Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

Кваліфікаційна робота

Освітній ступінь – магістр

На тему: «РОЗРОБКА ЕЛЕКТРОННОГО КУРСУ «ВСТУП ДО ПРОГРАМУВАННЯ» »

Виконав: студент 2-го року навчання освітньої програми «Комп’ютерні науки»,
спеціальності 122 Комп’ютерні науки
СЕМЕНЕНКО Ілля В’ячеславович

Керівник:
ГЛИБОВЕЦЬ Микола Миколайович
доктор фіз-мат наук, професор

Кваліфікаційна робота захищена

з оцінкою _____

Секретар ЕК _____

«____» _____ 2025 р.

Київ – 2025

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ

Зав.кафедри інформатики,
кандидат фіз-мат наук, доцент
Гороховський Семен Самуїлович

_____

(підпис)

«___»_____2025 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на кваліфікаційну роботу

студенту 2 р.н. магістерської програми Комп'ютерні науки
Семененку Іллі В'ячеславовичу

Розробити електронний курс «Вступ до програмування»

Зміст ТЧ до магістерської роботи:

Зміст
Анотація
Вступ
1. Аналіз предметної області електронного курсу «вступ до програмування»
2. Проєктування архітектури та методологічних основ електронного курсу «вступ до програмування»
3. Висновки та рекомендації щодо використання електронного курсу «вступ до програмування»
Список використаних джерел
Додаток А. Основні логічні файли backend коду
Додаток Б. Основні логічні файли frontend коду

Дата видачі „___" _____ 2025 р.

Керівник _____

М. М. ГЛИБОВЕЦЬ, доктор фіз-мат наук, професор. (підпис)

Завдання отримав _____

І. В. СЕМЕНЕНКО (підпис)

Тема: Розробка електронного курсу "Вступ до програмування"

Календарний план виконання роботи:

№ п/п	Назва етапу дипломного проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на дипломну роботу.	25.11.2024	
2.	Аналіз предметної області та освітніх технологій.	15.02.2025	
3.	Проєктування архітектури програмного забезпечення та моделі даних.	20.03.2025	
4.	Реалізація Backend електронного курсу.	01.04.2025	
5.	Реалізація Frontend електронного курсу.	20.04.2025	
6.	Розробка контенту, тестів та функції кастомізації.	30.04.2025	
7.	Комплексне тестування програмного продукту.	05.05.2025	
8.	Написання та оформлення пояснювальної записки.	22.05.2025	
9.	Підготовка презентації та доповіді до попереднього захисту.	29.05.2025	
10.	Корегування роботи за результатами попереднього захисту.	1.06.2025	
11.	Остаточне оформлення пояснювальної роботи та слайдів.	3.06.2025	
12.	Захист магістерської роботи (проєкту)	12.06.2025	

Студент СЕМЕНЕНКО Ілля В’ячеславович

Керівник ГЛИБОВЕЦЬ Микола Миколайович

“_____”_____ 2025 р.

ЗМІСТ

Анотація

Магістерська робота присвячена розробці інтерактивного електронного курсу «Вступ до програмування» на JavaScript для школярів. Метою стало створення адаптованого україномовного онлайн-ресурсу з оригінальним контентом та можливістю кастомізації тестів викладачами.

Робота включала аналіз освітніх методик, проєктування масштабованої клієнт-серверної архітектури (Next.js, .NET 8) на основі педагогічних засад професора Миколи ГЛИБОВЦЯ, розробку програмних модулів, наповнення контентом та тестування.

Результатом є готовий до впровадження електронний курс, що забезпечує інтерактивне вивчення JavaScript, містить україномовний контент, систему тестування та функцію кастомізації питань викладачами. Платформа побудована на гнучкій, масштабованій архітектурі.

Ключові слова: електронний курс, вступ до програмування, JavaScript, навчання школярів, інтерактивне навчання, онлайн-платформа, україномовний контент, кастомізація контенту, веб-розробка.

ВСТУП

Актуальність теми: швидкий розвиток інформаційних технологій та цифровізація всіх сфер життя зумовлюють зростаючу потребу у фахівцях з програмування. Для успішної інтеграції молоді у сучасне суспільство та підготовки до майбутніх професій, критично важливим є раннє опанування основ програмування. Однак, наразі спостерігається нестача якісних, інтерактивних та адаптованих до вікових особливостей україномовних освітніх ресурсів, які б ефективно залучали школярів до вивчення програмування з нуля. Розробка такого електронного курсу є актуальним завданням, що сприятиме підвищенню цифрової грамотності та розвитку логічного мислення учнів.

Мета і завдання дослідження. Розробка та впровадження інтерактивного електронного курсу "Вступ до програмування" на базі JavaScript, адаптованого для учнів загальноосвітніх шкіл, з акцентом на оригінальний україномовний контент та можливість розширення навчального матеріалу викладачами.

Завдання роботи:

- Проаналізувати та узагальнити сучасні підходи до навчання програмуванню для школярів, а також існуючі онлайн-платформи та їхні методики.
- Розробити педагогічно-методичні основи створення україномовного інтерактивного курсу з програмування, що враховуватиме особливості сприйняття інформації школярами.
- Спроектувати архітектуру та функціональні модулі програмного забезпечення електронного курсу відповідно до сучасних стандартів веб-розробки.
- Реалізувати бекенд- та фронтенд-частини електронного курсу, забезпечивши інтерактивність та зручність користувацького інтерфейсу.
- Наповнити курс оригінальним україномовним контентом, що охоплює основи JavaScript, а також розробити інтерактивні тестові завдання різних типів.

- Впровадити функціонал для адміністраторів/викладачів щодо додавання та редагування кастомних тестових питань.
- Провести тестування розробленого програмного продукту та оцінити його відповідність поставленим вимогам та ефективність для цільової аудиторії.

Об'єкт дослідження. Процес навчання учнів загальноосвітніх шкіл основам програмування.

Методи дослідження:

- Теоретичні: Аналіз, синтез, узагальнення – для вивчення педагогічних та методологічних засад навчання програмуванню, а також для порівняння існуючих освітніх платформ та технологічних рішень. Системний аналіз – для проектування архітектури програмного забезпечення.
- Емпіричні: Проектування, моделювання – для розробки структури курсу та інтерфейсу. Програмна реалізація – для створення функціоналу курсу. Тестування (модульне, інтеграційне, функціональне, юзабіліті) – для перевірки працездатності та ефективності розробленого рішення.

Джерела дослідження. Основним фактичним матеріалом для дослідження та розробки курсу виступають навчальні посібники та методичні рекомендації з програмування для початківців, адаптовані для шкільного віку, зокрема, книга "Вступ до дитячого програмування. Мова JavaScript" професора Миколи ГЛИБОВЦЯ [1]. Крім того, використовуються стандарти та документація мови програмування JavaScript (ECMAScript [2] специфікації, MDN Web Docs [3]) та обраних технологій розробки (.NET, Next.js, React, SQL Server). Для порівняльного аналізу та вивчення кращих практик інтерактивного навчання, аналізуватимуться структура та функціонал провідних світових та українських освітніх платформ, що пропонують курси з програмування, а також інструменти для створення інтерактивних завдань, такі як Kahoot, Mentimeter, Quizlet, Nearpod, Socrative.

Наукова новизна одержаних результатів.

- Створено архітектуру масштабованої інтерактивної освітньої платформи, яка базується на сучасних принципах N-Tier Architecture, патернах

проектування, що забезпечує гнучкість та можливість подальшого розширення курсу.

- Впроваджено механізм кастомізації контенту для викладачів, що дозволяє формувати індивідуальні набори тестових питань та адаптувати навчальний процес до специфічних потреб учнів та шкільних програм.

Практичне значення одержаних результатів. Результати магістерської роботи полягають у створенні готового до використання електронного курсу "Вступ до програмування", який може бути ефективно застосований у навчальному процесі загальноосвітніх шкіл як інструмент для початкового вивчення програмування, так і для самостійного поглиблення знань учнями. Розроблена платформа забезпечить викладачів гнучким інструментарієм для контролю знань, створюючи умови для персоналізованого навчання.

Перелік прийнятих скорочень

API (Application Programming Interface) – прикладний програмний інтерфейс

BLL (Business Logic Layer) – шар (рівень) бізнес-логіки

CORS (Cross-Origin Resource Sharing) – механізм спільного використання ресурсів між різними джерелами

CRUD (Create, Read, Update, Delete) – базові операції для роботи з даними (Створити, Прочитати, Оновити, Видалити)

CSS (Cascading Style Sheets) – каскадні таблиці стилів

DAL (Data Access Layer) – шар (рівень) доступу до даних

DTO (Data Transfer Object) – об'єкт передачі даних

ER-діаграма (Entity-Relationship diagram) – діаграма "сутність-зв'язок"

FSD (Feature-Sliced Design) – методологія проєктування архітектури клієнтської частини застосунку

HTTP (HyperText Transfer Protocol) – протокол передачі гіпертексту

JSON (JavaScript Object Notation) – текстовий формат обміну даними, заснований на JavaScript

JWT (JSON Web Tokens) – веб-токени у форматі JSON для безпечної передачі автентифікаційних даних

MDN (MDN Web Docs) – веб-документація для розробників від Mozilla (раніше Mozilla Developer Network)

N-Tier Architecture – багаторівнева архітектура програмного забезпечення

ORM (Object-Relational Mapper) – технологія програмування, яка зв'язує бази даних з концепціями об'єктно-орієнтованих мов програмування

REST (Representational State Transfer) – архітектурний стиль взаємодії компонентів розподіленого застосунку в мережі

SEO (Search Engine Optimization) – пошукова оптимізація

SOLID – акронім для п'яти принципів об'єктно-орієнтованого дизайну (Single Responsibility, Open/Closed, Liskov Substitution, Interface Segregation, Dependency Inversion)

SQL (Structured Query Language) – мова структурованих запитів

SSR (Server-Side Rendering) – рендеринг (відображення) на стороні сервера

SSG (Static Site Generation) – генерація статичних сайтів

UI (User Interface) – користувацький інтерфейс

URI (Uniform Resource Identifier) – уніфікований ідентифікатор ресурсу

WCAG (Web Content Accessibility Guidelines) – настанови з доступності веб-контенту

.NET – платформа для розробки програмного забезпечення компанії Microsoft

БД – База Даних

ООП – Об'єктно-орієнтоване програмування

СУБД – Система управління базами даних

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ЕЛЕКТРОННОГО КУРСУ "ВСТУП ДО ПРОГРАМУВАННЯ"

Розробка ефективного електронного курсу "Вступ до програмування" для школярів вимагає глибокого аналізу предметної області. Цей розділ присвячений дослідженню сучасних підходів до навчання програмуванню дітей шкільного віку, аналізу існуючих інтерактивних інструментів та формуванню теоретичного підґрунтя для створення курсу, що відповідатиме освітнім потребам цільової аудиторії та сучасним технологічним стандартам.

1.1 Огляд сучасних підходів до навчання програмуванню для школярів

Навчання програмуванню дітей шкільного віку є актуальним завданням сучасної освіти, що сприяє розвитку логічного мислення, алгоритмічних навичок та творчого потенціалу. Однак, підходи до викладання програмування для цієї вікової категорії мають свої специфічні особливості, зумовлені психологічними та когнітивними характеристиками учнів.

Ключовою проблемою при розробці навчальних матеріалів з програмування для дитячої аудиторії є адаптація подання інформації. Як слушно зауважено у підручнику "Вступ до дитячого програмування. Мова JavaScript" проф. М. ГЛИБОВЦЯ [1], математика, що стала еталоном мислення завдяки строгості формальних доведень, у своєму класичному викладі не завжди є оптимальною для школярів. Традиційне подання навчальних матеріалів, що включає теоретичний огляд (постановку задачі, історію розв’язання, необхідний математичний апарат), формалізацію алгоритму, аналіз його ефективності та особливостей, а також опис техніки програмної реалізації, може виявитися надто складним для сприйняття дітьми.

Тому, при розробці курсу "Вступ до програмування" для школярів, акцент зміщується на використання простих алгоритмів та ігрових задач. Такий підхід дозволяє зацікавити учнів та зробити процес навчання більш інтерактивним та зрозумілим. Навчальний матеріал, як пропонується в підручнику "Вступ до дитячого програмування. Мова JavaScript" М. ГЛИБОВЦЯ [1], має надавати змогу ознайомитися з базовими поняттями, такими як будова комп’ютера, основи програмування (типи даних, керівні структури, функції), елементи як класичного, так і веб-програмування, а також процес обробки різноманітної інформації.

