщо нормалізований текст відповіді здобувача – без урахування регістру, пробілів і розділових знаків – збігається з еталонною відповіддю, її зараховують як правильну незалежно від бала мовної моделі. Перевірку збігу виконують першою, що підвищує стійкість оцінювання для коротких однослівних відповідей і забезпечує правильний результат навіть у резервному режимі, коли бал розуміння є наближеним.

Для забезпечення відмовостійкості передбачено резервний режим оцінювання, який вмикається за відсутності доступу до зовнішнього інтерфейсу. У цьому режимі бал і пояснення формуються спрощеним вбудованим алгоритмом, завдяки чому застосунок залишається працездатним навіть без звернень до зовнішньої моделі.

Отже, метод оцінювання відкритих відповідей засобами штучного інтелекту поєднує структурований промпт, отримання структурованого результату та пороговий критерій переведення бала в ознаку правильності, а також надає змістовний зворотний зв'язок щодо засвоєння теми. Конкретну структуру промпту та схему відповіді розглянуто в розділі 3, а програмну реалізацію інтеграції – у розділі 4.

2.5 Методи аналітики прогресу

Накопичені результати відповідей самі собою не дають користувачеві уявлення про рівень підготовки, тому їх перетворюють на показники прогресу засобами статистичного опрацювання [12]. Аналітика розв'язує три завдання: відображає загальний рівень засвоєння кожного предмета, виявляє теми, що потребують додаткового опрацювання, і простежує динаміку результатів у часі. Усі показники обчислюють на основі збережених записів відповідей із зазначенням теми, правильності та часу.

Ключовим завданням аналітики є виявлення слабких тем. Для цього відповіді групують за темами завдань і для кожної теми обчислюють частку помилок як відношення хибних відповідей до загальної кількості спроб за формулою (2.4).

$$e_t = 1 - \frac{c_t}{n_t}, \quad (2.4)$$

де e_t – частка помилок за темою t ;

c_t – кількість правильних відповідей за темою t ;

n_t – загальна кількість відповідей за темою t .

Окрема помилкова відповідь ще не свідчить про прогалину в темі, тому слабкими вважають лише ті теми, за якими накопичено достатньо спроб і водночас частка помилок перевищує встановлений поріг. Відповідну умову подано формулою (2.5).

$$e_t \geq \tau, \quad n_t \geq n_{min}, \quad (2.5)$$

де τ – пороговий рівень частки помилок ($\tau = 0,4$);

n_{min} – мінімальна кількість спроб за темою ($n_{min} = 2$).

Тему вважають слабкою лише за одночасного виконання обох умов: коли частка помилкових відповідей e_t досягає або перевищує встановлений поріг τ , а кількість зафіксованих спроб за цією темою n_t є не меншою за n_{min} . Вимога щодо мінімальної кількості спроб запобігає передчасним хибним висновкам на основі поодиноких випадкових відповідей, для яких статистична оцінка частки помилок є ненадійною.

Значення $\tau = 0,4$ і $n_{min} = 2$ дібрано емпірично як компроміс між чутливістю виявлення прогалин і стійкістю оцінки до випадкових коливань.

Послідовність виявлення слабких тем показано на рисунку 2.7.



Рисунок 2.7 – Послідовність виявлення слабких тем

Опрацювання розпочинається з групування відповідей за темами. Для кожної теми перевіряють, чи накопичено щонайменше дві спроби. Теми з меншою кількістю спроб відкидають як статистично незначущі. Для решти тем обчислюють частку помилок і порівнюють її з пороговим рівнем: теми, що перевищують поріг, додають до переліку слабких. Сформований перелік упорядковують за спаданням частки помилок і відбирають теми з найвищими показниками, які першочергово потребують повторення.

Поряд із пошуком слабких тем аналітика відображає загальний рівень засвоєння кожного предмета. Його обчислюють як відсоток правильних відповідей серед усіх наданих з відповідного предмета за формулою (2.6).

$$P_s = \frac{c_s}{n_s} \cdot 100 \%, \quad (2.6)$$

де P_s – відсоток засвоєння предмета s ;

c_s – кількість правильних відповідей з предмета;

n_s – загальна кількість відповідей з предмета.

Для відстеження динаміки результати агрегують за окремими днями впродовж останнього місяця. Для кожного дня визначають кількість опрацьованих завдань і відсоток правильних відповідей, що дає змогу побудувати графік зміни активності та результативності в часі й помітити періоди спаду чи зростання.

Додатковим показником залученості є серія активності – кількість днів поспіль, у які користувач проходив тести. Її обчислюють, послідовно перебираючи дати завершених сесій від поточного дня у минуле, доки ланцюг суміжних днів не переривається. Цей показник має мотиваційне призначення й заохочує до регулярних занять.

Отже, методи аналітики поєднують обчислення частки помилок за темами з пороговим критерієм виявлення слабких місць, оцінювання засвоєння предметів, агрегацію результатів за днями та визначення серії активності. Сформовані показники є основою для персональних рекомендацій і планування підготовки, метод якого розглянуто в підрозділі 2.6, а їх відображення на дашборді – у розділі 4.

2.6 Метод формування розкладу

Підготовка до НМТ є тривалим процесом, ефективність якого значною мірою залежить від рівномірного розподілу занять у часі та між предметами. Тому до складу системи входить метод автоматичного формування розкладу підготовки, який враховує доступний час користувача, рівномірно охоплює всі обрані предмети та поєднує опрацювання матеріалу з його перевіркою.

Основою методу є каталог тем, укладений окремо для кожного предмета. Кожна тема каталогу пов'язує навчальний матеріал – статтю для

опрацювання – із тестом на тій самій темі, завдяки чому за читанням матеріалу одразу йде його перевірка. Назва теми в каталозі збігається з назвою теми у банку завдань, що гарантує наявність тестових питань для відповідної перевірки.

Щоб підготовка не зосереджувалася на одному предметі, теми відбирають у круговому порядку (round-robin): на кожному кроці до спільної черги додають по одній темі з кожного предмета по черзі. Унаслідок цього предмети чергуються рівномірно, а тема змінюється від однієї пари завдань до іншої, що урізноманітнює заняття й охоплює ширше коло матеріалу.

Користувач задає дні тижня для занять і кількість годин на день. Час занять поділяють на однакові півгодинні слоти. Кількість слотів на день роблять парною, щоб пара «стаття – тест» завжди вміщувалася в межах одного дня й не розривалася на межі діб. Кількість слотів на день визначають за формулою (2.7).

$$s_d = 2[h], \quad (2.7)$$

де s_d – кількість 30-хвилинних слотів на день (парна);

h – задана користувачем кількість годин занять на день.

Слоти впорядковують за датами лише для дозволених днів у межах заданого горизонту планування. Далі їх перебирають парами: у перший слот пари розміщують статтю з обраної теми, у другий – тест на тій самій темі. Перед додаванням завдання перевіряють, чи не заплановано вже подібного на цю дату, що унеможлиблює появу дублікатів за повторного формування розкладу.

Узагальнену послідовність формування розкладу показано на рисунку 2.8.



Рисунок 2.8 – Узагальнена схема алгоритму формування розкладу

Спочатку формують каталог тем і будують кругову чергу, у якій чергуються предмети. Потім визначають перелік доступних слотів на основі обраних днів і кількості годин на день. Слоти перебирають парами, і для кожної пари беруть наступну тему з черги: у перший слот планують опрацювання статті, у другий – тест на тій самій темі. Перебір триває, доки залишаються вільні слоти.

Отже, метод формування розкладу поєднує каталог тем із прив'язкою статті до тесту, кругову ротацію предметів і розміщення пар завдань у парній кількості слотів. Це забезпечує рівномірний, збалансований за предметами план підготовки, узгоджений із банком завдань. Поряд із плановим розкладом система додає до календаря й рекомендовані завдання, сформовані за результатами аналітики слабких тем (підрозділ 2.5). Модель цього методу розглянуто в розділі 3, а її програмну реалізацію – у розділі 4.

2.7 Методи захисту даних

Застосунок зберігає персональні дані користувачів – облікові записи та результати проходження тестів, тому його розроблення передбачає засоби захисту цих даних від неавторизованого доступу й забезпечення їх цілісності та доступності. Захист реалізовано сукупністю методів: автентифікацією користувачів, безпечним зберіганням паролів, розмежуванням доступу до даних, захистом від підроблених запитів і забезпеченням відмовостійкості.

Доступ до персональних функцій надають лише після автентифікації. Її побудовано на основі вбудованої підсистеми керування обліковими записами платформи, яка забезпечує реєстрацію за електронною адресою та паролем, вхід у систему і підтримання сеансу за допомогою захищених кук [1]. Сторінки, що працюють із персональними даними, позначено як доступні лише аутентифікованим користувачам, тому спроба відкрити їх без входу перенаправляє на форму авторизації.

Паролі користувачів не зберігають у відкритому вигляді. Замість самого пароля у сховищі утримують його одностороннє перетворення – хеш, обчислений з додаванням випадкової солі за алгоритмом багаторазового хешування [1]. Завдяки односторонності перетворення відновити початковий пароль із хешу практично неможливо, а індивідуальна сіль унеможлиблює однакові хеші для однакових паролів різних користувачів. Додатково до паролів висувають вимоги щодо мінімальної довжини та складу, що ускладнює їх добір.

Розмежування доступу не обмежується фактом входу: кожен користувач може працювати лише з власними даними. Результати сесій і відповіді прив'язано до ідентифікатора власника, а вибірки даних завжди обмежують записами поточного користувача. Унаслідок цього навіть аутентифікований користувач не має доступу до результатів інших осіб.

Операції, що змінюють стан системи, виконують через форми, які надсилають дані на сервер. Щоб запобігти підробленим міжсайтовим запитам, такі форми супроводжують антипідробним токеном, який сервер перевіряє перед опрацюванням запиту [6]. Відсутність або невідповідність токена призводить до відхилення запиту, що захищає від виконання дій від імені користувача без його відома.

Окремою складовою захисту є відмовостійкість, що забезпечує доступність системи за збоїв допоміжних служб. За недоступності зовнішнього сервісу оцінювання застосунок переходить у резервний режим і продовжує роботу зі спрощеним оцінюванням, а тимчасові збої під час звернень до зовнішніх ресурсів опрацьовують повторними спробами. Таким чином, відмова окремого складника не призводить до зупинки роботи всієї системи.

Додатковим рівнем захисту є безпечне зберігання конфіденційних параметрів конфігурації. Ключі доступу до зовнішніх сервісів та рядок підключення до бази даних не потрапляють до системи контролю версій, а зчитуються з оточення під час запуску застосунку, що унеможлиблює їх розкриття у вихідному коді. Передавання даних між клієнтом і сервером здійснюють захищеним протоколом, що шифрує трафік і запобігає його перехопленню чи підміні. Окрему увагу приділено перевірці вхідних даних на боці сервера: усі дані, що надходять від користувача, проходять валідацію перед опрацюванням, що знижує ризик некоректних станів та потенційних ін'єкцій.

Узагальнення застосованих механізмів захисту та загроз, яким вони протидіють, наведено в таблиці А.3 (додаток А).

Отже, захист даних у застосунку забезпечують узгоджені між собою методи: автентифікація та авторизація розмежовують доступ, одностороннє хешування захищає паролі, антипідробні токени запобігають підробленим запитам, а резервні режими й повторні спроби підтримують доступність системи. Програмну реалізацію цих механізмів розглянуто в розділі 4.

3 ПРОЄКТУВАННЯ ТА РОЗРОБКА МОДЕЛІ

3.1 Загальна архітектура системи

У цьому розділі розглянуто проєктну реалізацію підходів, обґрунтованих у розділі 2. Систему побудовано як рішення з трьох проєктів, кожен з яких відповідає за окремий аспект роботи: вебзастосунок для взаємодії з користувачем, спільну бібліотеку предметної логіки та доступу до даних і окрему утиліту для збору тестових завдань. Такий поділ дає змогу повторно використовувати спільний код і розвивати складники незалежно один від одного.

Структуру рішення та зв'язки між його складниками наведено на рисунку 3.1.

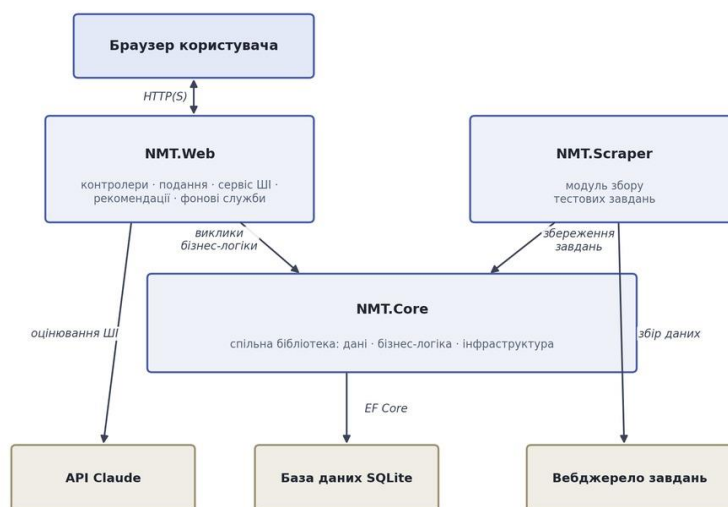


Рисунок 3.1 – Компонентна структура рішення

Вебзастосунок NMT.Web і утиліта збору NMT.Scraper залежать від спільної бібліотеки NMT.Core, яка містить моделі предметної галузі, доступ до даних і бізнес-логіку. Вебзастосунок взаємодіє з браузером користувача за протоколом HTTP, звертається до зовнішнього інтерфейсу мовної моделі

для оцінювання відкритих відповідей і використовує бібліотеку NMT.Core для всіх операцій із даними. Утиліта збору звертається до зовнішнього вебджерела й зберігає одержані завдання через ту саму бібліотеку, а доступ до бази даних SQLite в обох випадках забезпечує об'єктно-реляційний перетворювач [1]. Розподіл обов'язків між проєктами подано в таблиці 3.1.