У цьому контексті, вибір мови програмування JavaScript як основи для вступного курсу є обґрунтованим з кількох причин.

По-перше, JavaScript значно спрощує висвітлення процесів вводу вхідних даних (вхідної інформації) і виведення результатів обробки.

По-друге, ця мова дозволяє легко пояснити основні оператори управління програмою та функції.

По-третє, для початку програмування на JavaScript достатньо мати лише встановлений браузер, що значно спрощує взаємодію з ним та не відлякує незрозумілими для дітей назвами бібліотек, параметрів встановлення абощо.

Далі в цьому підрозділі будуть детальніше розглянуті специфічні методики викладання.

1.1.1 Дослідження методик викладання програмування для учнів методом візуального блочного програмування

Одним із найпопулярніших та ефективних підходів до навчання програмування дітей молодшого та середнього шкільного віку є використання методу візуального блочного програмування. Цей підхід дозволяє учням створювати програми, перетягуючи та з'єднуючи графічні блоки, що представляють різні команди, цикли, умови та змінні. Таким чином, основний акцент зміщується з вивчення складного синтаксису конкретної мови

програмування на розуміння фундаментальних концепцій програмування та розвиток алгоритмічного мислення.

Найвідомішими представниками середовищ візуального блочного програмування є Scratch (розроблений MIT Media Lab) та Blockly (розроблений Google). Scratch, зокрема, здобув величезну популярність завдяки своєму інтуїтивно зрозумілому інтерфейсу, великій спільноті користувачів та можливості створювати інтерактивні історії, ігри та анімації. Blockly, у свою чергу, є бібліотекою, яка дозволяє розробникам вбудовувати візуальний редактор коду у власні веб-застосунки, що робить її гнучким інструментом для створення освітніх ресурсів.

Переваги методу візуального блочного програмування для:

- Низький поріг входження: Відсутність необхідності запам'ятовувати складний синтаксис та команди дозволяє дітям швидко почати створювати власні програми та бачити результати своєї роботи. Це мінімізує фрустрацію на початкових етапах та підвищує мотивацію.
- Наочність та інтерактивність: Графічне представлення коду та можливість миттєво бачити, як зміни в блоках впливають на виконання програми (наприклад, на рух персонажа на сцені), робить процес навчання більш захопливим та зрозумілим.
- Фокус на логіці та алгоритмах: Учні концентруються на послідовності дій, умовах, циклах та інших логічних конструкціях, що сприяє розвитку навичок обчислювального мислення.
- Запобігання синтаксичним помилкам: Оскільки блоки можна з'єднувати лише у логічно правильних комбінаціях, виключається значна кількість типових помилок новачків, пов'язаних із друкарськими помилками чи неправильним синтаксисом.
- Розвиток творчості та навичок проєктної діяльності: Середовища візуального програмування часто заохочують учнів до створення власних

проєктів, що сприяє розвитку креативності, планування та навичок вирішення проблем.

Незважаючи на численні переваги, методика візуального блочного програмування має і певні обмеження та потенційні недоліки при розгляді її як єдиного чи довготривалого інструменту навчання:

- Обмеженість у складних проєктах: Для створення великих та складних програм візуальні блоки можуть стати громіздкими та менш ефективними порівняно з текстовим кодом.
- "Стеля" можливостей: Функціонал блочних середовищ, хоч і широкий, але все ж обмежений порівняно з можливостями повноцінних текстових мов програмування.
- Труднощі переходу до текстового програмування: Після тривалого використання блочного програмування учням може бути складно адаптуватися до необхідності суворого дотримання синтаксису, роботи з абстрактнішими поняттями та менш візуалізованим середовищем текстових мов. Це явище іноді називають "Scratch cliff" (урвище Scratch) або "Blockly cliff".
- Віддаленість від професійного програмування: Хоча концепції програмування є універсальними, інструментарій та підходи у професійній розробці суттєво відрізняються від блочних середовищ.

1.1.2 Дослідження методик викладання програмування для учнів методом текстового коду

Після ознайомлення з основами програмування за допомогою візуальних блочних середовищ або ж як альтернативний шлях для старших школярів, часто відбувається перехід до вивчення програмування методом текстового коду. Цей підхід передбачає написання програм за допомогою мов програмування, де команди, структури та логіка виражаються через послідовності текстових символів, що відповідають строгому синтаксису конкретної мови. Текстове

програмування є стандартом у професійній розробці програмного забезпечення та надає значно ширші можливості для створення складних та багатофункціональних проєктів.

Серед мов програмування, які часто обирають для першого знайомства школярів з текстовим кодом, популярними є Python та JavaScript. Python приваблює своїм відносно простим та читабельним синтаксисом, що нагадує англійську мову, а також великою кількістю бібліотек для різноманітних задач. JavaScript, у свою чергу, є незамінним для веб-розробки, дозволяючи створювати інтерактивні елементи на веб-сторінках і бачити результати своєї роботи безпосередньо у браузері, що може бути дуже мотивуючим для учнів. Саме JavaScript обрано як основу для курсу, що розробляється в рамках даної магістерської роботи, з огляду на його інтерактивність, візуальну віддачу та актуальність у сучасних веб-технологіях.

Переваги вивчення програмування методом текстового коду для школярів включають:

- Наближеність до професійного програмування: учні отримують досвід роботи з інструментами та концепціями, які використовуються у реальній індустрії розробки програмного забезпечення.
- Розвиток синтаксичної грамотності та уваги до деталей: необхідність дотримуватися правил синтаксису мови програмування тренує уважність, точність та логічне мислення.
- Глибше розуміння роботи комп'ютера: текстове програмування часто вимагає більш явного управління певними аспектами програми, що сприяє кращому розумінню того, як комп'ютер виконує інструкції.
- Ширші можливості для реалізації складних ідей: текстові мови надають більшу гнучкість та потужність для створення складних алгоритмів та проєктів, не обмежуючись функціоналом візуальних конструкторів.
- Підготовка до подальшого навчання та кар'єри: опанування текстових мов програмування відкриває шлях до вивчення більш спеціалізованих

мов та технологій, а також є важливою навичкою для багатьох сучасних професій.

Попри значні переваги, навчання текстовому програмуванню школярів пов'язане з певними викликами та труднощами:

- Вищий поріг входження: Необхідність вивчення та суворого дотримання синтаксису може бути складною для початківців, особливо для молодших школярів.
- Синтаксичні помилки та їх пошук (дебагінг): На відміну від блочного програмування, де синтаксичні помилки практично виключені, у текстовому коді вони є поширеним явищем. Процес пошуку та виправлення помилок може викликати фрустрацію.
- Менша наочність на початкових етапах: Результати виконання простих консольних програм можуть здаватися менш захопливими порівняно з інтерактивними анімаціями у Scratch.
- Потреба в абстрактному мисленні: Розуміння таких концепцій, як змінні, типи даних, функції, вимагає певного рівня абстрактного мислення.

Для вирішення цих проблем, у підручнику, за яким створюється цей курс [1] застосовані наступні методичні підходи:

- Поступове ускладнення матеріалу ("від простого до складного"): курс структуровано таким чином, щоб нові концепції вводилися поступово, спираючись на вже засвоєні знання.
- Використання зрозумілих прикладів та аналогій: складні концепції пояснюються через прості приклади з повсякденного життя або ігрові сценарії.
- Використання детального опису алгоритмів дій з візуальним супроводом: багато речей можуть здаватися доволі очевидним для дорослої людини або людини з досвідом у програмуванні, тому при написанні курсу можуть бути ненавмисно описані з недостатнім рівнем деталізації.
- Оригінальний україномовний контент: надання матеріалів рідною мовою спрощує розуміння та засвоєння інформації.

В контексті розробки електронного курсу "Вступ до програмування" на базі JavaScript, було прийнято рішення зосередитись на текстовому програмуванні. Це рішення зумовлене метою підготувати учнів до подальшого, більш глибокого вивчення мов програмування, які використовуються в реальних проєктах та веб-розробці. JavaScript, як текстова мова, дозволяє з самого початку знайомити учнів з синтаксичними конструкціями, структурою коду та інструментами, ближчими до тих, з якими вони можуть зіткнутися в майбутньому.

1.2 Аналіз існуючих онлайн-інструментів для інтерактивного навчання

1.2.1 Kahoot!

Почнемо огляд аналогів засобу тестування з одного з найрозповсюдженіших засобів онлайн тестування Kahoot.

Kahoot! — це популярна платформа для гейміфікованого навчання та вікторин, яка широко використовується у школах та університетах для підвищення залученості студентів. Платформа має обширний функціонал та декілька рівнів монетизації (рис. 1.2.2-1)

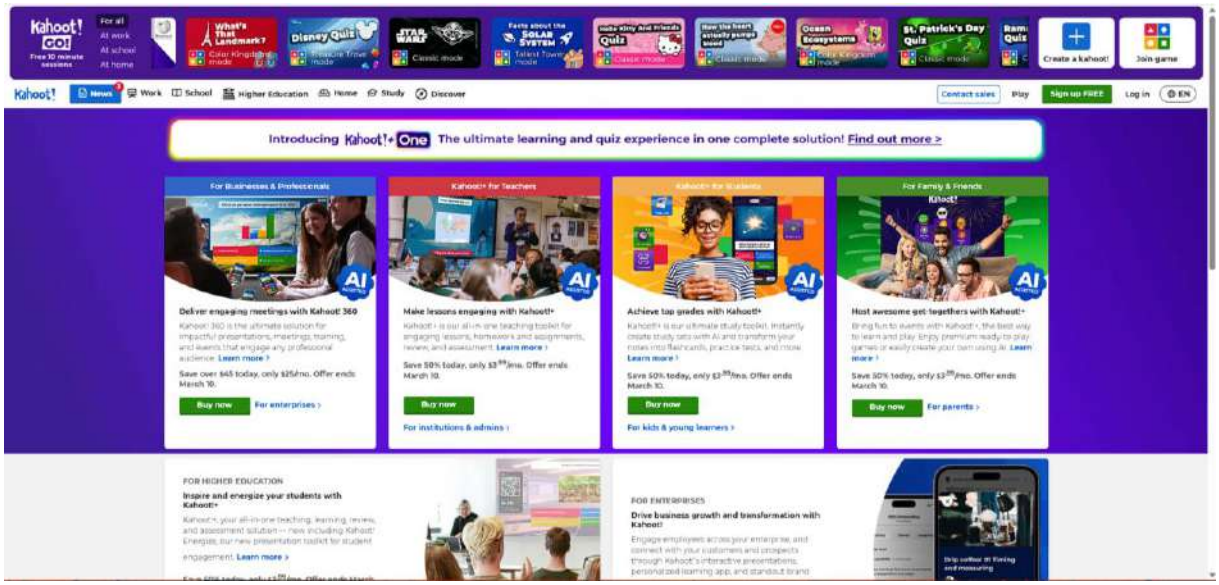

Рисунок 1.1

Розглянемо функціонал, який пов’язаний з нашою роботою, а саме створення тестів, для розуміння їх вигляду та можливості подачі завдань з найбільш цікавою подачею (рис. 1.2.1-2, 1.2.1-3).

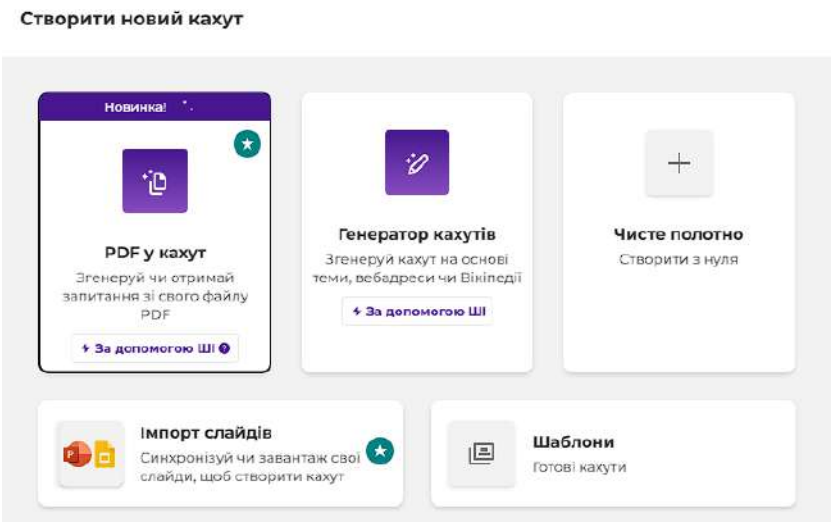

Рисунок 1.2

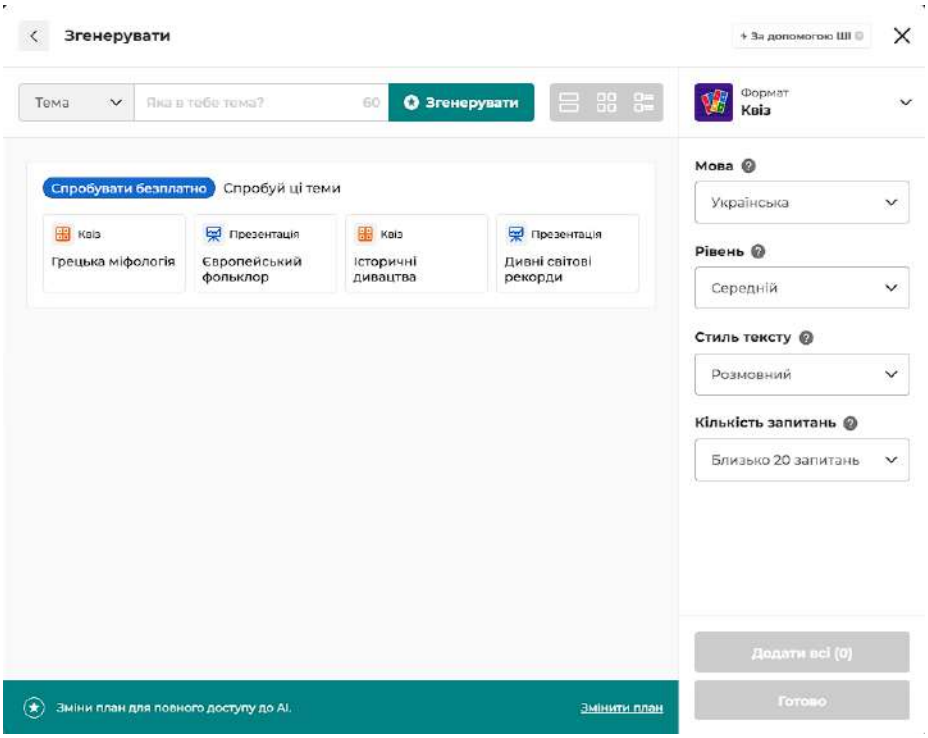

Рисунок 1.3

Як ми бачимо, kahoot використовує багато інтеграцій зі штучним інтелектом для геренації тестів з готового матеріалу, що є доволі корисним інструментом для створення тестів на основі готового матеріалу, та також має готові тести серед яких можна обрати собі шаблон.

Серед того, що нас цікавить варто звернути увагу на подачу та типи доступних тестів(рис. 1.2.1-4, 1.2.1-5).

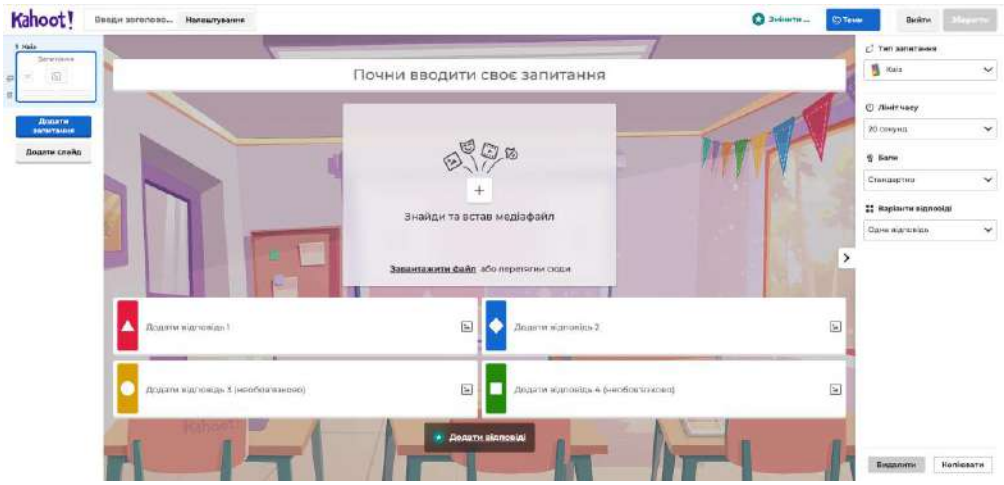

Рисунок 1.4

Рисунок 1.5

Як ми бачимо у нас є мінімалістичний яскравий дизайн, який не відволікає і заодно додає додаткової «цікавості» сторінці(рис. 4). Як додаткові мітки у варіантах відповідай розташовані фігури та кольорові блоки, що зменшує ризик переплутати варіант відповіді. Є доволі непоганим рішенням, на відміну від звичайних пустих білих блоків з монотонним чорним текстом. Також є керований таймер на кожне завдання.

Серед методів тестування присутні(рис. 5):

- Квіз (рис. 1.2.1-4)

Учасники тесту повинні обрати коректну відповідь/відповіді на питання.

- Так або ні (рис. 1.2.1-6)

Учасники тесту повинні обрати відповідь так або ні, в залежності від питання

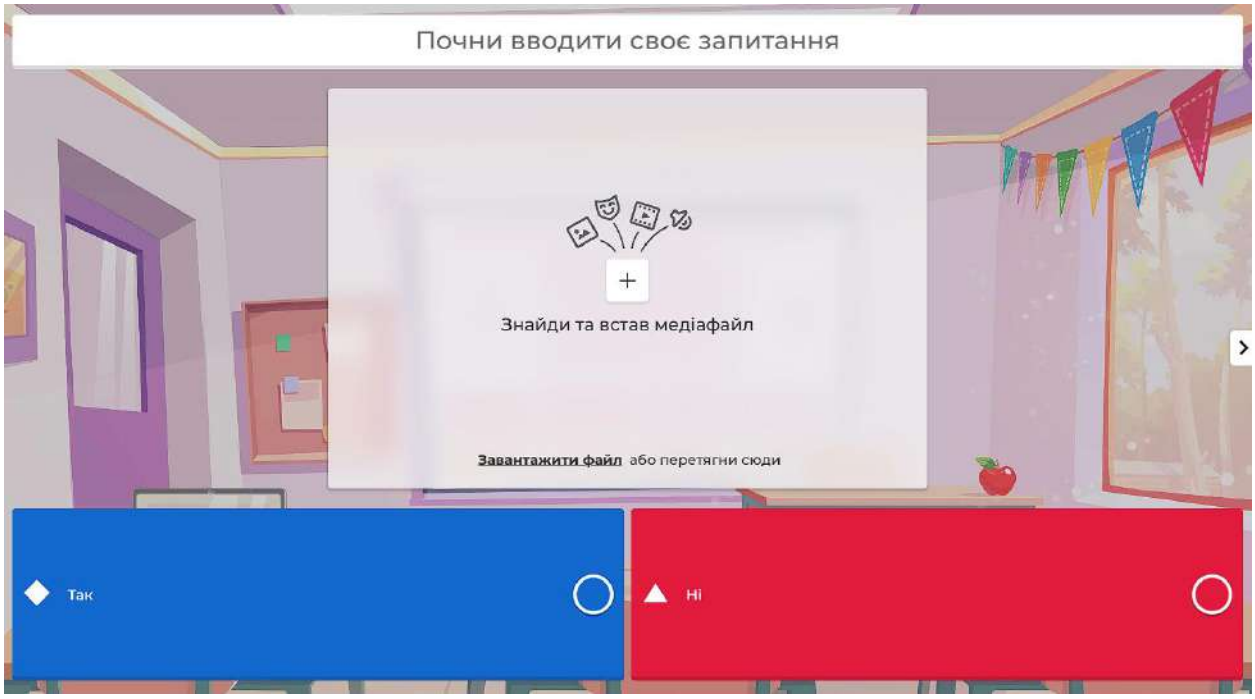

Рисунок 1.6

- Письмові відповідь (рис. 1.2.1-7)

Учасники тесту повинні правильно записати відповідь на питання

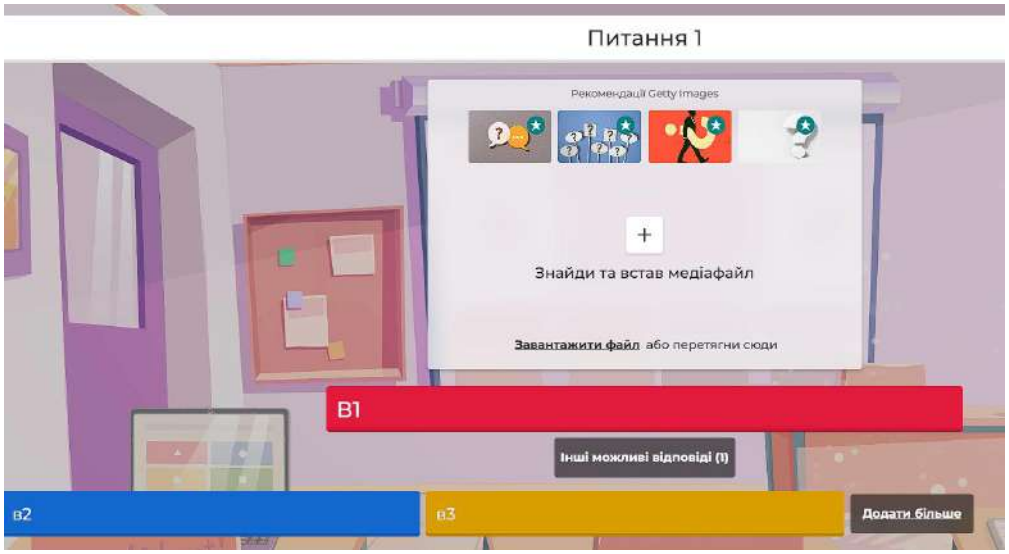

Рисунок 1.7

- Слайдер (рис. 1.2.1-8)

Учасники тесту повинні правильно вказати значення з діапазону. Також можливо вказати наскільки широкий діапазон відповідей зараховується як правильний

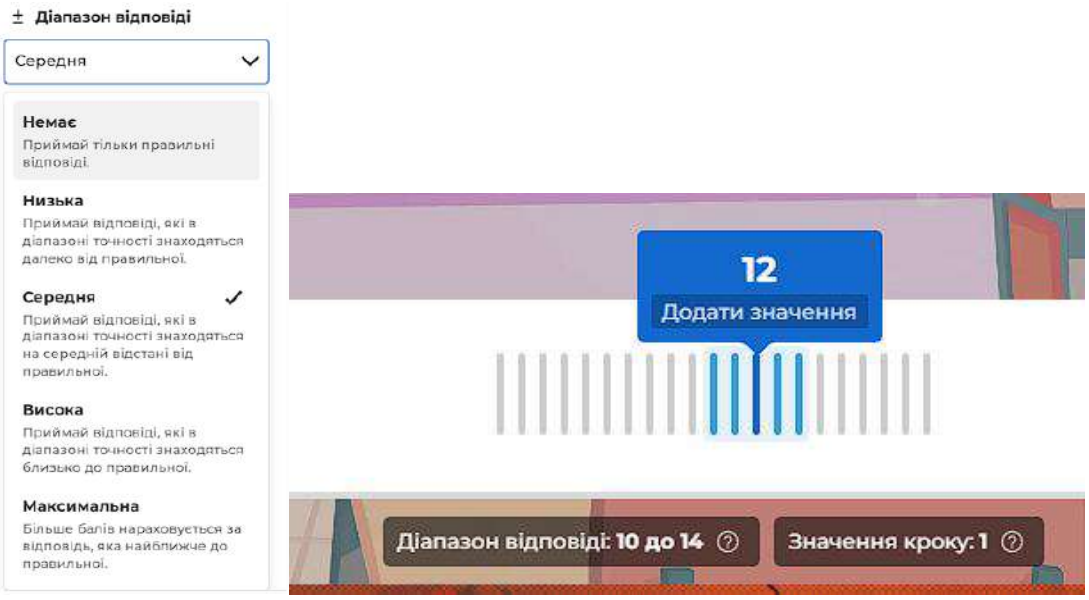

Рисунок 1.8

- Відповідь-мітка (рис. 1.2.1-9)

Учасники тесту повинні правильно відмітити ділянку/місця на заданому фото

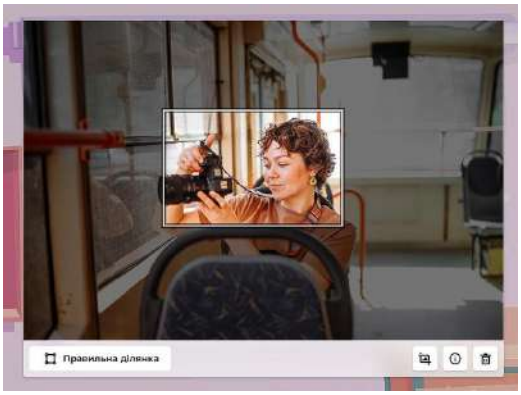

Рисунок 1.9

- Пазл (рис. 1.2.1-10)

Учасники тесту повинні впорядкувати відповіді згідно з їм правильним порядком

Рисунок 1.10

Особливості тестової частини програми:

- Використання гейміфікації: учасники змагаються за бали та місця у рейтингу.
- Можливість додавання зображень, відео та аудіофайлів.
- Яскравий дизайн, наявність великої кількості анімацій.