Таблиця 3.1 – Розподіл обов'язків між проєктами рішення

Проект	Призначення	Ключові складники
NMT.Web	Презентація та взаємодія з користувачем	Контролери, Razor-подання, сервіс III, рекомендації, фонові служби
NMT.Core	Спільна предметна логіка та доступ до даних	Моделі, репозиторії, контекст БД, сервіси, інтерфейси, DTO
NMT.Scraper	Автоматизований збір тестових завдань	Модуль скрапінгу на основі Playwright, AngleSharp та Polly

Усередині спільної бібліотеки та вебзастосунку код поділено на рівні відповідно до призначення. Поділ на рівні показано на рисунку 3.2.

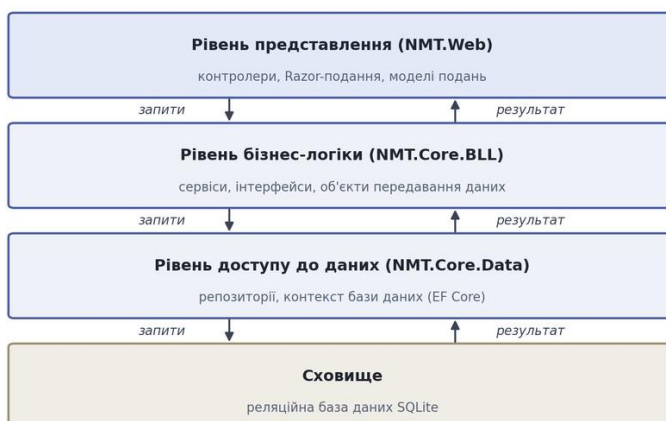


Рисунок 3.2 – Багаторівнева організація коду системи

Рівень представлення зосереджено у вебзастосунку й утворюють контролери, Razor-подання та моделі подань. Рівень бізнес-логіки об'єднує сервіси, їхні інтерфейси та об'єкти передавання даних, а рівень доступу до даних – репозиторії й контекст бази даних. Кожен рівень звертається лише до суміжного нижчого. Зв'язування складників виконує контейнер залежностей, у якому зареєстровано репозиторії та сервіси, при цьому інтерфейси сервісів оголошено в спільній бібліотеці, а реалізації, що потребують зовнішніх служб, у вебзастосунку, що зберігає незалежність бізнес-логіки від технічних деталей.

Для ілюстрації взаємодії складників розглянуто опрацювання запиту на прикладі надсилання відкритої відповіді. Послідовність такого опрацювання показано на рисунку 3.3.

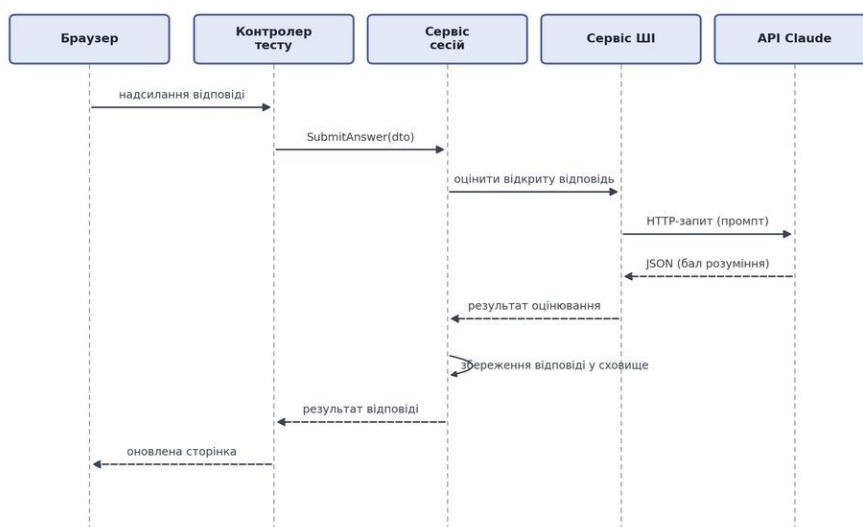


Рисунок 3.3 – Послідовність опрацювання запиту з відкритою відповіддю

Браузер надсилає відповідь контролеру тесту, який передає її сервісу сесій. Оскільки відповідь відкрита, сервіс сесій звертається до сервісу штучного інтелекту, що формує запит до зовнішнього інтерфейсу та одержує бал розуміння. Отриманий результат сервіс сесій зберігає у сховищі через репозиторій, після чого повертає підсумок контролеру, а

той – оновлену сторінку браузеру. Така послідовність наочно демонструє проходження запиту крізь усі рівні системи.

Отже, описана архітектура конкретизує поєднання клієнт-серверної моделі, шаблону MVC, багаторівневої організації коду, патерну «Репозиторій» та впровадження залежностей у вигляді трьох узгоджених проєктів. Деталізацію окремих складників – модель даних, структуру класів і сценарії використання – розглянуто в наступних підрозділах.

3.2 Модель даних застосунку

Модель даних є підґрунтям усієї системи, оскільки визначає, які відомості зберігаються та як вони пов'язані між собою. Її спроектовано як сукупність взаємопов'язаних сутностей, які за допомогою об'єктно-реляційного відображення подаються таблицями реляційної бази даних [2]. Основу моделі утворюють п'ять сутностей: користувачі, завдання, сесії тестування, відповіді та завдання календаря.

Концептуальну модель даних у вигляді діаграми «сутність – зв'язок» наведено на рисунку 3.4.

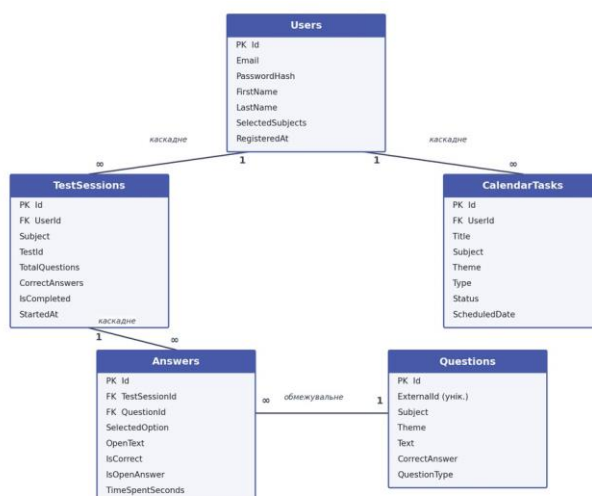


Рисунок 3.4 – Модель даних у вигляді діаграми «сутність – зв'язок»

Сутність Users зберігає облікові записи та налаштування користувачів, зокрема перелік обраних предметів. Сутність Questions становить банк тестових завдань із текстом, варіантами відповідей, правильною відповіддю й тематичною належністю. Сесію проходження тесту описує сутність TestSessions, що накопичує підсумкові показники, а кожну окрему відповідь у межах сесії – сутність Answers. Заплановані та рекомендовані заняття зберігає сутність CalendarTasks. Призначення сутностей узагальнено в таблиці 3.2.

Таблиця 3.2 – Сутності моделі даних та їхнє призначення

Сутність	Призначення
Users	Облікові записи користувачів та їхні налаштування
Questions	Банк тестових завдань, зібраних із зовнішнього джерела
TestSessions	Сесії проходження тестів із підсумковими результатами
Answers	Окремі відповіді користувача в межах сесії
CalendarTasks	Заплановані та рекомендовані завдання календаря

Між сутностями встановлено чотири зв'язки типу «один до багатьох». Користувач пов'язаний зі своїми сесіями тестування та завданнями календаря, а сесія – з наданими в ній відповідями. Кожна відповідь додатково посиляється на завдання, якому вона відповідає. Для зв'язків, підпорядкованих користувачеві та сесії, застосовано каскадне видалення: видалення користувача прибирає його сесії, відповіді та заплановані завдання. Натомість для зв'язку відповіді із завданням встановлено обмежувальне видалення, яке не дозволяє видалити завдання, доки на нього посиляються збережені відповіді, що захищає історію проходження від порушення цілісності.

Кожній сутності відповідає таблиця бази даних, а її властивостям – стовпці з відповідними обмеженнями довжини. Цілочислові ключі сутностей генеруються базою даних автоматично, тоді як ідентифікатор користувача успадковано від підсистеми керування обліковими записами. Переліки варіантів відповідей і результати оцінювання, що мають змінну структуру, зберігають як текст у форматі JSON, а перелічувані значення типу та стану завдання – у вигляді цілих чисел. Таблиці підсистеми автентифікації інтегровано в ту саму базу даних.

Щоб пришвидшити типові запити, у моделі визначено індекси. Унікальний індекс за зовнішнім ідентифікатором завдання забезпечує неможливість повторного збереження того самого завдання, а складені індекси прискорюють вибірки за предметом і роком, за користувачем і предметом та за користувачем і датою. Перелік індексів наведено в таблиці 3.3.

Таблиця 3.3 – Індеси бази даних

Сутність	Поля індексу	Тип
Questions	ExternalId	Унікальний
Questions	Subject, Year	Складений
Questions	TestId	Звичайний
TestSessions	UserId, Subject	Складений
CalendarTasks	UserId, ScheduledDate	Складений

Отже, спроектована модель даних поєднує нормалізоване подання основних сутностей зі збереженням гнучких за структурою даних у форматі JSON, чим забезпечує підтримку всіх операцій системи – від наповнення банку завдань до накопичення результатів і планування. Фізичну схему бази даних за цією моделлю формує об'єктно-реляційний перетворювач, а її реалізацію розглянуто в розділі 4.

3.3 Структура класів та сервісів

Класи системи згруповано відповідно до багаторівневої архітектури: рівень бізнес-логіки утворюють сервіси, рівень доступу до даних – репозиторії, а обмін даними між рівнями відбувається через об'єкти передавання даних. Кожен сервіс і репозиторій реалізує відповідний інтерфейс, тому складники взаємодіють через абстракції, а не конкретні класи, що послаблює зв'язність і спрощує заміну реалізацій.

Структуру основних класів і залежності між ними наведено на рисунку 3.5.

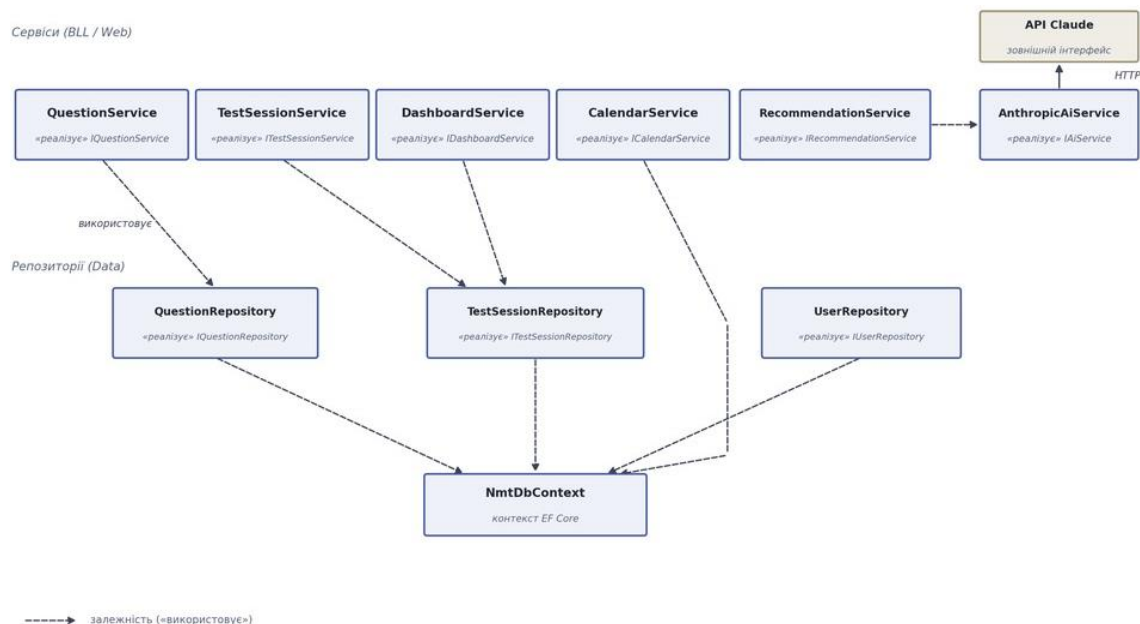


Рисунок 3.5 – Структура класів бізнес-логіки та доступу до даних

Бізнес-логіку реалізують сервіси, кожен з яких відповідає за окрему ділянку роботи. QuestionService надає тестові питання та теми, TestSessionService проводить сесії й перевіряє відповіді, DashboardService формує показники прогресу, а CalendarService планує заняття та веде календар. Оцінювання відкритих відповідей і формування рекомендацій винесено у сервіси вебзастосунку – AnthropicAiService та

RecommendationService, оскільки вони потребують доступу до зовнішніх служб і конфігурації. Контролери звертаються до сервісів лише через їхні інтерфейси, а конкретні реалізації надає контейнер залежностей [1].

Призначення сервісів та їхні залежності узагальнено в таблиці 3.4.

Таблиця 3.4 – Сервіси бізнес-логіки та їхні залежності

Сервіс	Призначення	Залежності
QuestionService	Надання тестових питань і тем	IQuestionRepository
TestSessionService	Проведення сесій і перевірка відповідей	ITestSessionRepository, IQuestionRepository, IAIService
DashboardService	Формування показників прогресу	ITestSessionRepository, IQuestionRepository
CalendarService	Планування та ведення календаря	NmtDbContext
RecommendationService	Формування персональних рекомендацій	IAIService, IDashboardService
AnthropicAiService	Оцінювання відповідей мовною моделлю	Зовнішній API Claude

Доступ до даних інкапсульовано в репозиторіях QuestionRepository, TestSessionRepository та UserRepository, кожен з яких реалізує власний інтерфейс і звертається до контексту бази даних. Завдяки цьому сервіси оперують методами репозиторіїв на кшталт отримання, додавання чи збереження сутностей, не залежачи від деталей формування запитів до бази даних.

Для передавання даних між рівнями замість сутностей бази даних використано окремі об'єкти передавання даних. Так, питання для проходження тесту подає `TestQuestionDto`, результат сесії – `TestResultDto`, показники дашборду – `DashboardDto`, а заплановане завдання – `CalendarTaskDto`. Такий підхід відокремлює подання від внутрішньої моделі даних: зміни у структурі сутностей не зачіпають інтерфейс користувача, доки незмінними залишаються об'єкти передавання.