1.2.2 Mentimeter

Mentimeter — це онлайн-платформа для створення інтерактивних презентацій і опитувань. Хоч система тестування не є основним продуктом застосунку, вона виступає хорошим доповненням до функціоналу, і використовується в основному для отримання думки аудиторії/брейнштorму. Розглядаючи дизайн застосунку та готових шаблонів, проглядається концентрація на бізнес-аудиторії та корпоративному середовищі. Продукт орієнтований на використання під час робочих зустрічей, конференцій, тренінгів та стратегічних сесій.

При першому вході сайт пропонує створити презентацію, яка тут називається «menti»(рис. 1.2.2-1).

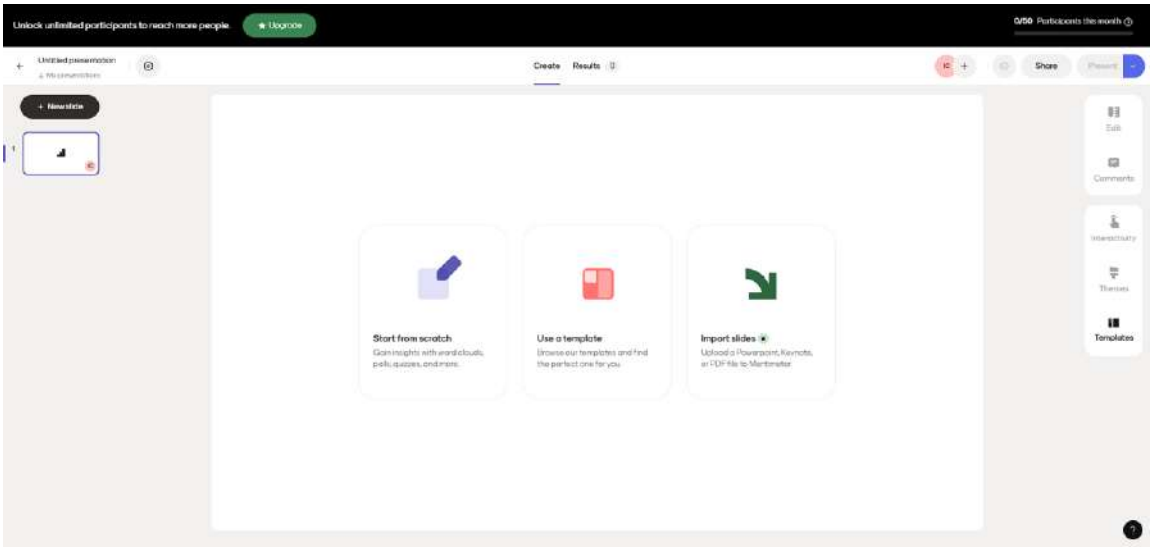

Рисунок 1.11

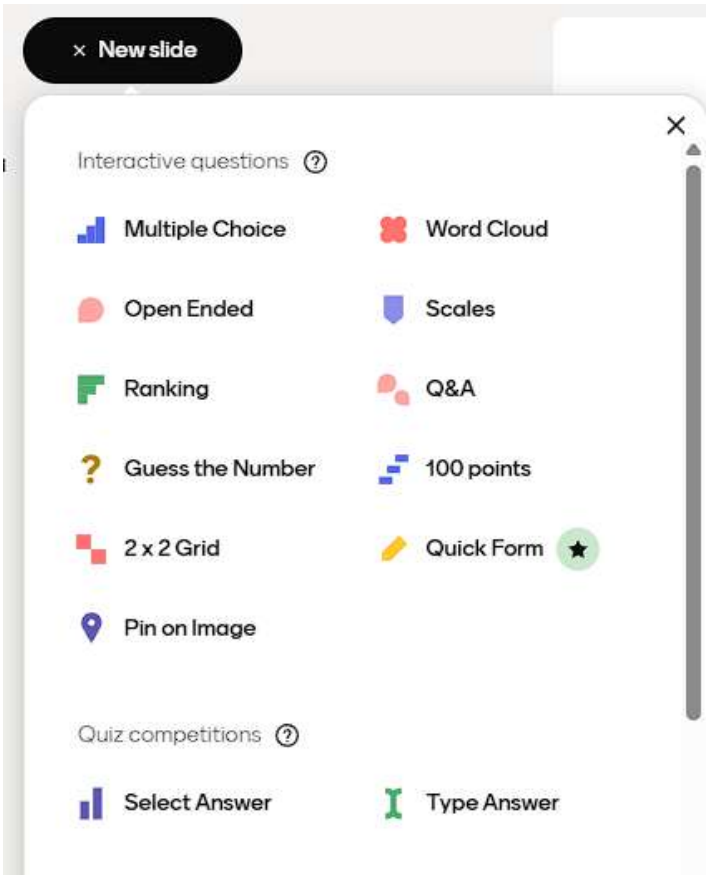

Рисунок 1.12

Серед функціоналу що нас цікавить наявні(рис. 12):

- Вибір декількох варіантів відповідей (рис. 1.2.2-3, 1.2.2-4, 1.2.2-5, 1.2.2-6).

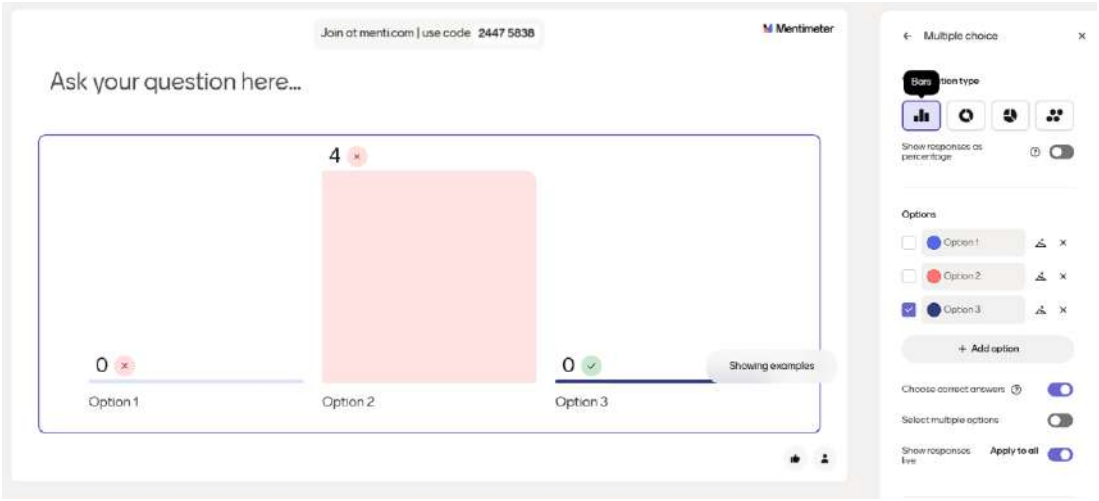

Рисунок 1.13

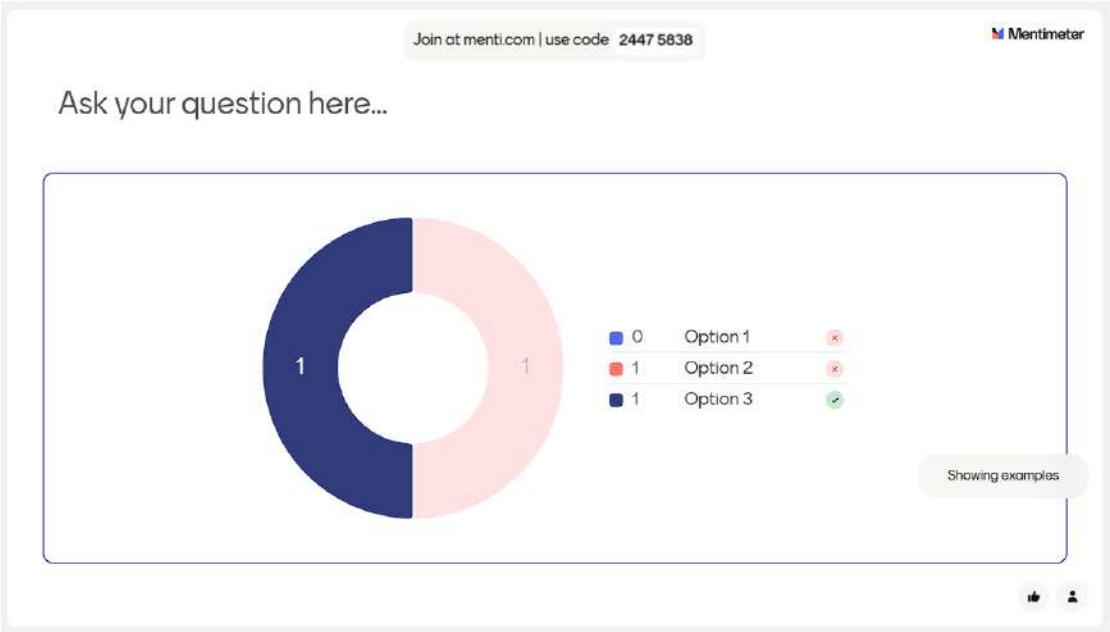

Рисунок 1.14

Рисунок 1.15

Рисунок 1.16

- Хмара слів (рис. 1.2.2-7).

Демонструє слова які частіше за все пишуть учасники вікторини, збільшуючи розмір слова відносно кількості учасників що обрали цей варіант.

Рисунок 1.17

- Відкриті питання (рис. 1.2.2-8).

Дозволяє всім учасникам надавати свої ідеї та пропозиції стосовно заданого питання.

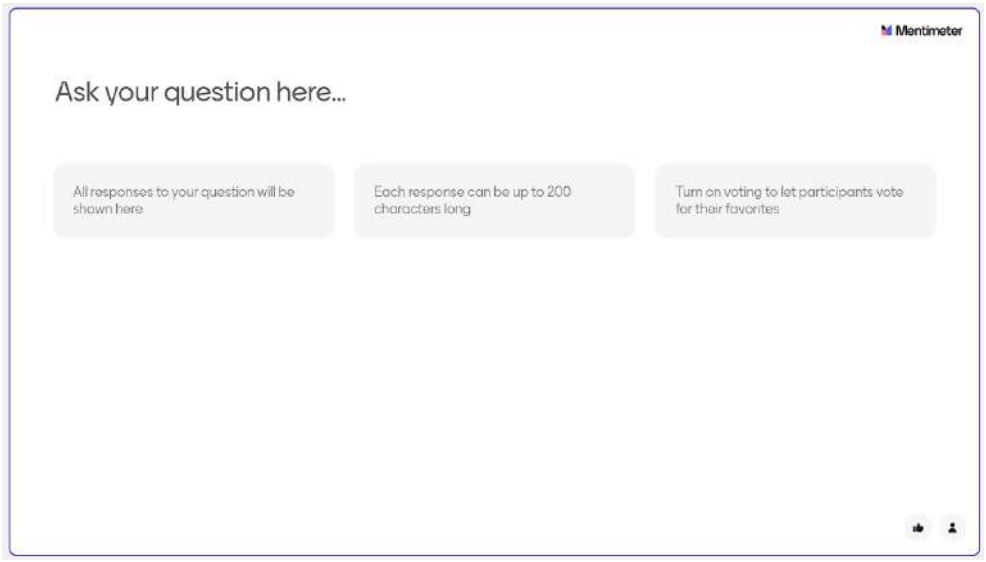

Рисунок 1.18

- Шкала (рис. 1.2.2-9, 1.2.2-10).