Отже, структура класів конкретизує патерн «Репозиторій» та впровадження залежностей: сервіси й репозиторії розділено за відповідальністю, пов'язано через інтерфейси й об'єкти передавання даних, що забезпечує низьку зв'язність і придатність до тестування. Програмну реалізацію цих класів розглянуто в розділі 4.

3.4 Проєктування сценаріїв

Щоб формалізувати функційну поведінку системи, обґрунтовану в попередніх підрозділах, її подано у вигляді сукупності варіантів використання та сценаріїв їх перебігу [3]. Такий опис визначає, хто взаємодіє із системою та які саме дії виконує, і слугує підставою для подальшої програмної реалізації. Спершу виокремлено дійових осіб (акторів), а далі – доступні їм варіанти використання й типові сценарії роботи.

У системі розрізняють двох безпосередніх акторів-користувачів та двох зовнішніх. Актор «Гість» відповідає відвідувачеві, який ще не пройшов автентифікацію, тож йому доступні лише реєстрація та вхід. Актор «Здобувач» – це автентифікований користувач із роллю студента, якому відкрито всі навчальні можливості: проходження тестів, перегляд аналітики та планування занять. Зовнішніми акторами є «Сервіс ІІІ» – зовнішній інтерфейс мовної моделі, що залучається для оцінювання відкритих відповідей і формування рекомендацій, та «Планувальник» –

фонова служба, яка періодично ініціює оновлення рекомендацій без участі користувача.

Сукупність акторів і варіантів використання, а також зв'язки між ними наведено на рисунку 3.6.



Рисунок 3.6 – Діаграма варіантів використання системи

Гостю доступні два варіанти використання – реєстрація з вибором переліку предметів та вхід у систему. Здобувач керує власним профілем, налаштовує й запускає тести, проходить їх, переглядає результати та історію тестувань, аналізує прогрес на дашборді, оновлює рекомендації, а також веде календар занять. Варіант використання «Пройти тест» через відношення включення («include») залучає окремий варіант «Оцінити відкриту відповідь», що виконується лише для відкритих питань і потребує звернення до сервісу ШІ. Формування рекомендацій так само спирається на сервіс ШІ й може ініціюватися як здобувачем уручну, так і планувальником автоматично. Натомість автоматичне формування розкладу виконує сама

система за визначеним алгоритмом, без звернення до мовної моделі, тому цей варіант із зовнішніми акторами не пов'язаний.

Призначення основних варіантів використання та відповідних їм акторів узагальнено в таблиці А.4 (додаток А).

Кожен варіант використання розгортається в один або кілька сценаріїв упорядкованих послідовностей дій актора й системи, серед яких вирізняють основний (успішний) та альтернативні. Центральним для системи є сценарій проходження тесту, докладний перебіг якого у вигляді діаграми діяльності наведено на рисунку 3.7.

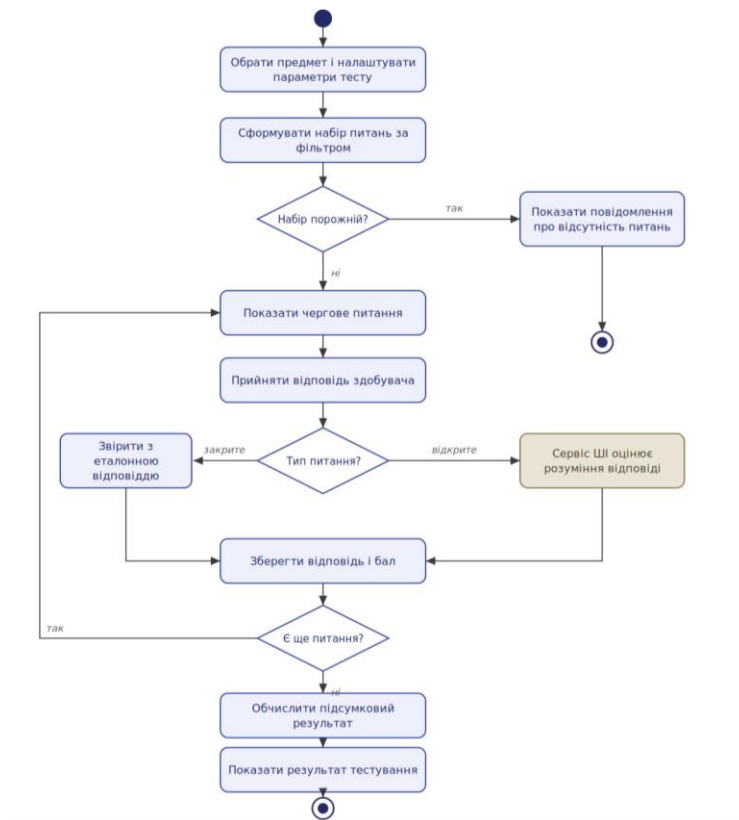


Рисунок 3.7 – Діаграма діяльності сценарію проходження тесту

Основний сценарій починається з вибору предмета та налаштування параметрів тесту – теми, року або кількості питань. За цими параметрами система формує набір питань. Якщо жодного питання не дібрано, виконання переходить до альтернативної гілки з повідомленням про відсутність даних

і завершується. В успішному перебігу система послідовно показує питання та приймає відповідь здобувача, після чого залежно від типу питання опрацьовує її по-різному: відповідь на закриті питання звіряється з еталоном безпосередньо системою, а відкрита відповідь передається сервісу ІІІ, який оцінює рівень розуміння. Збережену відповідь із виставленим балом ураховують у результатах сесії. Доки лишаються неопрацьовані питання, цикл повторюється. Після опрацювання останнього питання система обчислює підсумковий результат і показує його здобувачеві.

Отже, побудована модель сценаріїв формалізує функційну поведінку системи: визначає акторів, перелік варіантів використання та перебіг ключових сценаріїв, чітко розмежовуючи дії, що їх виконує сама система, і ті, що потребують зовнішнього сервісу ІІІ. Програмну реалізацію описаних сценаріїв розглянуто в розділі 4.

3.5 Модель збору завдань

Якість підготовки до тестування безпосередньо залежить від обсягу та різноманітності банку завдань, тому наповнювати його вручну недоцільно. Натомість передбачено окремий модуль автоматизованого збору завдань (NMT.Scraper), який видобуває тестові питання з відкритого вебархіву завдань ЗНО та НМТ і зберігає їх у спільну базу даних. Модуль реалізовано як окрему консольну програму, що використовує ту саму модель даних і той самий рівень доступу до даних, що й основний застосунок, завдяки чому зібрані завдання одразу стають придатними для використання.

Процес збору організовано як послідовний конвеєр опрацювання даних, у якому кожен етап виконує окремий компонент. Загальну схему взаємодії компонентів модуля з вебсайтом-джерелом і базою даних наведено на рисунку 3.8.

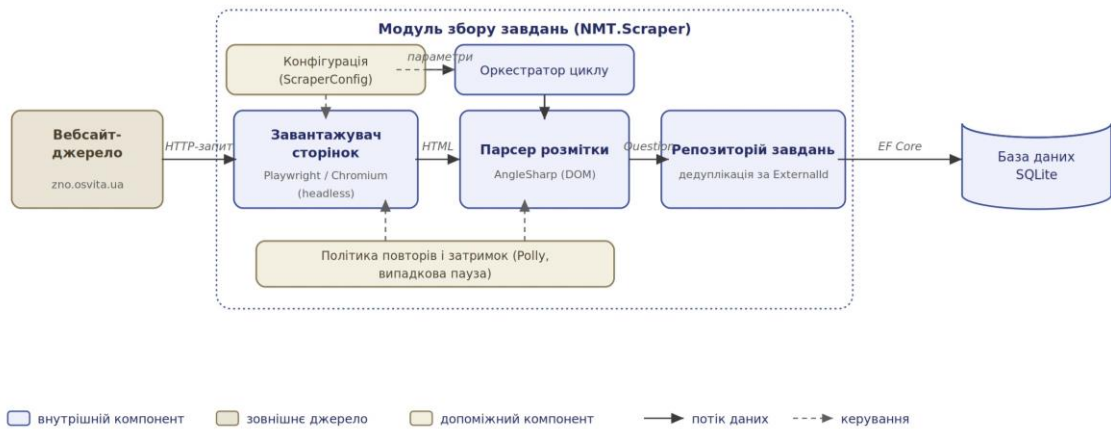


Рисунок 3.8 – Загальна схема процесу збору завдань

Параметри збору (адресу джерела, діапазон ідентифікаторів тестів, перелік предметів, межі затримок і кількість повторів) винесено в конфігурацію, що дає змогу змінювати поведінку модуля без перекомпіляції. Оркестратор циклу за цими параметрами по чергові формує адреси сторінок і керує опрацюванням. Завантажувач сторінок на основі бібліотеки Playwright відкриває сторінку у фоновому браузері Chromium і повністю відтворює її вміст, зокрема елементи, що формуються сценаріями на боці клієнта. Отриману розмітку передають парсеру на основі бібліотеки AngleSharp, який за CSS-селекторами виловлює зі сторінки структуровані завдання. Сформовані об'єкти завдань надходять до репозиторію, що перевіряє їх на повторюваність і зберігає нові записи в базу даних SQLite. Стійкість збору забезпечує допоміжний компонент політики повторів на основі бібліотеки Polly: у разі тимчасової помилки запит повторюється з наростаючою затримкою, а між зверненнями до сайту витримується випадкова пауза, що знижує навантаження на джерело.

Зі сторінки кожного завдання модуль виловлює набір полів, які повністю описують питання та потрібні для його подальшого використання в тестуванні й оцінюванні. Перелік цих полів, їх джерела на сторінці та призначення узагальнено в таблиці А.5.

Усі зібрані завдання належать до закритого типу – із переліком варіантів відповіді та однією правильною, тож їх можна оцінювати автоматично, звіряючи з еталоном. Послідовність дій модуля під час збору, разом із перевітками доступності сторінки та повторюваності завдань, подано у вигляді діаграми діяльності на рисунку 3.9.

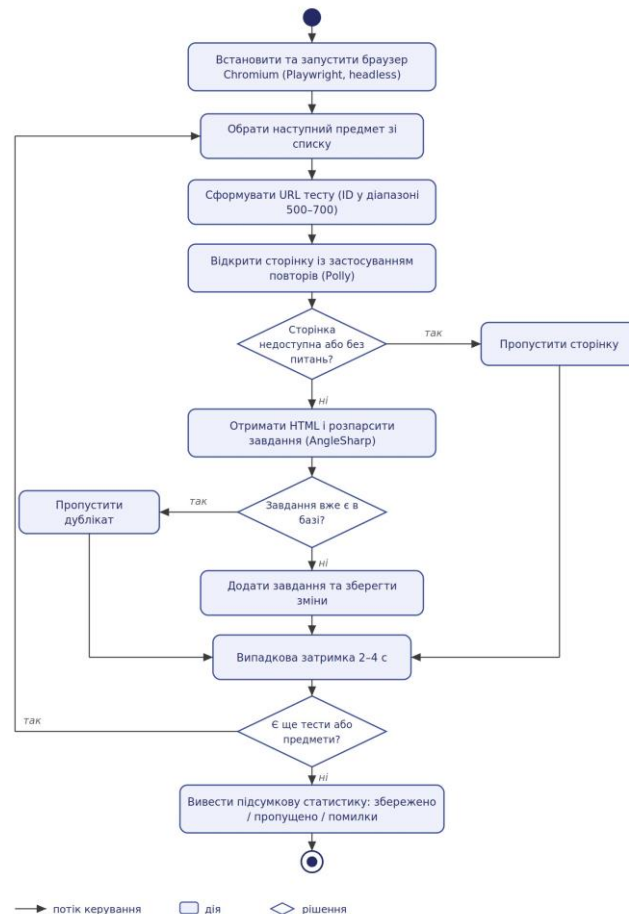


Рисунок 3.9 – Діаграма діяльності процесу збору завдань

Робота модуля починається з установаження та запуску фонового браузера. Далі для кожного предмета зі списку послідовно перебираються номери тестів із заданого діапазону: за ними формується адреса сторінки, яку відкривають із застосуванням політики повторів. Якщо сторінка недоступна або не містить завдань, її пропускають і переходять до наступної. Інакше отриману розмітку розбирають на окремі завдання, і

кожне з них перевіряють на наявність у базі за зовнішнім ідентифікатором: дублікати пропускають, а нові завдання додають і зберігають. Між зверненнями витримується випадкова затримка, після чого цикл повторюється, доки не буде опрацьовано всі тести й предмети. На завершення модуль виводить підсумкову статистику – кількість збережених, пропущених і помилкових сторінок.

Отже, описана модель збору завдань забезпечує автоматизоване наповнення банку питань із зовнішнього джерела, поєднуючи відтворення динамічних сторінок, структурований розбір розмітки, виявлення повторів та стійкість до тимчасових збоїв. Програмну реалізацію цього модуля докладно розглянуто в підрозділі 4.3.

3.6 Модель оцінювання ІІІ

Відповіді на закриті питання система перевіряє автоматично, звіряючи обраний варіант з еталоном. Проте для відкритих питань такий підхід незастосовний: вони передбачають розгорнуту відповідь власними словами, правильність якої не зводиться до точного збігу рядків і потребує оцінювання змісту й розуміння. Для цього передбачено окрему підсистему оцінювання на основі великої мовної моделі (Anthropic Claude), яка аналізує відкриту відповідь здобувача та виставляє кількісну оцінку розуміння разом зі змістовим коментарем.

Щоб бізнес-логіка не залежала від конкретного постачальника мовної моделі, доступ до ІІІ приховано за інтерфейсом IAiService. Споживачі – сервіс тестування та сервіс рекомендацій – звертаються лише до цього інтерфейсу, не знаючи про деталі реалізації. Інтерфейс мають дві реалізації: основна звертається до зовнішнього API, а резервна виконує спрощене локальне оцінювання, звіряючи ключові слова й числа відповіді з еталонною, і використовується, коли ключ доступу не налаштовано або потрібна робота без мережі. Конкретну реалізацію обирають під час

налаштування залежностей за наявністю ключа. Структуру підсистеми оцінювання наведено на рисунку 3.10.

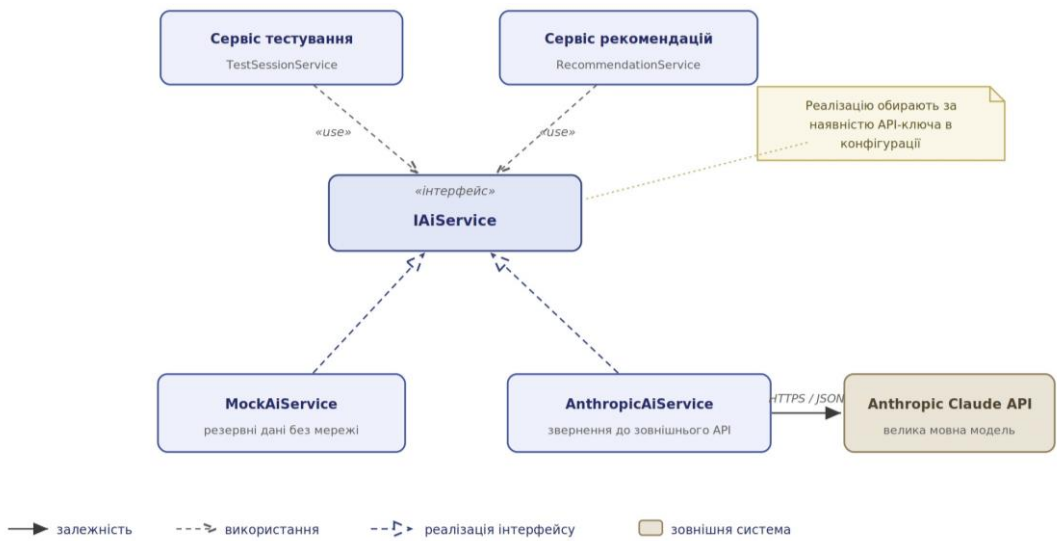


Рисунок 3.10 – Структура підсистеми оцінювання ІІІ

Як видно з рисунка, інтерфейс IAIService реалізовано двома класами: AnthropicAiService для звернення до моделі Claude та MockAiService для роботи без мережі.

Інтерфейс IAIService об'єднує три операції, які застосунок делегує мовній моделі. Їх призначення, вхідні дані та результат узагальнено в таблиці 3.5.

Таблиця 3.5 – Операції підсистеми оцінювання ІІІ

Операція	Вхідні дані	Результат
Оцінювання відкритої відповіді	Текст питання, еталонна відповідь, пояснення, відповідь здобувача	Оцінка розуміння (0–100) з переліком засвоєного, пропущеного та рекомендацією

Продовження таблиці 3.5

Операція	Вхідні дані	Результат
Формування рекомендацій	Перелік слабких тем здобувача	Оцінка готовності, мотиваційне повідомлення та впорядкований план тем
Резюме результату тесту	Підсумок сесії (бал, час)	Коротке мотиваційне резюме результату

Центральною є операція оцінювання відкритої відповіді, що відбувається безпосередньо під час проходження тесту. Перебіг взаємодії компонентів під час її виконання подано у вигляді діаграми послідовності на рисунку 3.11.

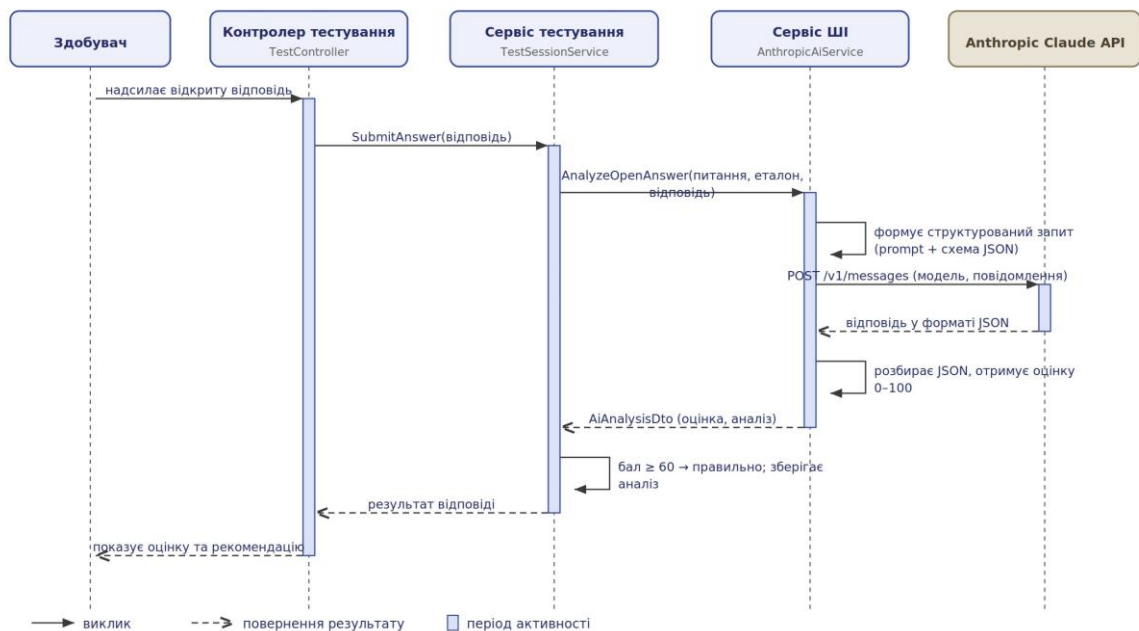


Рисунок 3.11 – Діаграма послідовності оцінювання відкритої відповіді

Коли здобувач надсилає відкриту відповідь, контролер передає її сервісу тестування, а той звертається до сервісу ШІ, передаючи текст питання, еталонну відповідь і відповідь здобувача. Сервіс ШІ формує структурований запит, у якому, окрім самих даних, задає роль моделі та чітку схему очікуваного результату у форматі JSON. Запит надсилається до зовнішнього API мовної моделі. Одержану відповідь розбирають і перетворюють на об'єкт з оцінкою розуміння за шкалою від 0 до 100 та текстовими поясненнями. Сервіс тестування інтерпретує цю оцінку: відповідь вважається правильною, якщо її текст точно збігається з еталонною відповіддю або якщо оцінка розуміння становить щонайменше 60 балів. Перевірку точного збігу виконують першою, тому коротка однослівна відповідь зараховується навіть тоді, коли модель недоступна чи дала занижену оцінку. Результат оцінювання разом із коментарем зберігається й показується здобувачеві. Якщо ключ доступу не налаштовано, оцінювання виконує резервна реалізація локально, без звернення до мережі, якщо ж звернення до API завершується помилкою, замість оцінки повертається порожній результат, і застосунок продовжує роботу без переривання.

Отже, описана модель оцінювання ШІ дає змогу перевіряти відкриті відповіді за змістом, а не за точним збігом, відокремлюючи бізнес-логіку від конкретного постачальника мовної моделі та забезпечуючи стійкість до відсутності зовнішнього сервісу. Програмну реалізацію інтеграції з API мовної моделі докладно розглянуто в підрозділі 4.4.

3.7 Модель аналітики та планування

Окрім проходження тестів, застосунок має допомагати здобувачеві відстежувати власний прогрес і впорядковувати підготовку. Для цього передбачено дві взаємопов'язані складові: підсистему аналітики, що узагальнює результати тестувань у наочні показники, та підсистему

планування, що формує календар занять. Аналітика виявляє слабкі теми здобувача, які стають підставою для персональних рекомендацій, а ті, своєю чергою, перетворюються на заплановані завдання.

Аналітику будує сервіс аналітики, який зчитує завершені тестові сесії та обчислює зведені показники прогресу. Окремо він визначає слабкі теми – теми з достатньою кількістю спроб і високою часткою помилок, – які передаються сервісу рекомендацій. Той звертається до сервісу ШІ для впорядкування тем і формування плану, кешує результат на певний час і додає рекомендовані завдання до календаря. Періодичне оновлення рекомендацій для всіх користувачів виконує фонові служба. Взаємодію цих складових наведено на рисунку 3.12.

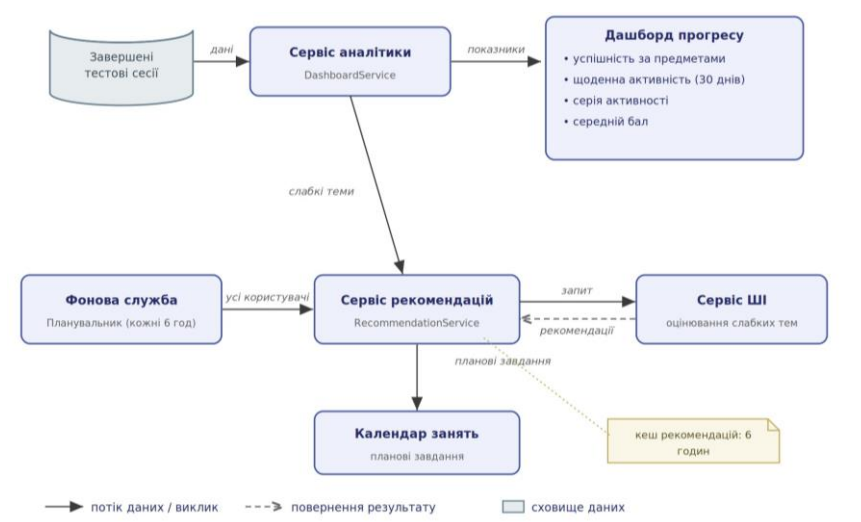


Рисунок 3.12 – Схема формування аналітики та рекомендацій

Основні показники, які обчислює сервіс аналітики, та спосіб їх отримання узагальнено в таблиці А.6.

Підсистема планування веде календар занять, у якому завдання бувають двох типів – опрацювання статті та проходження тесту – і мають статус запланованого або виконаного. Завдання можна додавати, позначати виконаними та вилучати вручну, проте основну цінність становить

автоматичне формування розкладу. Воно виконується за детермінованим алгоритмом і не звертається до мовної моделі: теми беруться з наперед укладеного каталогу, по одній на кожен предмет почергово, що забезпечує рівномірне чергування предметів. Послідовність кроків автоматичного планування подано на рисунку 3.13.

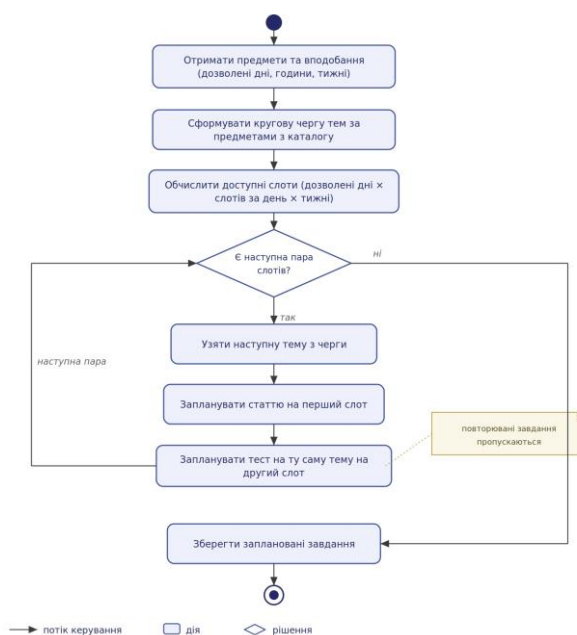


Рисунок 3.13 – Діаграма діяльності автоматичного формування розкладу

Спочатку за вподобаннями здобувача (дозволені дні тижня, кількість годин на день і тривалість у тижнях) визначають набір доступних навчальних слотів, а з каталогу тем формують кругову чергу, у якій предмети чергуються. Далі слоти проходять парами: у перший слот пари планують статтю з певної теми, а в наступний – тест із тієї самої теми, завдяки чому теоретичне опрацювання завжди передує перевірці. Уже наявні завдання повторно не додаються, що дає змогу безпечно повторювати планування. Після заповнення всіх пар слотів сформовані завдання зберігаються в календарі. Окремо до календаря потрапляють і завдання-повторення, що їх формує сервіс рекомендацій на основі виявлених слабких тем.

4 ПРОЄКТУВАННЯ ТА РОЗРОБКА КІНЦЕВОГО ЗАСТОСУНКУ

4.1 Реалізація рівня даних

У цьому розділі розглянуто програмну реалізацію рішень, спроєктованих у розділі 3. Рівень доступу до даних є найнижчим у багаторівневій структурі системи й розміщений у спільній бібліотеці NMT.Core (каталог Data), завдяки чому ним однаково користуються і вебзастосунок, і утиліта збору завдань. Рівень утворюють три групи складників: моделі сутностей предметної галузі, контекст бази даних та репозиторії. Реалізовано їх засобами об'єктно-реляційного перетворювача Entity Framework Core [9] поверх вбудованої бази даних SQLite [15].

Кожній із п'яти сутностей, обґрунтованих у підрозділі 3.2, відповідає окремий клас у каталозі Data/Models. Властивості класів позначено атрибутами-анотаціями ([Key], [Required], [MaxLength]), за якими перетворювач формує обмеження відповідних стовпців таблиці. Дані змінної структури – варіанти відповідей і результати оцінювання – зберігають як текст у форматі JSON, а перелічувані значення типу та стану завдання – у вигляді цілих чисел. Як приклад у лістингу Б.1 (додаток Б) наведено сутність Answer, що пов'язує сесію тестування з конкретним завданням і зберігає результат оцінювання відкритої відповіді.

Ключ Id генерується базою даних автоматично, а цілочислові поля TestSessionId і QuestionId слугують зовнішніми ключами. Поле SelectedOption заповнюють для закритих питань, тоді як OpenText – для відкритих, а серіалізований результат оцінювання мовною моделлю зберігають у текстовому полі AiAnalysisJson. Навігаційні властивості з атрибутом [ForeignKey] дають перетворювачеві змогу побудувати зв'язки між таблицями. Решту сутностей реалізовано за тим самим принципом: клас ApplicationUser розширює базовий клас підсистеми автентифікації, додаючи профільні поля та перелік обраних предметів, а клас CalendarTask

використовує перелічувані типи `TaskType` і `NmtTaskStatus` для позначення виду й стану запланованого заняття.