Інструмент що дозволяє дізнатися позицію учасників стосовно певних тверджень.

Рисунок 1.19

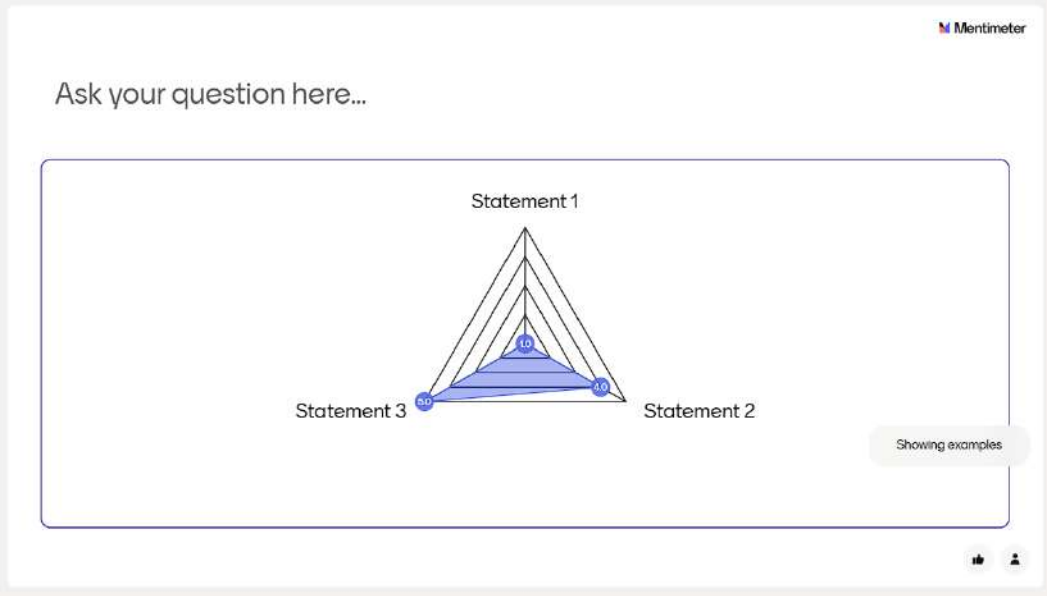

Рисунок 1.20

- Рейтинг (рис. 1.2.2-11).

Дозволяє визначити пріоритет завдань через впорядкування цілей за їх доречнісю учасниками.

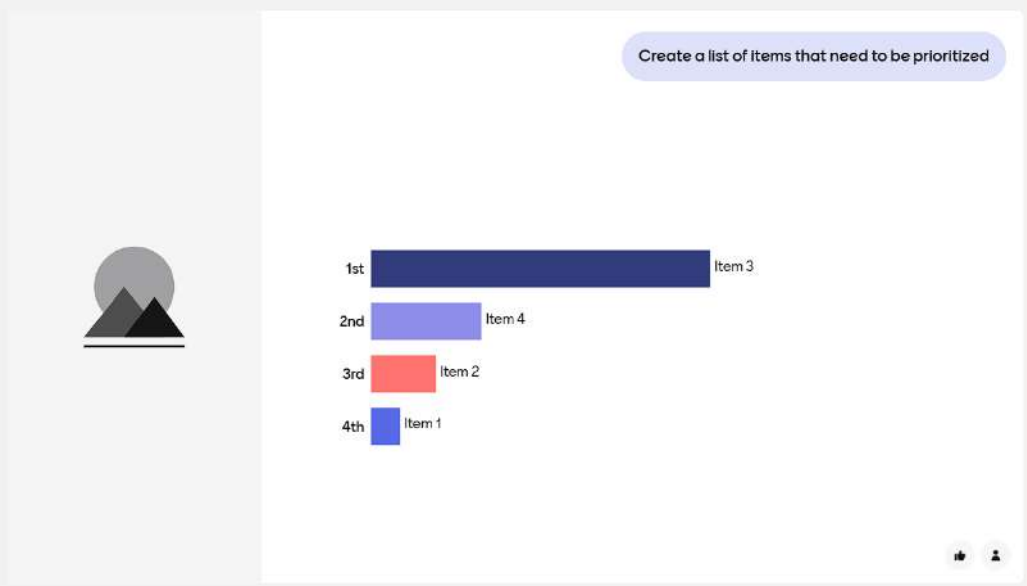

Рисунок 1.21

- Q&A (рис. 1.2.2-12).

Інструмент для отримання запитань від аудиторії та відповіді на них у усному форматі.

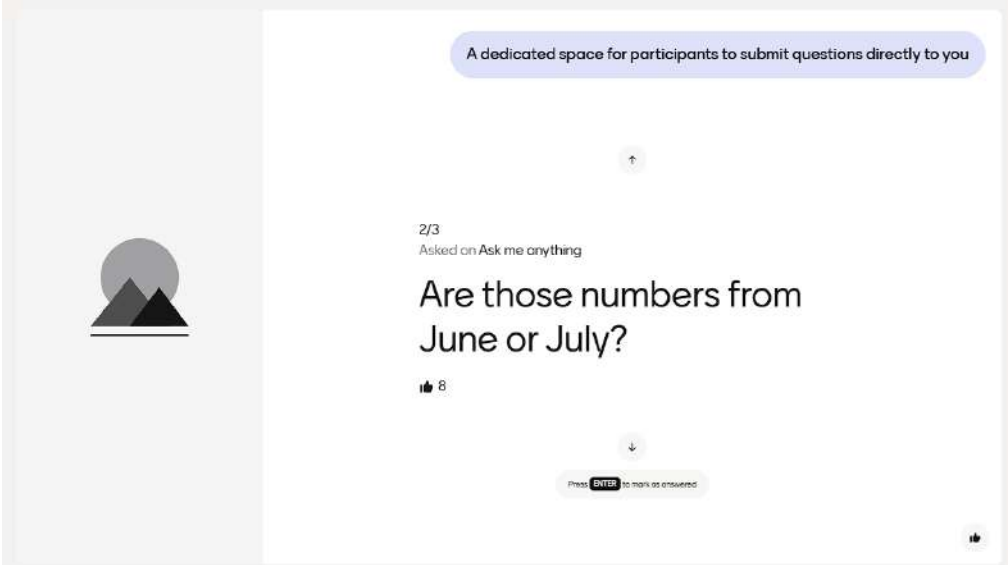

Рисунок 1.22

- Обери число(рис. 1.2.2-13).

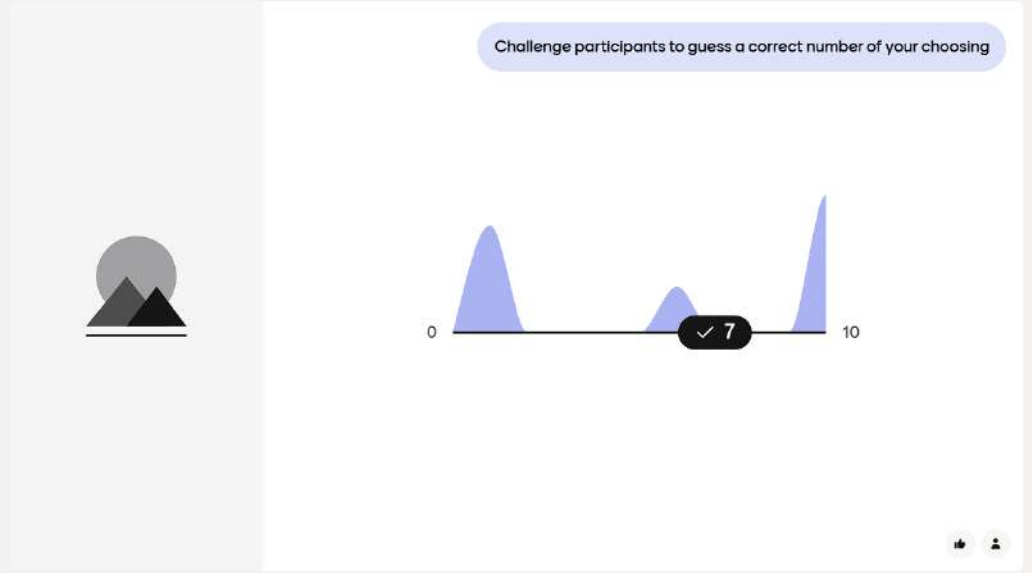

Рисунок 1.23

- 100 балів (рис. 1.2.2-14).

Учасники повинні розподілити 100 балів серед наданих варіантів, після чого програма видасть середній результат

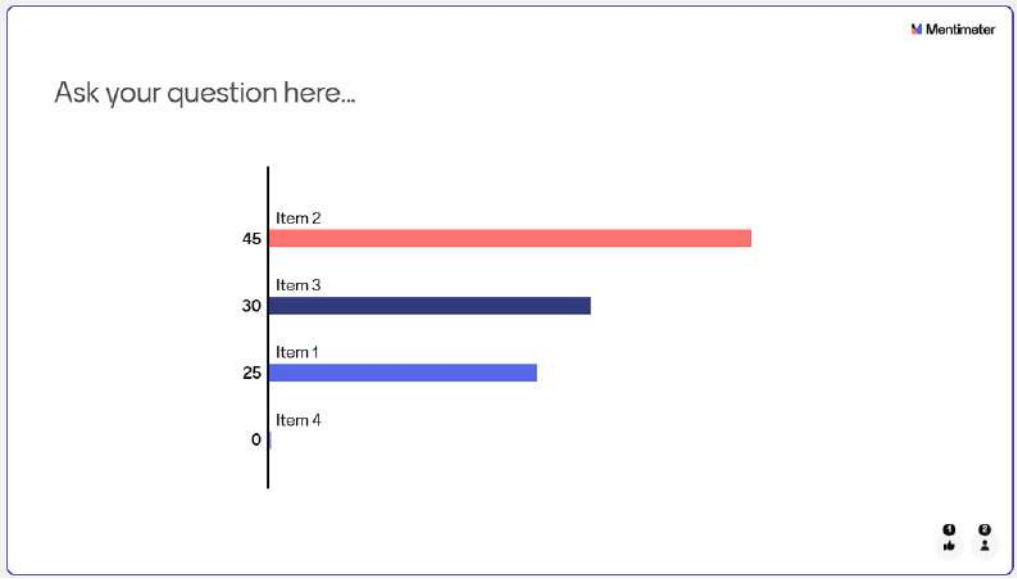

Рисунок 1.24

- Графік 2x2 (рис. 1.2.2-15).

Дозволяє графічно відобразити думку учасників стосовно впливу предмету питання на певні фактори.

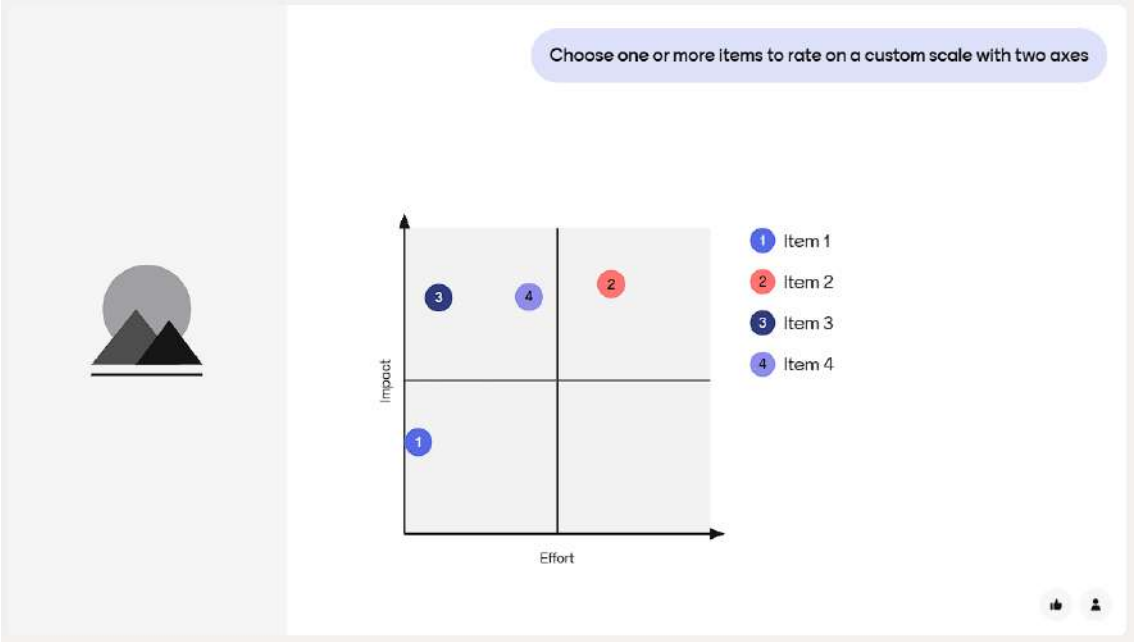

Рисунок 1.25

- Вибір на зображенні/відповідь-мітка (рис. 1.2.2-16).