Центральним складником рівня є контекст бази даних – клас `NmtDbContext`, що успадковує `IdentityDbContext<ApplicationUser>`, через що прикладні таблиці й таблиці підсистеми автентифікації [16] зберігаються в одній базі. Колекції сутностей подано властивостями типу `DbSet`, а зв'язки, індекси та поведінку під час видалення налаштовано в перевизначеному методі `OnModelCreating`, наведеному в лістингу Б.2 (додаток Б).

Налаштування відтворює модель даних із підрозділу 3.2. Унікальний індекс за зовнішнім ідентифікатором завдання унеможливорює повторне збереження того самого питання, а складені індекси за предметом і роком, за користувачем і предметом та за користувачем і датою прискорюють типові вибірки. Зв'язки користувача із сесіями й завданнями календаря, а також сесії з відповідями налаштовано на каскадне видалення, тоді як для зв'язку відповіді із завданням встановлено обмежувальне видалення, яке захищає історію проходження від втрати цілісності. Виклик `ToTable(«Users»)` перейменовує таблицю облікових записів на коротшу назву.

Доступ до даних інкапсульовано в репозиторіях, кожен з яких реалізує власний інтерфейс і звертається до контексту бази даних, отриманого через впровадження залежностей. Такий поділ відповідає патерну «Репозиторій» [2] і дає змогу сервісам оперувати методами на кшталт отримання чи додавання сутностей, не залежачи від деталей формування запитів. Фрагмент репозиторію завдань `QuestionRepository` наведено в лістингу Б.3 (додаток Б).

Запити сформульовано мовою LINQ, яку перетворювач трансліує у SQL. Метод `GetRandomBySubjectAsync` добирає випадковий набір питань предмета й вилучає вже опрацьовані за переліком `excludeIds`, метод `GetByThemeAsync` повертає завдання однієї теми, що потрібно для формування пов'язаних занять у календарі, а метод `ExistsAsync` перевіряє

наявність завдання за зовнішнім ідентифікатором і застосовується модулем збору для уникнення повторів. Контекст бази даних репозиторій одержує параметром конструктора.

Складники рівня даних зв'язують у єдиний рівень засобами контейнера залежностей [3]. Метод-розширення `AddNmtCore` реєструє контекст бази даних із вказаним рядком з'єднання та постачальником `SQLite`, а також репозиторії й сервіси з областю життя в межах запиту. Цей метод викликають обидва проєкти, що використовують спільну базу, тому конфігурацію рівня даних задано в одному місці. Реєстрацію наведено в лістингу 4.1.

Лістинг 4.1 – Реєстрація рівня даних у контейнері залежностей

```
public static IServiceCollection AddNmtCore(
    this IServiceCollection services, string connectionString)
{
    services.AddDbContext<NmtDbContext>(opt =>
        opt.UseSqlite(connectionString,
            b => b.MigrationsAssembly("NMT.Core")));

    services.AddScoped<IQuestionRepository,
        QuestionRepository>();
    services.AddScoped<ITestSessionRepository,
        TestSessionRepository>();
    services.AddScoped<IUserRepository, UserRepository>();

    services.AddScoped<IQuestionService, QuestionService>();
    services.AddScoped<ITestSessionService,
        TestSessionService>();
    services.AddScoped<IDashboardService, DashboardService>();
    services.AddScoped<ICalendarService, CalendarService>();

    return services;
}
```

Оскільки репозиторії й сервіси зареєстровано через інтерфейси, споживачі залежать від абстракцій, а конкретні реалізації надає контейнер. Фізичну схему бази даних за описаною моделлю формує сам перетворювач: під час першого запуску застосунку виклик `EnsureCreatedAsync` створює всі таблиці відповідно до моделі, а допоміжний метод перевіряє наявність окремих стовпців, доданих до моделі згодом, і за потреби додає їх без перестворення бази. Організацію складників рівня даних у проєкті, наведено на рисунку 4.1.

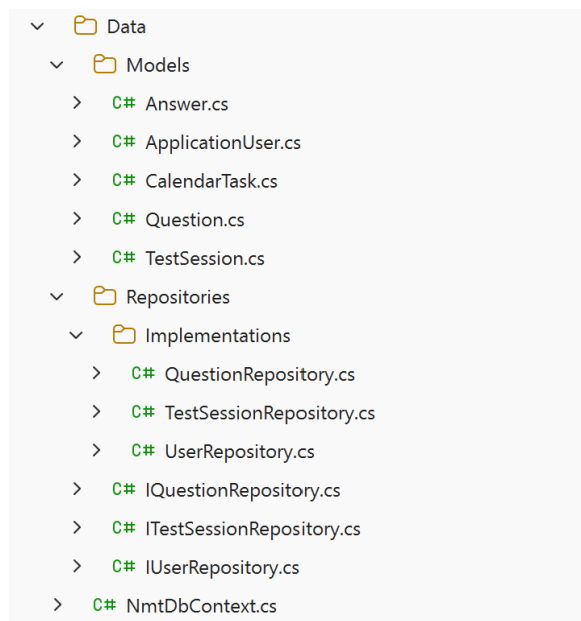


Рисунок 4.1 – Організація рівня доступу до даних у проєкті NMT.Core

Отже, у підрозділі реалізовано рівень доступу до даних: моделі сутностей із анотаціями, контекст бази даних з індексами та налаштованою поведінкою видалення, репозиторії з типовими запитам та реєстрацію складників у контейнері залежностей. Сформований рівень забезпечує всі операції зі зберігання й вибирання даних, на які спираються бізнес-логіка та інтерфейс користувача, розглянуті в наступних підрозділах.

4.2 Реалізація бізнес-логіки

Рівень бізнес-логіки реалізує прикладні правила системи й розміщений у спільній бібліотеці NMT.Core (каталог BLL). Його утворюють чотири сервіси, обґрунтовані в підрозділі 3.3: QuestionService надає тестові завдання, TestSessionService проводить сесії тестування й перевіряє відповіді, DashboardService обчислює показники прогресу, а CalendarService веде календар і формує розклад занять. Кожен сервіс реалізує відповідний інтерфейс і одержує потрібні репозиторії та контекст бази даних через впровадження залежностей [3], тому звертається до нижчого рівня лише через абстракції.

Сервіс завдань QuestionService перетворює сутності банку питань на об'єкти передавання даних, придатні для подання. Під час формування питання він розбирає збережений у форматі JSON перелік варіантів відповіді й визначає, чи має поточне питання потребувати розгорнутої відповіді: відкритою стає кожне третє питання за умовою $\text{questionNumber} \% 3 == 0$. Добір питань для тесту виконується або за конкретним тестом, або у вигляді випадкового набору з предмета, якщо тест не задано.

Центральним складником рівня є сервіс сесій TestSessionService, що відкриває сесію, приймає та перевіряє відповіді й обчислює підсумковий результат. Найважливіша його операція – перевірка відповіді, реалізацію якої наведено в лістингу Б.4 (додаток Б). Вона по-різному опрацьовує закриті та відкриті питання відповідно до моделі оцінювання з підрозділу 3.6.

Відповідь на закриті питання визнають правильною за збігом обраного варіанта з еталонним без урахування реєстру. Для відкритої відповіді спершу виконують швидку перевірку точного збігу з еталонним текстом, що зараховує коротку однослівну відповідь навіть тоді, коли мовна модель недоступна. Якщо точного збігу немає, відповідь передають сервісу

III, і вона вважається правильною за оцінки розуміння щонайменше 60 балів. Опрацьовану відповідь разом із результатом оцінювання зберігають, а підсумок повертають для відображення. Деталі звернення до зовнішнього API мовної моделі розглянуто в підрозділі 4.4. Завершення сесії обчислює кількість правильних відповідей і тривалість безпосередньо за збереженими записами, тому результат лишається коректним незалежно від стану кешу.

Сервіс аналітики `DashboardService` узагальнює завершені сесії у показники прогресу, що відповідає моделі аналітики з підрозділу 3.7. Він обчислює відсоток правильних відповідей за кожним предметом, активність за останні тридцять днів і безперервну серію днів занять. Окремо сервіс визначає слабкі теми здобувача – теми з достатньою кількістю спроб і високою часткою помилок, які стають підставою для рекомендацій. Алгоритм їх виявлення наведено в лістингу 4.2.

Лістинг 4.2 – Виявлення слабких тем

```
var themeGroups = answers
    .GroupBy(a => a.Question!.Theme)
    .Where(g => g.Count() >= 2)
    .Select(g => new
    {
        Theme = g.Key,
        Total = g.Count(),
        Correct = g.Count(a => a.IsCorrect),
        ErrorRate = 1.0 - (double)g.Count(a => a.IsCorrect) /
g.Count()
    })
    .Where(t => t.ErrorRate >= 0.4)
    .OrderByDescending(t => t.ErrorRate)
    .Take(5);
```

Відповіді групують за темою завдання, після чого відкидають теми з менш ніж двома спробами, щоб уникнути випадкових висновків за

поодинокими відповідями. Для кожної теми обчислюють частку помилок, залишають лише теми з часткою від сорока відсотків і впорядковують їх за спаданням цього показника. Дібрані теми разом із показниками прогресу формують зведення дашборду, наведене на рисунку 4.2.

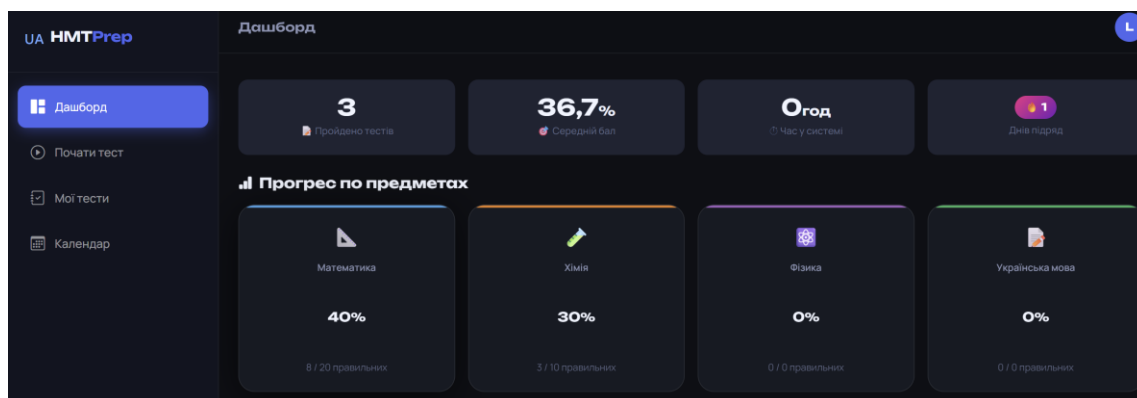


Рисунок 4.2 – Дашборд прогресу здобувача

Сервіс календаря CalendarService реалізує підсистему планування з підрозділу 3.7. Він підтримує ручне керування завданнями – додавання, позначення виконаними та видалення, – проте основну цінність становить автоматичне формування розкладу. Алгоритм укладає заняття парами «стаття – тест», у яких теоретичне опрацювання теми передуює її перевірці, а самі теми беруться з наперед визначеного каталогу по черзі для рівномірного чергування предметів. Ключову частину алгоритму наведено в лістингу 4.3.

Лістинг 4.3 – Формування пар занять «стаття – тест»

```
var topicQueue = new List<(string subject, string theme, string
title, string url)>();
int maxlen = subjects.Select(s =>
topicCatalog.GetValueOrDefault(s)?.Length ?? 0)
.DefaultIfEmpty(0).Max();
for (int round = 0; round < maxlen; round++)
foreach (var s in subjects)
```

Продовження лістингу 4.3

```

{
    var topics = topicCatalog.GetValueOrDefault(s);
    if (topics != null && round < topics.Length)
        topicQueue.Add((s, topics[round].theme,
            topics[round].title, topics[round].url));
}
for (int slotIdx = 0; slotIdx + 1 < slots.Count; slotIdx += 2)
{
    var unit = topicQueue[topicIdx++ % topicQueue.Count];
    var articleDate = slots[slotIdx];
    var testDate = slots[slotIdx + 1];
}

```

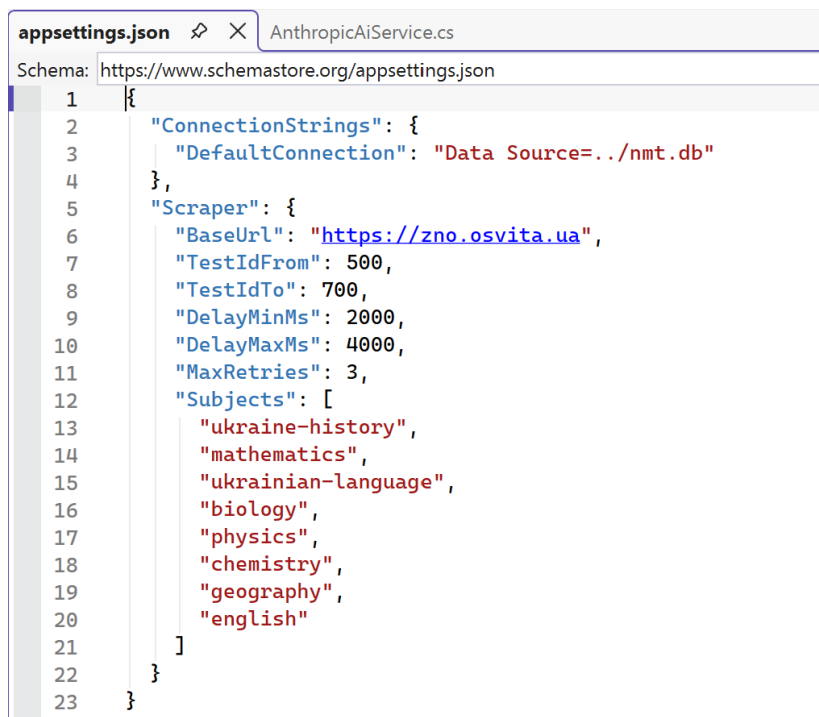
Спочатку за вподобаннями здобувача формують перелік доступних навчальних слотів, а з каталогу тем – кругову чергу, у якій предмети чергуються. Слоти проходять парами: у перший слот пари планують статтю з певної теми, а в наступний – тест із тієї самої теми, через що теоретичне опрацювання завжди передуює перевірці. Перед додаванням завдання перевіряють на наявність, тому повторний запуск планування не створює дублікатів. Теми каталогу збігаються зі значеннями теми у банку завдань, що гарантує добір питань для запланованого тесту.