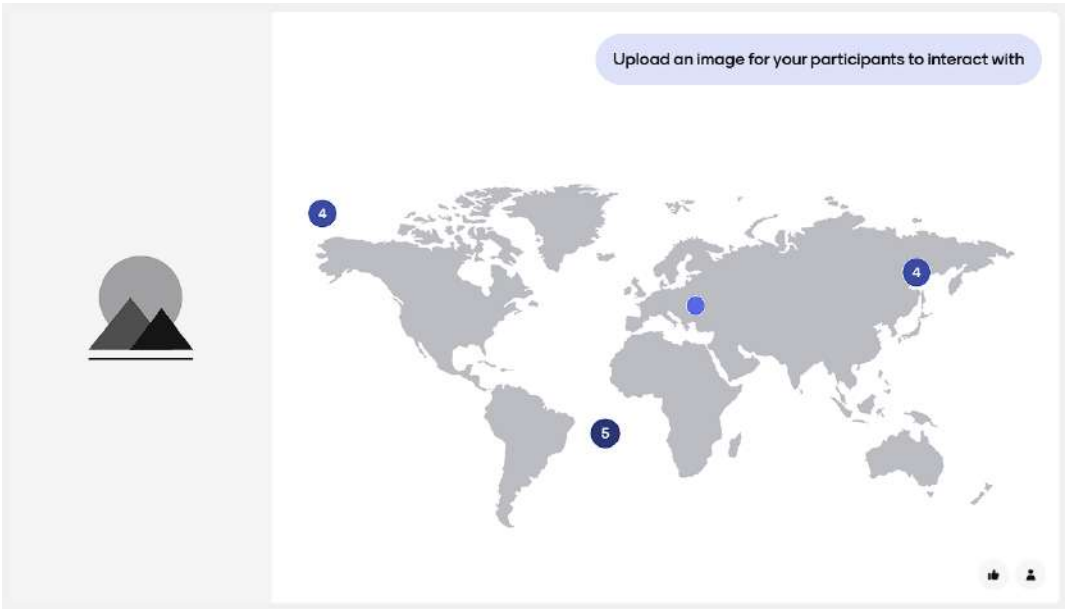

Рисунок 1.26

- Письмова відповідь (рис. 1.2.2-17).

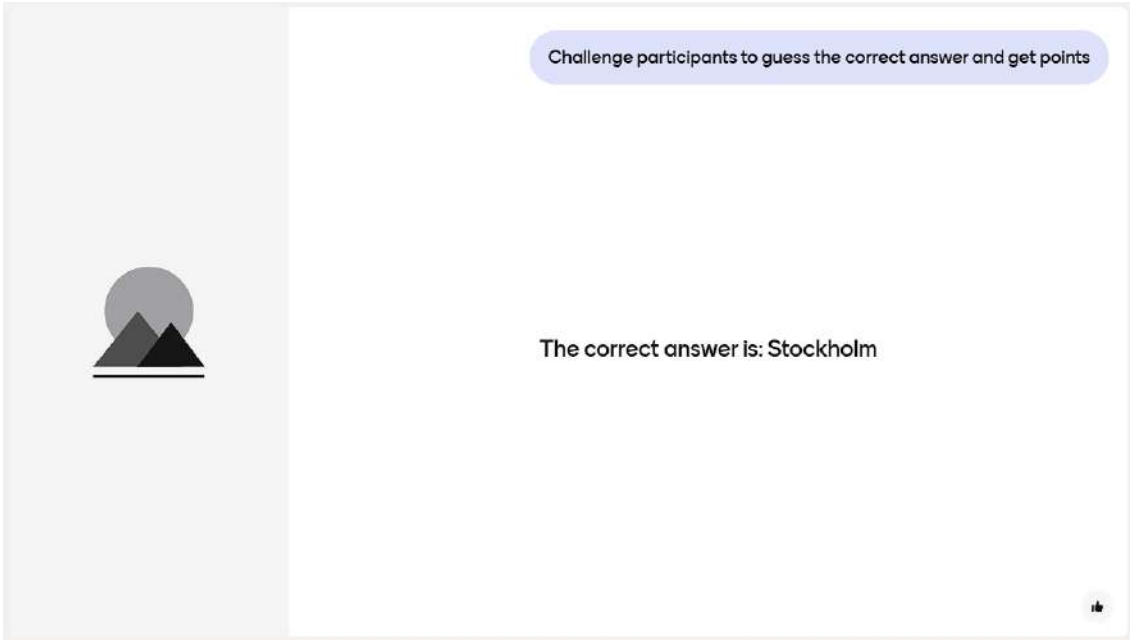

Рисунок 1.27

Застосунок має багатий функціонал в опитуванні аудиторії. Він має невеликий функціонал порівняно з іншими програмами з точки зору тестування учасників, проте має широкий спектр інструментів для роботи з групою.

1.2.3 Quizlet

Quizlet — це онлайн-платформа, розроблена для спрощення процесу запам’ятовування та навчання. Дизайн був натхнений флеш-картками, які з однієї сторони мають термін або зображення, а з іншої визначення. Quizlet зміг реалізувати цю ідею для онлайн навчання, і, судячи з популярності платформи, доволі успішно. Окрім веб версії також має мобільний застосунок.

Всі його методи тестування так чи інакше пов’язані з основним принципом термін-визначення(рис. 1.2.3-1). Як додаткова функція – можливість генерування цих карток з конспектів або книжок автоматично.

Платформа має свою унікальну ідею, але з точки розгляду як аналог для майбутнього застосунку з тестування знань мови JavaScript у учнів не є конкурентом, через фокусування на одному типі тестування, яке в нашому випадку краще замінити на завдання типу «встановлення відповідності».

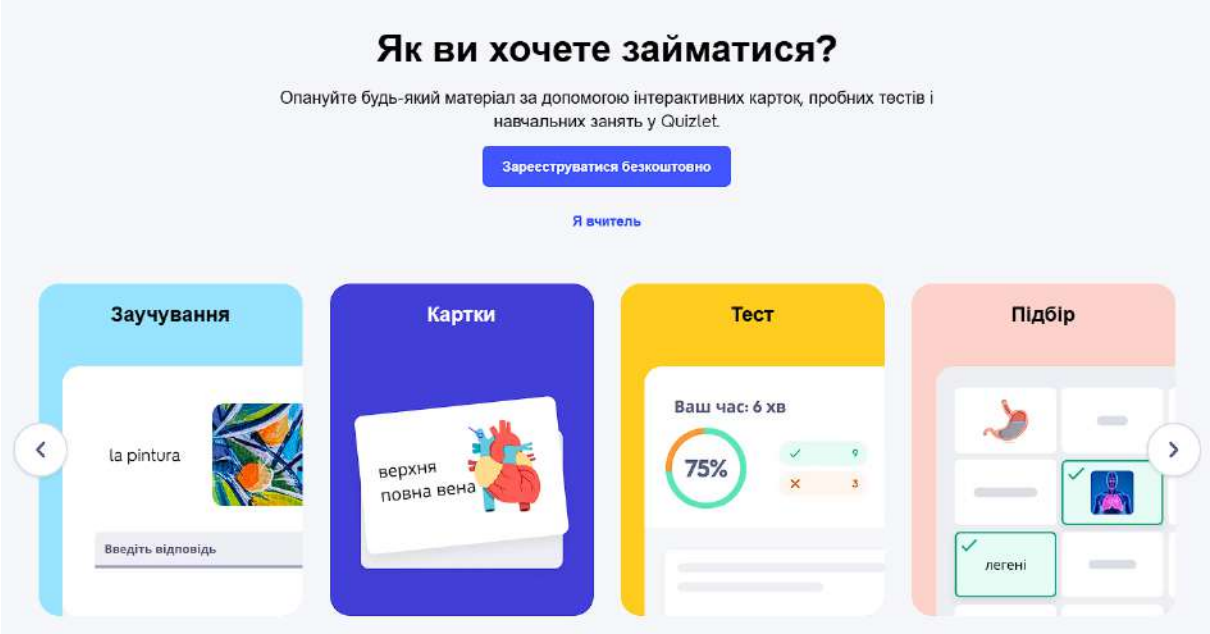

Рисунок 1.28

1.2.4 Nearpod

Nearpod — це онлайн-платформа для створення навчальних презентацій. Окрім стандартного функціоналу на кшталт можливості додавання відео, аудіо, тексту та можливості тестування, він надає можливість проводити VR тури на певних локаціях. Nearpod орієнтований на освітнє середовище, надаючи інструменти для управління навчальним процесом.

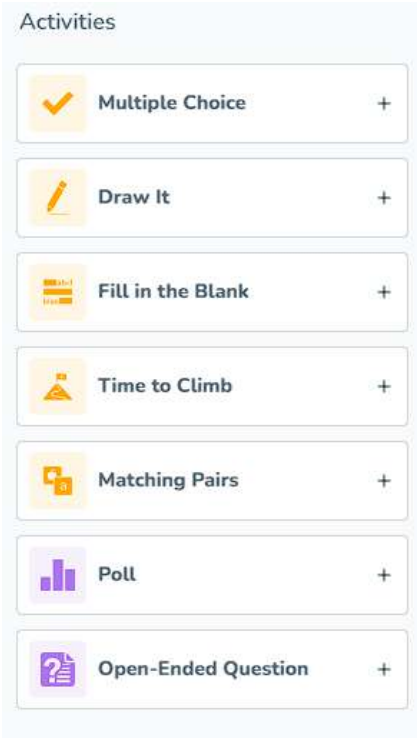

Рисунок 1.29

Серед методів тестування/активностей наявні(рис. 1.2.4-1):

- Вибір коректної відповіді(рис. 1.2.4-2).

Рисунок 1.30

- Намалюй(рис. 1.2.4-3).

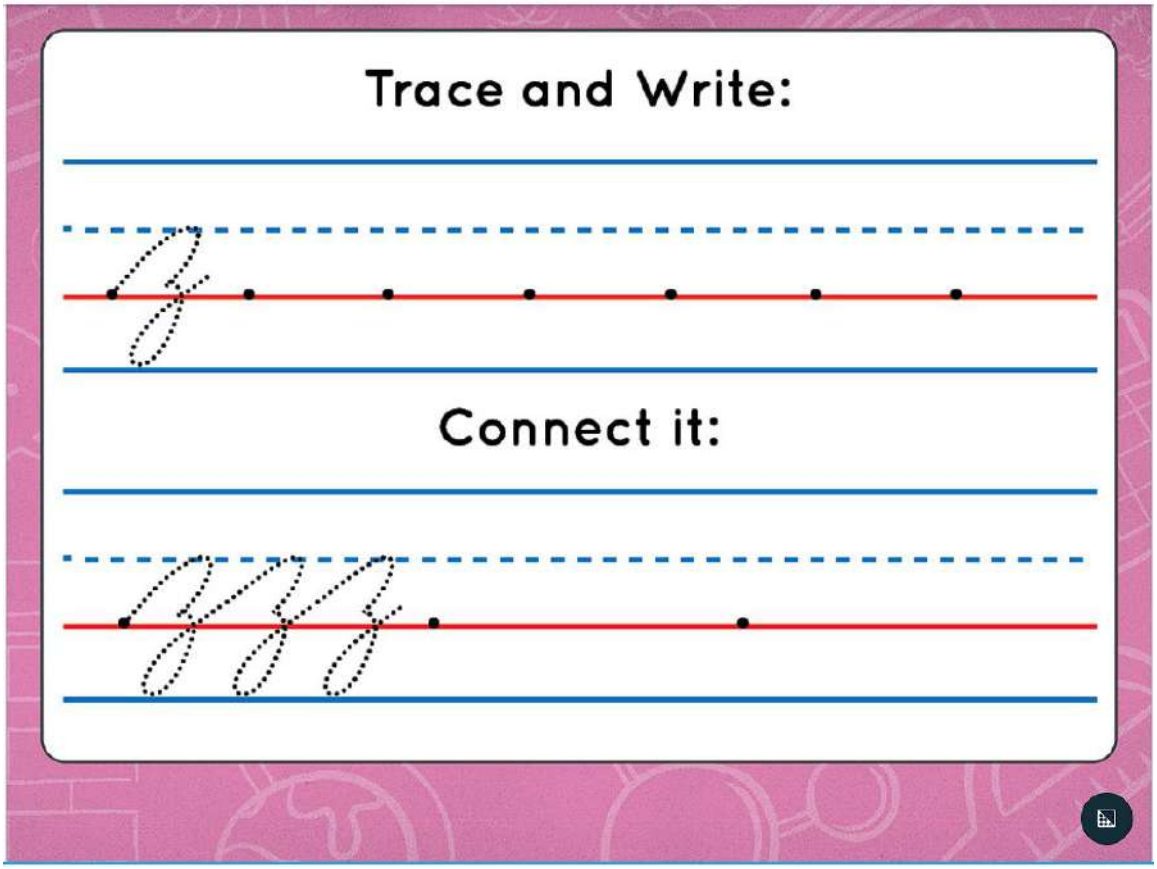

Рисунок 1.31

- Заповни пробіли(рис. 1.2.4-4).

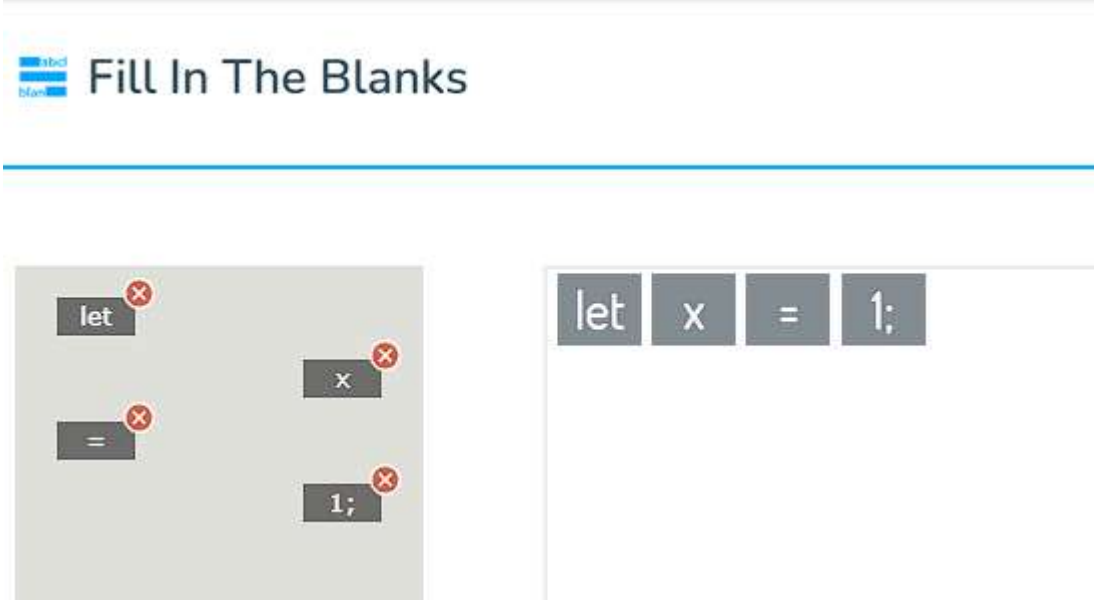

Рисунок 1.32

- «Час підніматися» (рис. 1.2.4-5).

В застосунку це представлення методу «Вибір корректної відповіді» але з таймером та зміненим дизайном.

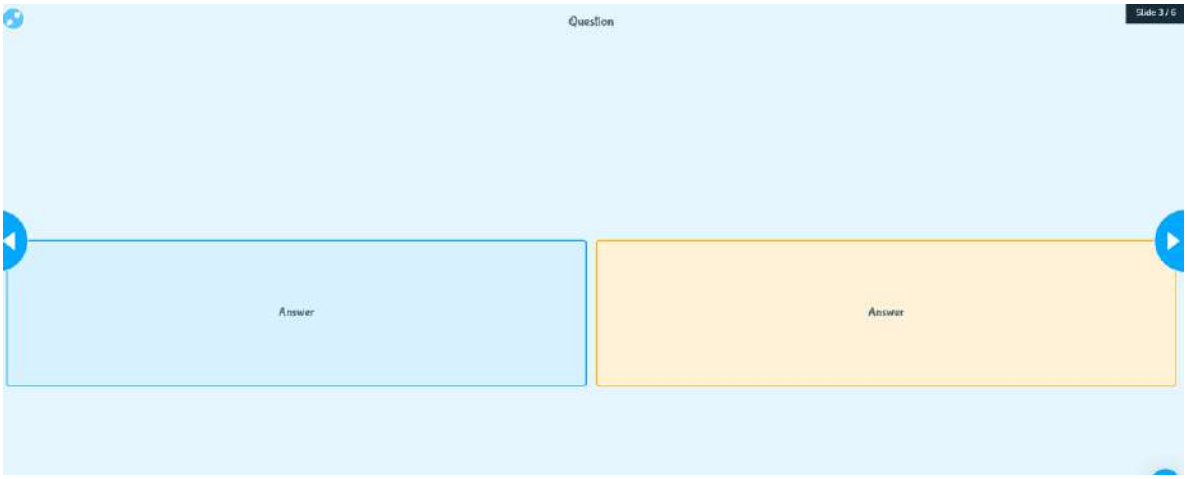

Рисунок 1.33

- Встановлення відповідності(рис. 1.2.4-6).

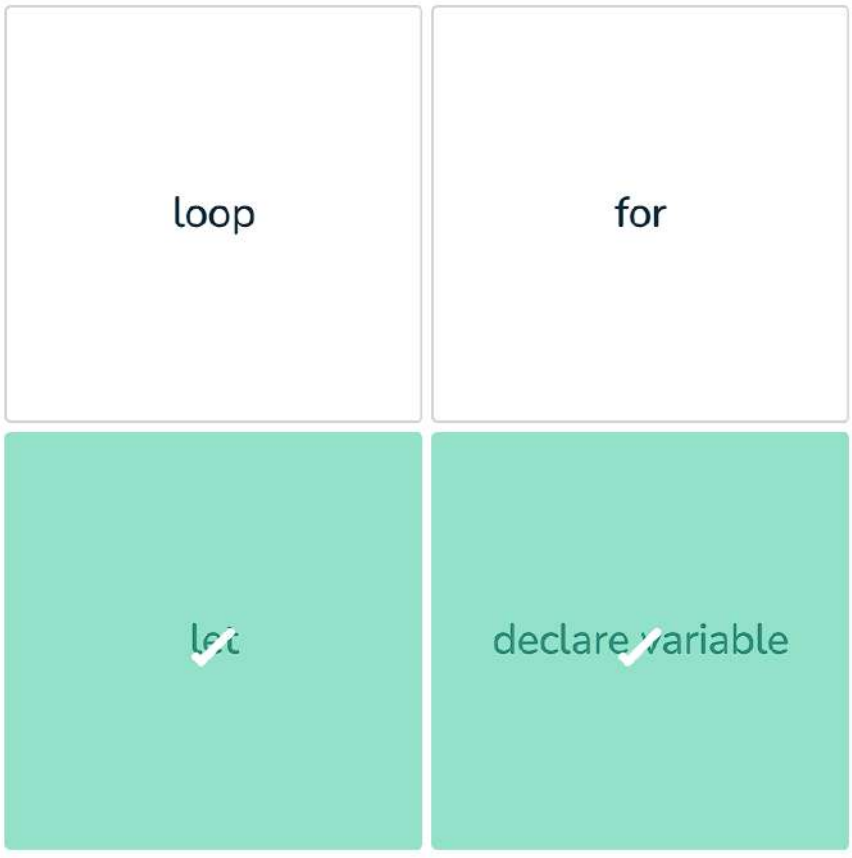

Рисунок 1.34

- Опитування(рис. 1.2.4-7).

Має аналогічний дизайн до «Вибору конкретної відповіді» але без можливості обрати правильну відповідь.

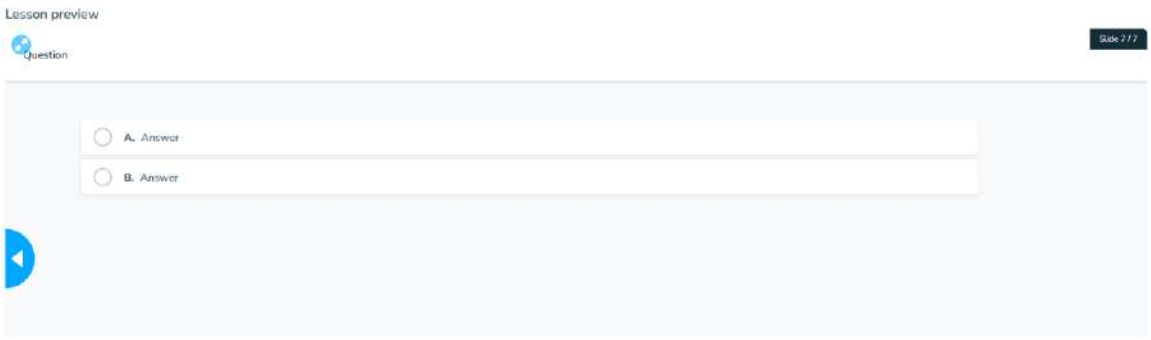

Рисунок 1.35

- Відкрите питання(рис. 1.2.4-8).

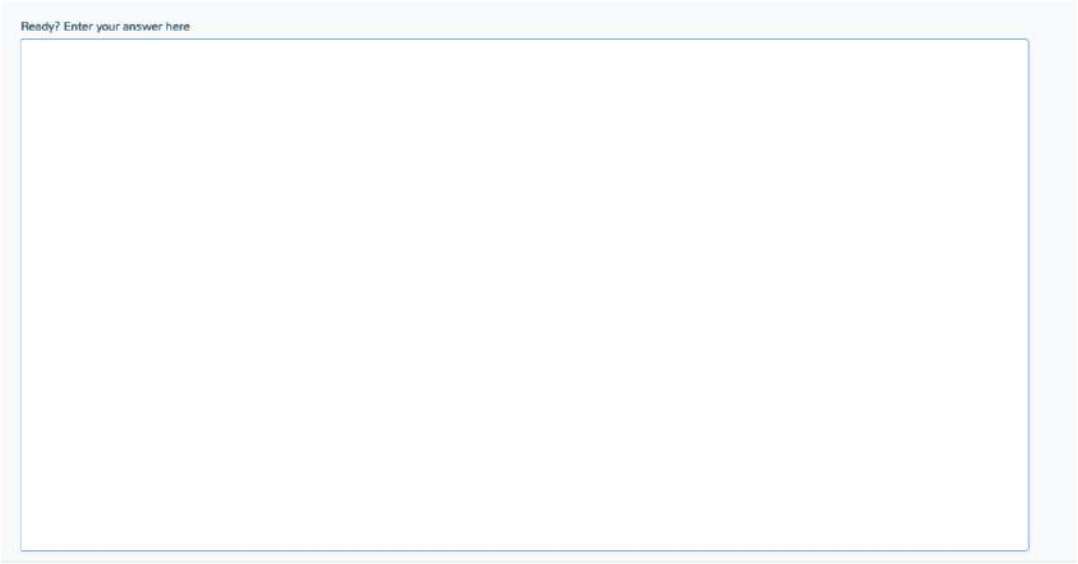

Рисунок 1.36

Програма має широкий спектр послуг. Серед наявних методів тестування хотілося б виділити «Заповнення пробілів». Цей метод тестування з певними покращеннями може бути застосований для тестування учнів нашою програмою. Дизайн доволі мінімалістичний.