Отже, у підрозділі реалізовано рівень бізнес-логіки: сервіс завдань готує питання до подання, сервіс сесій проводить тестування з роздільним опрацюванням закритих і відкритих відповідей, сервіс аналітики обчислює прогрес і виявляє слабкі теми, а сервіс календаря формує парний розклад занять. Завдяки розмежуванню відповідальності між сервісами кожен із них інкапсулює власну предметну логіку, що спрощує супровід застосунку та полегшує подальше розширення функціоналу. Сервіси спираються на рівень доступу до даних і надають готові об'єкти передавання для рівня представлення, розглянутого в підрозділі 4.5.

4.3 Реалізація модуля скрапінгу

Модуль збору завдань реалізовано як окрему консольну програму NMT.Scraper, що відтворює конвеєр опрацювання даних, спроектований у підрозділі 3.5. Програма посилається на спільну бібліотеку NMT.Core, тому використовує ту саму модель даних і той самий рівень доступу до даних, що й вебзастосунок, і зібрані завдання одразу стають придатними для тестування. Складники модуля зв'язує контейнер залежностей узагального вузла (Generic Host): параметри збору зчитуються з файлу конфігурації у клас ScraperConfig, реєструється рівень доступу до даних і сам скрапер, після чого виконується створення схеми бази даних та запуск збору.

Параметри збору винесено в конфігурацію, тому поведінку модуля змінюють без перекомпіляції. Клас ScraperConfig зберігає адресу джерела, діапазон ідентифікаторів тестів, перелік предметів, межі випадкової затримки між запитами та кількість повторів, відповідну конфігурацію наведено на рисунку 4.3.



```
1  {
2    "ConnectionStrings": {
3      "DefaultConnection": "Data Source=../nmt.db"
4    },
5    "Scraper": {
6      "BaseUrl": "https://zno.osvita.ua",
7      "TestIdFrom": 500,
8      "TestIdTo": 700,
9      "DelayMinMs": 2000,
10     "DelayMaxMs": 4000,
11     "MaxRetries": 3,
12     "Subjects": [
13       "ukraine-history",
14       "mathematics",
15       "ukrainian-language",
16       "biology",
17       "physics",
18       "chemistry",
19       "geography",
20       "english"
21     ]
22   }
23 }
```

Рисунок 4.3 – Конфігурація модуля збору завдань

Стійкість збору до тимчасових збоїв забезпечує політика повторів на основі бібліотеки Polly [6], яку налаштовують у конструкторі скрапера (лістинг 4.4).

Лістинг 4.4 – Налаштування політики повторів

```
_retryPolicy = Policy
    .Handle<Exception>()
    .WaitAndRetryAsync(_config.MaxRetries,
        attempt => TimeSpan.FromSeconds(Math.Pow(2, attempt)),
        (ex, ts, attempt, _) =>
            _logger.LogWarning("Retry {Attempt} after
{Delay}s: {Msg}",
                attempt, ts.TotalSeconds, ex.Message));
```

Політика перехоплює виняток будь-якого типу й повторює операцію задану кількість разів із наростаючою затримкою, що подвоюється на кожній спробі. Це згладжує тимчасову недоступність джерела, не перериваючи збір повністю.

Збором керує метод `RunAsync`, що утворює зовнішній цикл конвеєра. Перед початком роботи він установлює й запускає фоновий браузер Chromium у безголовому режимі, після чого по чергову перебирає предмети, діапазони та ідентифікатори тестів. Для кожного тесту формується адреса сторінки, яку опрацьовують із застосуванням політики повторів, одержані завдання перевіряють на повторюваність і зберігають. Ключову частину циклу наведено в лістингу Б.5 (додаток Б).

Кожне завдання перевіряють за зовнішнім ідентифікатором через метод репозиторію `ExistsAsync`: наявні в базі завдання пропускають і обліковують окремо, а нові додають і зберігають. Між зверненнями до сайту витримується випадкова пауза в заданих межах. Цикл реагує на скасування, тому збір можна коректно зупинити сполученням клавіш, а після

завершення модуль виводить підсумкову статистику – кількість збережених, пропущених і помилкових сторінок.

Завантаження сторінки покладено на метод `ScrapeTestPageAsync`, що використовує бібліотеку `Playwright` [4]. Браузер відкриває сторінку й повністю відтворює її вміст, зокрема елементи, сформовані сценаріями на боці клієнта, після чого метод перевіряє доступність сторінки й передає готову розмітку парсеру. Реалізацію наведено в лістингу 4.5.

Лістинг 4.5 – Відтворення сторінки фоновим браузером

```
var page = await browser.NewPageAsync();
try
{
    await page.GotoAsync(url, new() { Timeout = 30_000,
                                     WaitUntil =
WaitUntilState.NetworkIdle });
    var title = await page.TitleAsync();
    if (title.Contains("404") || title.Contains("не
знайдено")) return null;
    try { await page.WaitForSelectorAsync(".question", new() {
Timeout = 10_000 }); }
    catch { return null; }
    var html = await page.ContentAsync();
    return await ParseQuestionsAsync(html, subject, testId,
year, session);
}
finally { await page.CloseAsync(); }
```

Метод очікує завершення мережевої активності, що гарантує завантаження динамічного вмісту. Сторінки з ознакою відсутності (заголовок із кодом 404) або без блоків завдань пропускають, повертаючи порожній результат. З коректної сторінки одержують відтворену розмітку й передають її на розбір.

Розбір розмітки виконує бібліотека AngleSharp [5], яка будує модель документа й дає змогу видобувати елементи за CSS-селекторами. Метод ParseQuestionsAsync відбирає вузли завдань, а кожен із них перетворює на сутність метод ParseSingleQuestion, наведений у лістингу Б.6 (додаток Б).

З вузла завдання видобувають ідентифікатор, з якого разом із предметом і номером тесту складають унікальний зовнішній ідентифікатор для виявлення повторів. Далі добувають текст питання, перелік варіантів відповіді (не більше чотирьох, із позначками А-Г), правильну відповідь, а також тему й пояснення, якщо вони наявні. Завдання без тексту або без варіантів відкидають. Перелік варіантів зберігають у форматі JSON, а позначку правильної відповіді приводять до єдиного вигляду методом NormalizeAnswer, що зводить латинські літери та цифри до української абетки. Усі зібрані завдання належать до закритого типу, тому позначаються як `multiple_choice`.

Отже, у підрозділі реалізовано модуль автоматизованого збору завдань: параметри винесено в конфігурацію, відтворення динамічних сторінок виконує безголовий браузер, структурований розбір розмітки – бібліотека AngleSharp, виявлення повторів – рівень доступу до даних, а стійкість до тимчасових збоїв – політика повторів із наростальною затримкою. Зібрані завдання поповнюють спільний банк питань, на якому ґрунтується робота вебзастосунку.

4.4 Інтеграція API штучного інтелекту

Підсистему оцінювання відкритих відповідей реалізовано згідно з моделлю, обґрунтованою в підрозділі 3.6. Доступ до мовної моделі приховано за інтерфейсом IAIService, який має дві реалізації, розміщені у вебзастосунку (каталог Services/AI), оскільки вони потребують доступу до зовнішніх служб і конфігурації. Основна реалізація AnthropicAiService звертається до зовнішнього API Anthropic Claude [7], а резервна

MockAiService виконує спрощене локальне оцінювання без мережі. Конкретну реалізацію обирають під час налаштування залежностей за наявністю ключа доступу, як показано в лістингу 4.6.

Лістинг 4.6 – Вибір реалізації сервісу ІІІ за наявністю ключа

```
var apiKey = builder.Configuration["AI:AnthropicApiKey"] ??
"";
if (!string.IsNullOrEmpty(apiKey) && apiKey !=
"YOUR_ANTHROPIC_API_KEY_HERE")
    builder.Services.AddScoped<IAiService,
    AnthropicAiService>();
else
    builder.Services.AddScoped<IAiService, MockAiService>();
```

Якщо ключ налаштовано, контейнер залежностей реєструє реалізацію, що працює з реальним API, інакше підставляється резервна реалізація. Споживачі сервісу – сервіс тестування та сервіс рекомендацій – звертаються лише до інтерфейсу й не залежать від обраної реалізації. Сам ключ зчитується з конфігурації, де його зберігають поза системою контролю версій у механізмі User Secrets платформи .NET.

Центральною операцією підсистеми є оцінювання відкритої відповіді. Метод AnalyzeOpenAnswerAsync формує текстовий запит, у якому задає роль моделі (досвідчений учитель), передає текст питання, еталонну відповідь і відповідь здобувача та чітко описує очікувану структуру результату у форматі JSON. Одержану відповідь розбирають на об'єкт з оцінкою розуміння за шкалою від 0 до 100 і текстовими поясненнями. Реалізацію наведено в лістингу Б.7 (додаток Б).

Оскільки модель іноді повертає JSON, обрамлений символами розмітки, перед розбором результат очищають від таких символів і службового префікса. У разі помилки розбору метод повертає порожній

результат, тому збій оцінювання не перериває проходження тесту. Подання результату оцінювання здобувачеві наведено на рисунку 4.4.

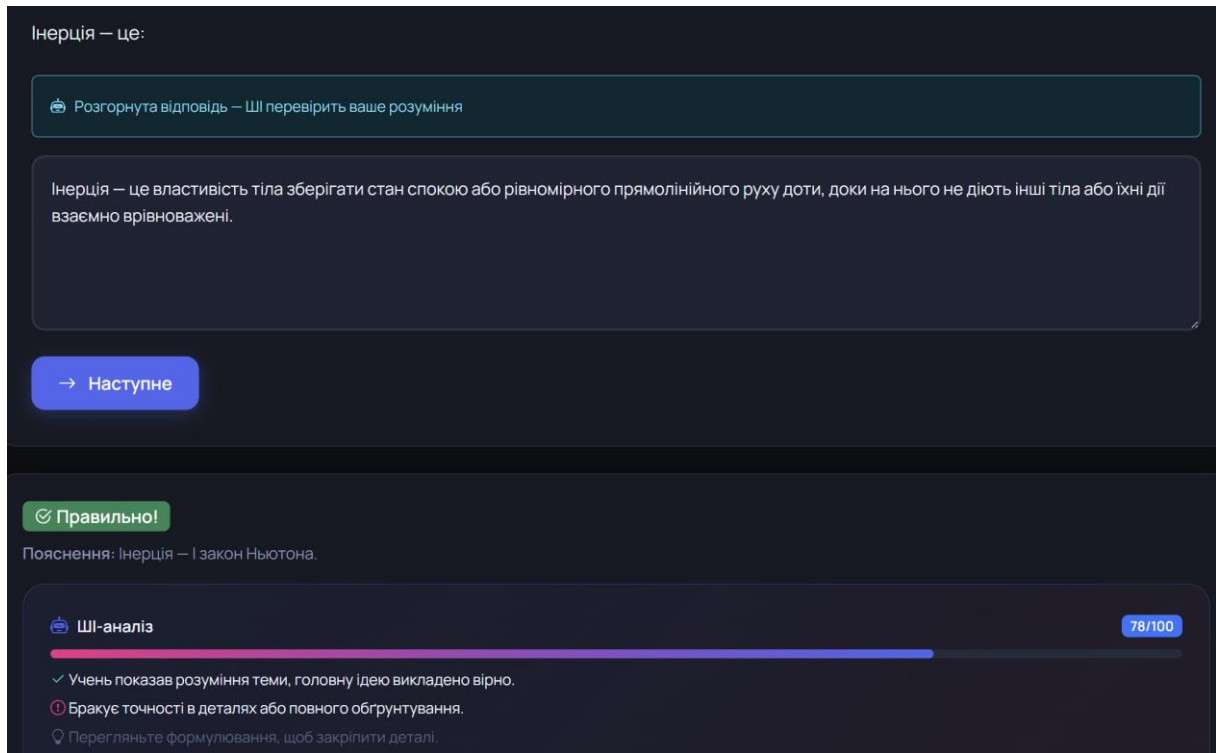


Рисунок 4.4 – Результат оцінювання відкритої відповіді засобами ШІ

Звернення до зовнішнього API зосереджено в допоміжному методі `CallClaudeAsync`, який використовують усі операції підсистеми. Метод формує HTTP-запит до інтерфейсу повідомлень Claude, додає потрібні заголовки автентифікації та версії, серіалізує тіло запиту з назвою моделі, обмеженням довжини відповіді та переліком повідомлень, після чого видобуває текст із відповіді. Реалізацію наведено в лістингу Б.8 (додаток Б).

За замовчуванням використовують модель `claude-3-5-haiku`, що забезпечує прийнятний баланс швидкості та якості оцінювання. Об'єкт `HttpClient` сервіс одержує через фабрику клієнтів, що повторно використовує з'єднання. На тому самому методі ґрунтуються ще дві операції: формування персональних рекомендацій за переліком слабких тем (`GenerateRecommendationsAsync`) і коротке підсумкове резюме

результату тесту (`GenerateTestSummaryAsync`). Обидві так само задають у запиті схему очікуваного результату й розбирають відповідь моделі.

Резервна реалізація `MockAiService` забезпечує роботу застосунку за відсутності ключа або потреби працювати без мережі. Замість звернення до API вона оцінює відповідь локально, звіряючи числа та ключові слова відповіді здобувача з еталонною. Алгоритм обчислення оцінки наведено в лістингу Б.9.

Оцінку обчислюють як зважену суму частки збігу ключових слів і частки збігу чисел еталонної відповіді, причому для завдань із числовою відповіддю більшу вагу має збіг чисел. За одержаним балом резервна реалізація обирає й текстові коментарі. Так застосунок зберігає працездатність незалежно від доступності зовнішнього сервісу.

Отже, у підрозділі реалізовано інтеграцію з API мовної моделі: доступ до ШІ відокремлено інтерфейсом із двома реалізаціями, основна з яких формує структуровані запити до зовнішнього сервісу та розбирає його відповіді, а резервна виконує оцінювання локально. Такий підхід дає змогу оцінювати відкриті відповіді за змістом, не прив'язуючи бізнес-логіку до конкретного постачальника й зберігаючи стійкість до відсутності мережі.

4.5 Реалізація вебінтерфейсу

Рівень представлення реалізовано у вебзастосунку `NMT.Web` за шаблоном «Модель-Подання-Контролер» [1] і відповідає архітектурі з підрозділу 3.1. Його утворюють контролери, що приймають запити користувача, Razor-подання, що формують розмітку сторінок, моделі подань для передавання даних у подання та спільний шаблон сторінки. Контролери залишаються тонкими: вони перевіряють права доступу, звертаються до сервісів бізнес-логіки через їхні інтерфейси й повертають подання або дані, не містячи прикладних правил. Зовнішній вигляд побудовано на основі фреймворку `Bootstrap 5` [21] із власною системою

стилів у файлі `nmt.css`, інтерактивність на боці клієнта забезпечує бібліотека `Alpine.js`, а календар занять – бібліотека `FullCalendar` [22].

Спільний шаблон `_Layout` задає структуру всіх сторінок: для автентифікованого користувача він виводить бічну панель навігації, а для гостя – спрощений вигляд. У шаблоні також розміщено маркер захисту від міжсайтової підробки запитів, який використовують асинхронні звернення до сервера. Оформлення підпорядковане єдиній темній темі з кольоровим кодуванням предметів.

Центральним сценарієм інтерфейсу є проходження тесту, формалізоване в підрозділі 3.4. Його провадить контролер `TestController`. Дія `Start` показує перелік обраних предметів із показниками, а після вибору предмета – форму налаштування тесту (тема, рік, кількість питань). Екран вибору предмета наведено на рисунку 4.5. Дія `Begin` за обраними параметрами формує набір питань, відкриває сесію тестування й перенаправляє до першого питання, а дія `Answer` приймає відповідь асинхронним запитом і повертає результат у форматі `JSON`. Ключові дії контролера наведено в лістингу Б.10 (додаток Б).

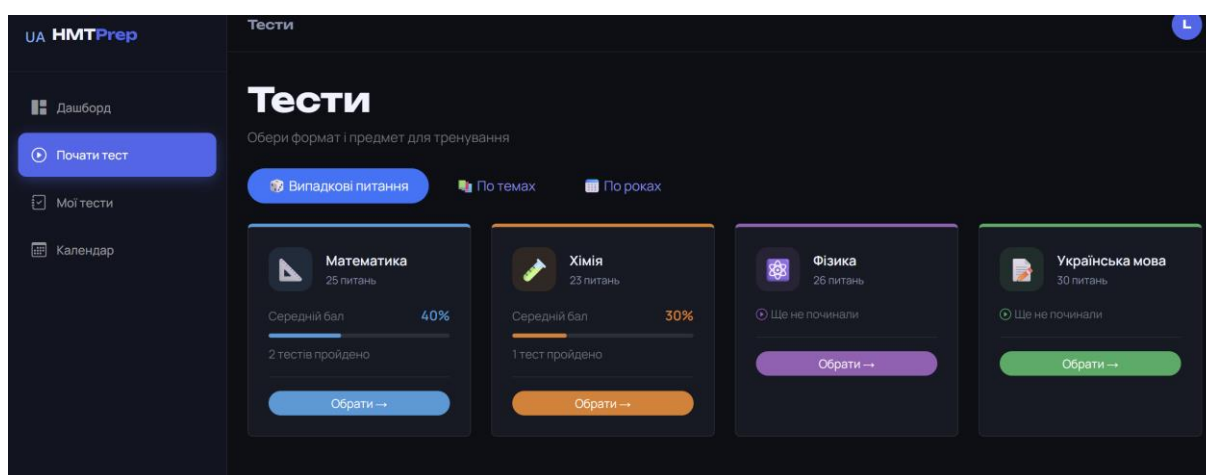


Рисунок 4.5 – Екран вибору предмета та налаштування тесту

Контролер позначено атрибутом, що вимагає автентифікації, а дії, які змінюють стан – перевіркою маркера захисту від підробки запитів. Перевірку відповіді делеговано сервісу сесій, тому дія Answer лише передає одержані дані й повертає результат.

Сторінку питання реалізовано як компонент Alpine.js, що дає змогу перевіряти відповідь без перезавантаження сторінки. Користувач обирає варіант або вводить розгорнуту відповідь, після чого метод submitAnswer надсилає дані до сервера асинхронним запитом і за одержаним результатом оновлює вигляд – показує правильність, пояснення та аналіз III. Реалізацію наведено в лістингу Б.11 (додаток Б), а вигляд сторінки під час проходження тесту – на рисунку 4.6.

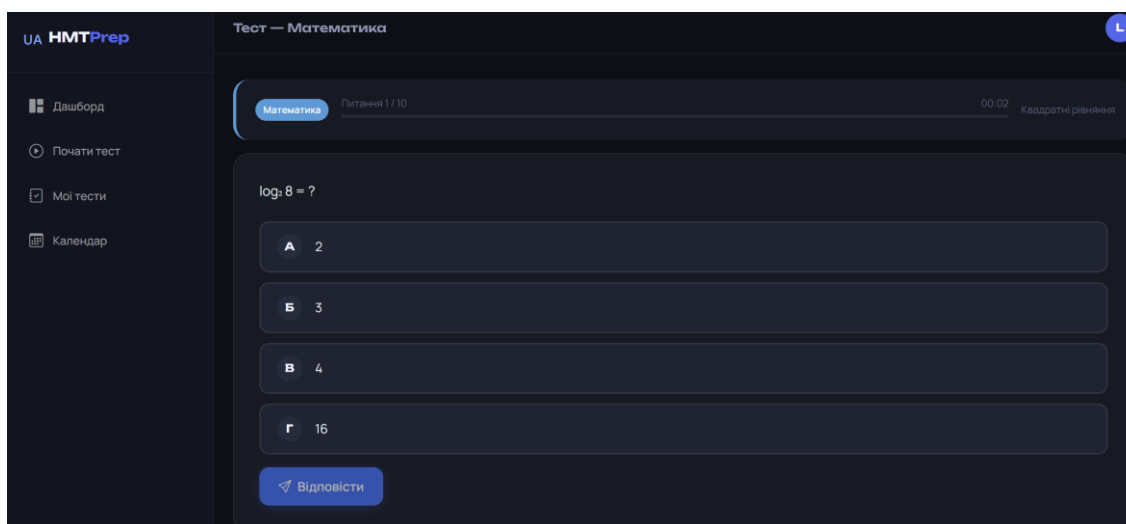


Рисунок 4.6 – Сторінка проходження тесту

Запит надсилається методом POST на дію Answer із маркером захисту в заголовку, а одержаний об'єкт результату оновлює властивості компонента, що автоматично відображається в розмітці. Завдяки цьому перехід між станами «питання – перевірено» відбувається миттєво, без повного перезавантаження сторінки.

Доступ до навчальних можливостей відкривається після реєстрації, яку провадить контролер AccountController на основі підсистеми ASP.NET

Core Identity [16]. Під час реєстрації користувач обирає рівно чотири предмети, які складатиме на тесті. Перелік предметів зберігається в обліковому записі. Реалізацію дії наведено в лістингу Б.12 (додаток Б), а форму реєстрації – на рисунку 4.7.

The image shows a registration form for the Core Identity system. At the top, there are two input fields for the first and last names: "Леся" and "Хамінова". Below these are fields for "Email" (containing "lesiakhaminova@gmail.com"), "Password" (with a hint "Мінімум 6 символів"), and "Confirm password". A section titled "Оберіть 4 предмети НМТ" (4/4) displays eight subject options in a grid: "Історія України", "Математика", "Українська мова", "Біологія", "Фізика", "Хімія", "Географія", and "св Англійська мова". The first four subjects are highlighted with blue borders, indicating they are selected. At the bottom, there is a large blue button labeled "Зареєструватися" (Register) and a link "Вже є акаунт? Увійти" (Already have an account? Log in).

Рисунок 4.7 – Форма реєстрації з вибором предметів

Перед створенням облікового запису перевіряють, що обрано саме чотири предмети, інакше форму повертають із повідомленням про помилку. Успішно створеному користувачеві призначають роль студента й

автоматично виконують вхід. Створення облікового запису, перевірку пароля та керування сеансом виконує підсистема Identity.

Планування занять подано окремою сторінкою з інтерактивним календарем на основі бібліотеки FullCalendar. Календар отримує заплановані завдання з окремої дії контролера у форматі, придатному для відображення, а дії над завданнями (створення, позначення виконаним, вилучення, автоматичне формування розкладу) виконуються асинхронними запитами з наступним оновленням подій. Ініціалізацію календаря наведено в лістингу Б.13 (додаток Б), а його вигляд – на рисунку 4.8.

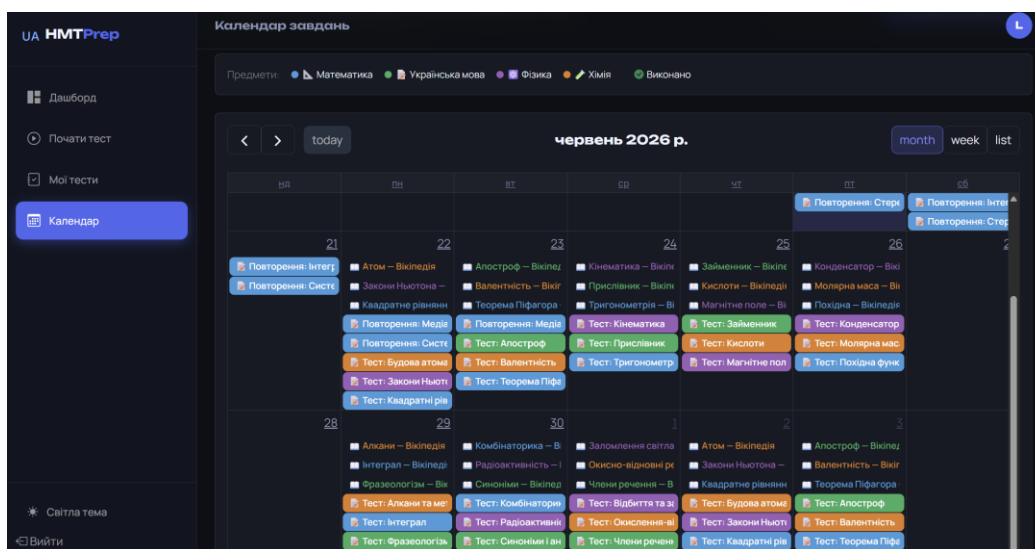


Рисунок 4.8 – Інтерфейс календаря підготовки

Календар налаштовано на українську локаль і місячне подання з перемиканням на тижневе та списком, а завдання він завантажує з адреси /Calendar/GetTasks. Виконані завдання виділяють окремим класом стилю, а натискання на завдання відкриває картку з його деталями. Після будь-якої зміни календар повторно запитує події, тому відображення завжди відповідає стану даних.

Отже, у підрозділі реалізовано рівень представлення: тонкі контролери з перевіркою прав і захистом від підробки запитів делегують

роботу сервісам, Razor-подання на основі Bootstrap формують сторінки, а бібліотеки Alpine.js і FullCalendar забезпечують інтерактивність без перезавантаження. Побудований інтерфейс реалізує сценарії, описані в підрозділі 3.4, і надає здобувачеві цілісний доступ до всіх можливостей системи.

4.6 Забезпечення безпеки

Безпеку застосунку забезпечено на всіх рівнях обробки запиту – від автентифікації користувача до захисту даних під час передавання та зберігання. Засоби безпеки реалізовано переважно вбудованими механізмами платформи ASP.NET Core, які налаштовують під час побудови застосунку.

Автентифікацію та керування обліковими записами реалізовано на основі підсистеми ASP.NET Core Identity [16]. Паролі ніколи не зберігають у відкритому вигляді – підсистема зберігає лише їхні криптографічні хеші. Політику паролів, сховище облікових записів і параметри автентифікаційного кук-файла налаштовують під час реєстрації служб, як показано в лістингу Б.14 (додаток Б).

Сеанс користувача підтримується автентифікаційним кук-файлом, доступним лише серверу (прапорець HttpOnly), переданим лише захищеним з'єднанням (прапорець Secure) та обмеженим за міжсайтовим використанням (атрибут SameSite), що унеможливорює його викрадення сценаріями на боці клієнта чи використання в підроблених запитах. Строк дії автентифікаційного квитка налаштовано на чотирнадцять днів зі скоєним продовженням за активності, а спроби доступу неавтентифікованого користувача перенаправляються на сторінку входу.

Розмежування доступу побудовано на ролях і політиці авторизації. Контролери, що надають доступ до особистих даних, позначено атрибутом обов'язкової автентифікації, а ролі створюють під час запуску застосунку.

Окрім перевірки автентифікації, застосовано перевірку належності ресурсу на рівні окремого об'єкта: перш ніж показати сесію тестування, контролер пересвідчується, що вона належить поточному користувачеві, інакше повертає відмову в доступі. Реалізацію наведено в лістингу 4.7.

Лістинг 4.7 – Перевірка належності ресурсу користувачеві

```
private async Task<TestSession?> GetSessionOrForbidAsync(int
sessionId, string userId)
{
    var session = await ctx.TestSessions.FindAsync(sessionId);
    return session is { } s && s.UserId == userId ? s : null;
}
var session = await GetSessionOrForbidAsync(sessionId,
userId);
if (session == null) return Forbid();
```

Завдяки такій перевірці користувач не може звернутися до чужої сесії, навіть знаючи її ідентифікатор, оскільки розбіжність власника спричиняє відмову в доступі.

Запити, що змінюють стан системи, захищено від міжсайтової підробки. Службу захисту налаштовують із назвою заголовка для асинхронних запитів, а дії, які приймають дані, позначають атрибутом перевірки маркера [1]. Реалізацію наведено в лістингу 4.8.

Лістинг 4.8 – Захист від міжсайтової підробки запитів

```
builder.Services.AddAntiforgery(o => o.HeaderName =
"RequestVerificationToken");
[HttpPost, ValidateAntiForgeryToken]
public async Task<IActionResult> Answer([FromBody]
SubmitAnswerDto dto) { /* ... */ }
```

Маркер захисту розміщено у спільному шаблоні сторінки, а асинхронні звернення передають його у визначеному заголовку, тому запит без дійсного маркера відхиляється. Звичайні форми передають маркер прихованим полем автоматично.

Безпеку передавання даних забезпечує перенаправлення всіх запитів на захищений протокол HTTPS, а в робочому середовищі додатково ввімкнено механізм суворого транспортного захисту HSTS [14], який зобов'язує браузер звертатися лише захищеним з'єднанням. Захист від впровадження коду досягається без додаткових зусиль: усі запити до бази даних формує об'єктно-реляційний перетворювач із параметризацією значень [9], що унеможливорює впровадження SQL-коду, а Razor-подання автоматично екранують виведені дані, запобігаючи міжсайтовому виконанню сценаріїв. Вхідні дані додатково перевіряють за анотаціями моделей подань. Ключ доступу до зовнішнього API зберігають поза кодом і системою контролю версій – у механізмі User Secrets платформи .NET, тому він не потрапляє до сховища вихідного коду.

Належність автентифікаційного кук-файла до захищених і недоступних сценаріям підтверджує його перегляд у засобах розробника браузера, наведений на рисунку 4.9.

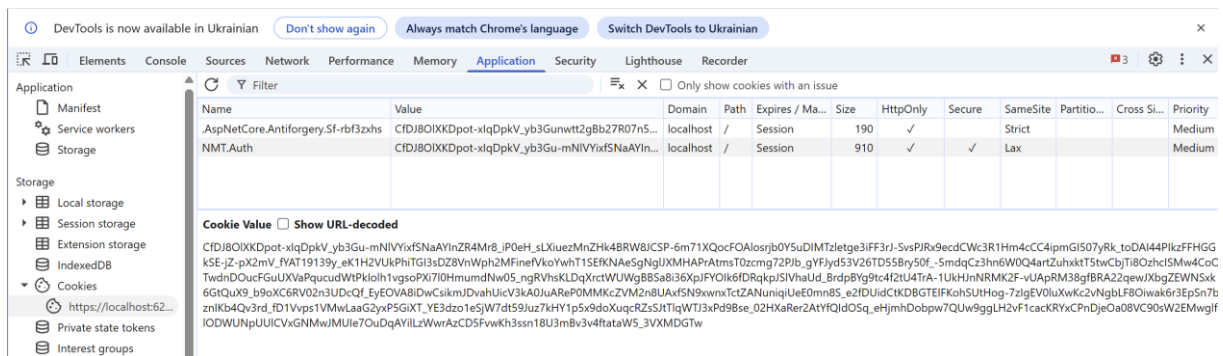


Рисунок 4.9 – Параметри автентифікаційного кук-файла

Отже, у підрозділі забезпечено безпеку застосунку: автентифікацію та хешування паролів виконує підсистема Identity, доступ розмежовано ролями й перевіркою належності ресурсів, запити захищено від міжсайтової підробки, передавання даних – протоколом HTTPS, а зберігання – параметризацією запитів і захищеним зберіганням ключів. Сукупність цих заходів захищає персональні дані здобувачів і цілісність роботи системи.

4.7 Тестування застосунку

Перевірку працездатності застосунку виконано методом функціонального тестування за принципом «чорної скриньки»: за основу взято сценарії використання, описані в підрозділі 3.4, а правильність роботи оцінювали за відповідністю фактичної поведінки очікуваній без аналізу внутрішнього коду. Такий підхід обрано з огляду на характер системи – інтерактивного вебзастосунку, цінність якого визначає узгоджена робота всіх рівнів у межах цілісного сценарію, а не окремих функцій ізольовано.

Тестування охопило основні сценарії: реєстрацію й автентифікацію, проходження тестів із закритими та відкритими питаннями, оцінювання відкритих відповідей засобами ІІІ, відображення показників прогресу та автоматичне планування занять. Для кожного сценарію визначено вхідні дії та очікуваний результат, після чого сценарій відтворювали в застосунку. Результати наведено в таблиці А.7 (додаток А).

Окремо перевірено межові випадки та коректність опрацювання неправильних вхідних даних. Спроба зареєструватися з кількістю предметів, відмінною від чотирьох, або з уже зайнятою адресою пошти відхиляється з відповідним повідомленням – «Порожню відповідь надіслати неможливо». Звернення до сесії, що належить іншому користувачеві, спричиняє відмову в доступі. Перевірено також стійкість системи до зовнішніх збоїв: за відсутності ключа доступу або недоступності мережі оцінювання відкритих відповідей виконує резервна реалізація, тому

проходження тесту не переривається. Здатність підсистеми оцінювання розрізняти повноту відповіді ілюструє рисунок 4.10, де неповній відповіді виставлено низьку оцінку з переліком недоліків.

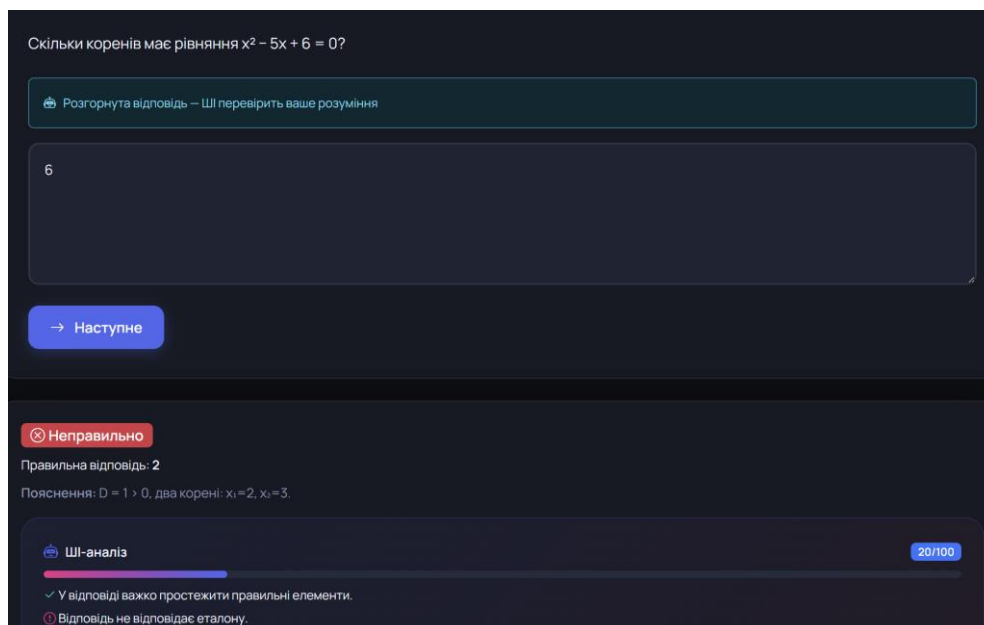


Рисунок 4.10 – Результат оцінювання неповної відповіді

Під час тестування виявлено й усунено два дефекти. По-перше, питання, зміст яких спирається на перелік варіантів (на кшталт «Яке з наведених чисел є ірраціональним?»), показувалися як відкриті без самих варіантів і втрачали зміст. Дефект усунено, обмеживши відкритий формат самодостатніми питаннями. По-друге, окремі завдання, зібрані модулем скрапінгу, містили неповний текст умови через особливості розмітки джерела, оскільки відкриті відповіді оцінює мовна модель за змістом, а не за точним збігом, цей недолік не порушує роботи оцінювання, проте його враховано як напрям для вдосконалення розбору розмітки.

Отже, функціональне тестування підтвердило відповідність поведінки застосунку поставленим вимогам в усіх основних сценаріях використання. Виявлені під час тестування дефекти усунено, що засвідчує готовність системи до практичного застосування.

ВИСНОВКИ

У кваліфікаційній роботі досягнуто поставленої мети – спроектовано та розроблено вебзастосунок для підготовки випускників шкіл до складання Національного мультидисциплінарного тесту, що містить підсистему автоматизованої перевірки результатів з аналізом допущених помилок і відстеженням динаміки навчального прогресу.

Проаналізовано наявні програмні рішення для підготовки до стандартизованих тестів і виявлено їхні основні недоліки: відсутність персоналізованого зворотного зв'язку, обмеженість банку завдань і неможливість відстежувати прогрес здобувача впродовж тривалого часу. Виявлені недоліки покладено в основу вимог до розроблюваної системи.

Досліджено структуру та зміст НМТ, на підставі чого сформовано банк тестових завдань і реалізовано окремий модуль автоматизованого збору завдань, який відтворює сторінки джерела безголовим браузером, структуровано розбирає їхній вміст і поповнює спільну базу, виявляючи повтори.

Обґрунтовано вибір технологій: серверну частину побудовано на платформі ASP.NET Core 8 за шаблоном «Модель-Подання-Контролер», доступ до даних реалізовано засобами Entity Framework Core поверх бази даних SQLite, автентифікацію – підсистемою ASP.NET Core Identity, а клієнтську частину – із застосуванням Bootstrap, Alpine.js і FullCalendar. Систему організовано як рішення з трьох проєктів з виокремленим спільним рівнем основної логіки.

Спроектовано багаторівневу архітектуру застосунку та реляційну модель даних, а на її основі реалізовано рівні доступу до даних і бізнес-логіки з поділом на репозиторії та сервіси.

Розроблено підсистему перевірки результатів, яка по-різному опрацьовує закриті та відкриті відповіді: закриті звіряються з еталоном, а відкриті оцінюються за змістом засобами мовної моделі з формуванням

детального аналізу – що здобувач зрозумів, що пропустив і що варто повторити. Для роботи за відсутності зовнішнього сервісу передбачено резервний локальний оцінювач, що забезпечує стійкість системи. На підставі результатів тестування реалізовано аналітику прогресу з виявленням слабких тем і підсистему планування занять, що автоматично формує розклад із пар «теоретичне опрацювання – перевірка».

Проведено функціональне тестування розробленого застосунку за сценаріями використання, межовими випадками та умовами відсутності зовнішніх служб. Усі основні сценарії пройдено успішно, а виявлені під час тестування дефекти усунено.

Практична цінність роботи полягає в тому, що розроблений вебзастосунок може бути впроваджений у навчальний процес загальноосвітніх закладів або використаний для індивідуальної самопідготовки. На відміну від розглянутих аналогів, застосунок поєднує змістове оцінювання відкритих відповідей, персоналізований аналіз слабких тем і автоматичне планування підготовки, що сприяє цілеспрямованому опрацюванню навчального матеріалу.

Перспективними напрямками подальшого розвитку є розширення банку завдань і вдосконалення розбору розмітки джерел, додавання нових предметів і форматів завдань, а також розроблення мобільного клієнта для підвищення доступності системи. Окремий напрям становить удосконалення підсистеми оцінювання відкритих відповідей шляхом донавчання або тонкого налаштування мовної моделі на предметному матеріалі, що дозволить підвищити точність змістового аналізу. Доцільним є також впровадження адаптивного добору завдань, за якого складність наступних питань визначається поточним рівнем підготовки здобувача, та інтеграція із системами управління навчанням для застосування у складі цілісного освітнього середовища.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Freeman A. ASP .NET Core MVC in context. Pro ASP.NET Core MVC
2. Berkeley, CA, 2017. 310 p. URL: https://doi.org/10.1007/978-1-4842-3150-0_1 (date of access: 12.06.2026).
2. Fowler M. Patterns of enterprise application architecture. Boston : Addison-Wesley, 2002. 533 p.
3. Dependency injection in ASP.NET Core. *Microsoft Learn: Build with answers in reach.* URL: <https://learn.microsoft.com/aspnet/core/fundamentals/dependency-injection> (date of access: 12.06.2026).
4. Playwright for .NET. *Fast and reliable end-to-end testing for modern web apps | Playwright.* URL: <https://playwright.dev/dotnet/> (date of access: 19.06.2026).
5. AngleSharp: library for parsing HTML. *AngleSharp - Documentation.* URL: <https://anglesharp.github.io/> (date of access: 19.06.2026).
6. Polly: resilience and transient-fault-handling library for .NET. The polly project. URL: <https://www.thepollyproject.org/> (date of access: 19.06.2026).
7. Claude API: messages. *Anthropic.* URL: <https://docs.anthropic.com/> (date of access: 20.06.2026).
8. Mann P. S. Introductory statistics. 9th ed. Hoboken : John Wiley & Sons, 2016. 672 p.
9. Entity framework core. *Microsoft Learn.* URL: <https://learn.microsoft.com/ef/core/> (date of access: 15.06.2026).
10. Date C. J. An introduction to database systems. 8th ed. Boston : Addison-Wesley, 2003. 1024 p.
11. Booch, Rumbaugh, Jacobson. The unified modeling language user guide. 2nd ed. Boston : Addison-Wesley, 2005. 496 p.

12. Національний мультипредметний тест. *Український центр оцінювання якості освіти*. URL: <https://testportal.gov.ua/> (дата звернення: 09.06.2026).

13. C# Guide - .NET managed language - C#. *Microsoft Learn: Build with answers in reach*. URL: <https://learn.microsoft.com/dotnet/csharp/> (date of access: 11.06.2026).

14. ASP.NET Core fundamentals overview. *Microsoft Learn: Build with answers in reach*. URL: <https://learn.microsoft.com/aspnet/core/fundamentals> (date of access: 11.06.2026).

15. SQLite documentation. *SQLite*. URL: <https://www.sqlite.org/docs.html> (date of access: 16.06.2026).

16. Introduction to identity on ASP.NET Core. *Microsoft Learn: Build with answers in reach*. URL: <https://learn.microsoft.com/aspnet/core/security/authentication/identity> (date of access: 17.06.2026).

17. Background tasks with hosted services in ASP.NET Core. *Microsoft Learn*. URL: <https://learn.microsoft.com/aspnet/core/fundamentals/host/hosted-services> (date of access: 18.06.2026).

18. Cockburn A. Writing effective use cases. Boston : Addison-Wesley, 2001. 270 p.

19. Martin R. C. Clean architecture: a craftsman's guide to software structure and design. Boston : Prentice Hall, 2017. 432 p.

20. Mitchell R. Web Scraping with Python. 2nd ed. Sebastopol : O'Reilly Media, 2018. 308 p.

21. Bootstrap. *Bootstrap · the most popular HTML, CSS, and JS library in the world*. URL: <https://getbootstrap.com/docs/> (date of access: 20.06.2026).

22. FullCalendar. *FullCalendar*. URL: <https://fullcalendar.io/docs> (date of access: 20.06.2026).