1.2.5 Socrative

Socrative — це платформа для швидкого тестування знань студентів. Має можливіть генерації тесту по текстовому опису за допомогою штучного інтелекту. Інтерфейс гарно організований, відразу після початку тесту викладачу

демонструється список учасників та їх відповіді(рис. 1.2.5-1), які відразу ж можна відправити поштою іншим учасникам

Рисунок 1.37

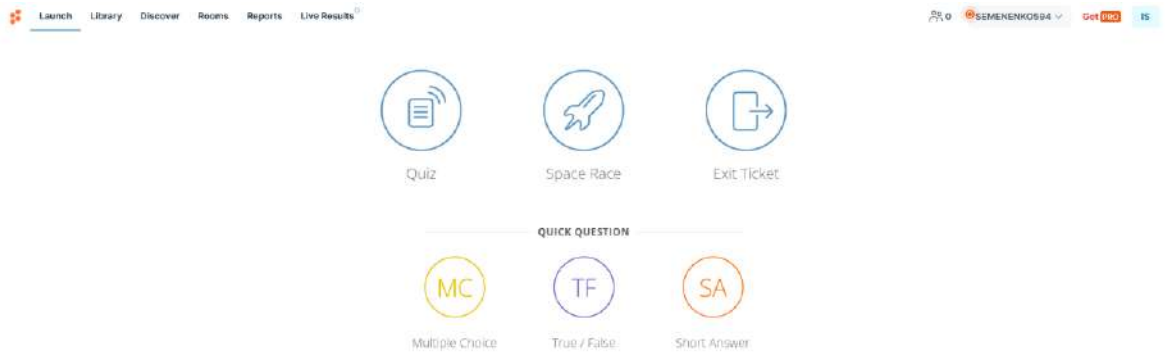

Рисунок 1.38

Серед можливих типів тестування наявні(рис. 1.2.5-2):

- Вибір коректної відповіді(рис. 1.2.5-3).

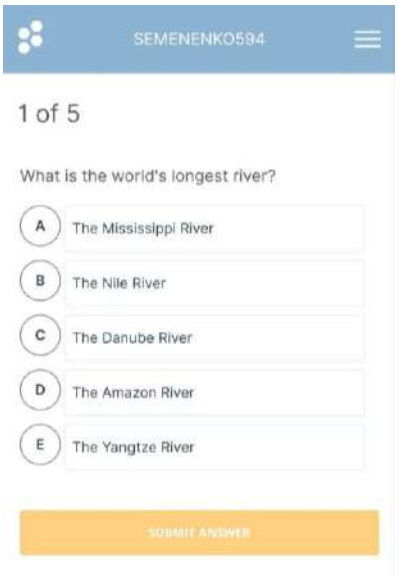

Рисунок 1.39

- Істина/Хиба(рис. 1.2.5-4).

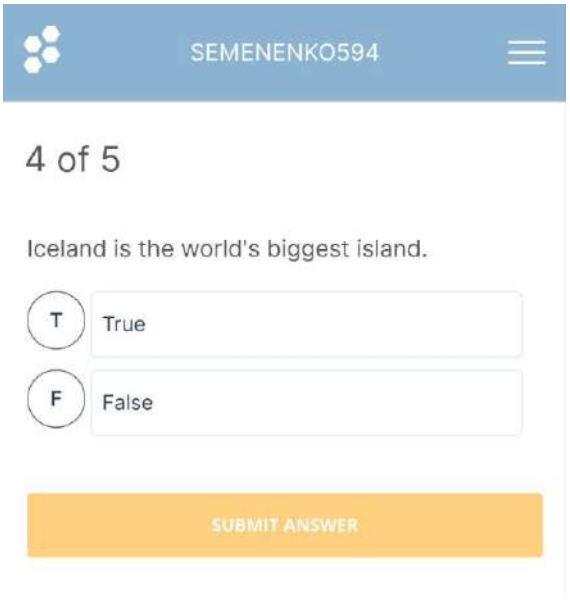

Рисунок 1.40

- Відкрите питання(рис. 1.2.5-5).

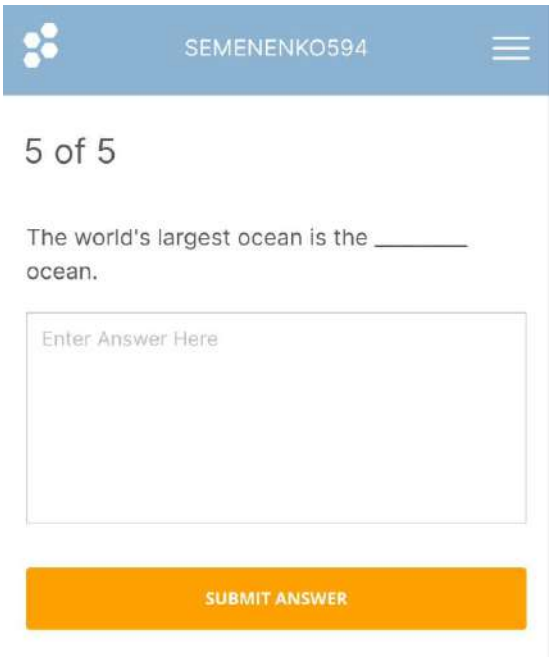

Рисунок 1.41

Програма виділяється своєю простотою на фоні інших з точки зору кількості кроків для досягнення бажаного результату та включає лише стандартні методи тестування.

1.3 Висновки до розділу 1

У цьому розділі було проведено комплексний аналіз поточних методик викладання програмування для школярів та детально вивчено функціонал популярних інтерактивних освітніх платформ. Результати аналізу дозволили виявити ключові переваги та недоліки існуючих рішень, що стало основою для формування вимог до власної розробки та вибору ефективних підходів до навчання та технічної реалізації. Було обрано 4 найбільш оптимальні типи тестування для майбутнього онлайн курсу: вибір коректної відповіді, істина/хиба, відкрите питання та вибір декількох варіантів відповідей.

РОЗДІЛ 2. ПРОЄКТУВАННЯ АРХІТЕКТУРИ ТА МЕТОДОЛОГІЧНИХ ОСНОВ ЕЛЕКТРОННОГО КУРСУ "ВСТУП ДО ПРОГРАМУВАННЯ"

2.1 Методологічні засади створення онлайн-курсу з програмування

Педагогічно-методичні основи для подачі навчального матеріалу з програмування українською мовою для школярів, структура онлайн-курсу (включаючи послідовність модулів та уроків), а також детальне обґрунтування вибору JavaScript як першої мови програмування для цільової аудиторії, ґрунтуються на підходах, викладених у навчальному посібнику "Вступ до дитячого програмування. Мова JavaScript" М. Глибовця [1]. Дана магістерська робота зосереджена на розробці програмного забезпечення електронного курсу, що реалізує ці методичні засади, тому детальне повторення зазначених педагогічних аспектів у цій роботі не передбачається, оскільки вони вичерпно висвітлені у вказаному джерелі.

2.1.1 Вимоги до програмного забезпечення електронного курсу

На основі мети та завдань дослідження, а також з урахуванням специфіки цільової аудиторії, до розроблюваного програмного забезпечення електронного курсу "Вступ до програмування" висуваються наступні функціональні та нефункціональні вимоги.

2.1.2 Функціональні вимоги

Функціональні вимоги визначають основні можливості та дії, які система повинна надавати користувачам [4]. Для електронного курсу "Вступ до програмування" ключовими функціональними вимогами є:

- Реєстрація та автентифікація користувачів:
  - Можливість реєстрації нових користувачів (учнів та викладачів/адміністраторів).
  - Безпечна автентифікація зареєстрованих користувачів для доступу до системи.
  - Розмежування прав доступу на основі ролей користувачів (учень, викладач, адміністратор).
- Доступ до навчальних матеріалів:
  - Надання учням доступу до структурованих навчальних модулів та уроків курсу.
  - Відображення теоретичного матеріалу, прикладів коду та інтерактивних елементів.
- Проходження тестів та завдань:
  - Можливість для учнів проходити інтерактивні тестові завдання різних типів.
  - Автоматична перевірка відповідей.
- Адміністрування навчального контенту (для викладачів/адміністраторів):
  - Можливість додавання, редагування та видалення кастомних тестових питань.

2.1.3 Нефункціональні вимоги

Нефункціональні вимоги визначають атрибути якості системи та обмеження, що накладаються на її розробку та функціонування [4].

- Зручність використання (Юзабіліті):

- Інтерфейс системи має бути інтуїтивно зрозумілим та легким у засвоєнні для школярів.
- Навігація по курсу та його елементах повинна бути простою та логічною.
- Забезпечення позитивного користувацького досвіду.

- Адаптивність дизайну: коректне відображення та функціональність інтерфейсу на різних типах пристроїв (персональні комп'ютери, ноутбуки, планшети, смартфони) та у популярних веб-браузерах.
- Продуктивність:
  - Система повинна забезпечувати швидке завантаження сторінок та навчальних матеріалів.
  - Час відгуку на дії користувача має бути мінімальним для забезпечення комфортної взаємодії.
- Безпека:
  - Захист персональних даних користувачів відповідно до чинних стандартів.
  - Забезпечення безпеки автентифікації та авторизації.
  - Захист системи від поширених веб-вразливостей.
- Масштабованість: Архітектура системи повинна дозволяти ефективне нарощування ресурсів для обслуговування зростаючої кількості користувачів та обсягу навчального контенту без суттєвого погіршення продуктивності.
- Надійність:
  - Стабільна та безперебійна робота системи.
  - Мінімізація ймовірності виникнення помилок та збоїв.
- Підтримуваність: код та архітектура системи повинні бути добре структурованими, документованими та легкими для розуміння, що спростить внесення змін, виправлення помилок та подальший розвиток функціоналу.
-

2.2 Архітектурні підходи, патерни проєктування та обґрунтування вибору технологій для реалізації проєкту

Вибір правильної архітектури та технологічного стеку є критично важливим для успішної розробки, підтримки та масштабування будь-якого програмного продукту, особливо такого, як інтерактивний електронний курс. У цьому підрозділі детально обґрунтовуються ключові архітектурні рішення, обрані патерни проєктування та технології, що ляжуть в основу електронного курсу "Вступ до програмування".

2.2.1 Обґрунтування використання клієнт-серверної моделі та принципів REST API.

Для розробки електронного курсу "Вступ до програмування" було обрано клієнт-серверну архітектуру взаємодії. Ця модель передбачає розподіл системи на дві основні логічні частини: клієнтську частину (Frontend), яка відповідає за представлення інформації користувачеві та взаємодію з ним, та серверну частину (Backend), що забезпечує зберігання даних, обробку бізнес-логіки та надання даних клієнту.

Клієнт-серверна модель була обрана через такі особливості:

- Розподіл відповідальності (Separation of Concerns): Чітке розмежування функцій між клієнтом і сервером спрощує розробку, тестування та підтримку кожної частини окремо. Команди розробників можуть працювати паралельно над фронтендом та бекендом.
- Централізоване зберігання та управління даними: Усі навчальні матеріали, дані про користувачів, тестові завдання зберігаються централізовано на сервері. Це забезпечує цілісність, консистентність та безпеку даних.
- Масштабованість: Серверну частину можна масштабувати незалежно від клієнтської для обробки зростаючої кількості користувачів або

обсягу даних. Наприклад, можна збільшити потужність сервера або додати нові сервери без необхідності змінювати клієнтські застосунки.

- Доступність з різних пристроїв: Клієнтська частина, реалізована як веб-застосунок, буде доступна користувачам через стандартні веб-браузери з будь-яких пристроїв (ПК, ноутбуки, планшети, смартфони), що мають доступ до Інтернету.
- Безпека: Критично важлива бізнес-логіка та доступ до бази даних інкапсулюються на сервері, що зменшує ризики несанкціонованого доступу або маніпуляцій з боку клієнта.

Для взаємодії між клієнтською та серверною частинами електронного курсу буде використовуватися архітектурний стиль REST (Representational State Transfer) API (Application Programming Interface). REST — це набір принципів та обмежень для створення веб-сервісів, які забезпечують просту, стандартизовану та ефективну комунікацію [5].

Ключові принципи REST API, що будуть застосовані [5]:

- Клієнт-серверна архітектура: Як вже було зазначено, цей принцип є основоположним.
- Відсутність стану (Statelessness): Кожен запит від клієнта до сервера повинен містити всю необхідну інформацію для його обробки. Сервер не зберігає жодного стану клієнта між запитами. Це спрощує масштабування та підвищує надійність системи, оскільки будь-який екземпляр сервера може обробити запит.
- Кешування (Cacheability): Відповіді сервера можуть бути позначені як кешовані. Це дозволяє клієнтам або проміжним вузлам (наприклад, CDN) зберігати копії відповідей для зменшення навантаження на сервер та прискорення отримання даних.
- Єдиний інтерфейс (Uniform Interface): Цей принцип є ключовим для REST і включає:
    - Ідентифікація ресурсів через URI (Uniform Resource Identifier): Кожен ресурс (наприклад, урок, тест, користувач) має унікальний ідентифікатор URI (наприклад, /api/lessons/1).

- Маніпуляція ресурсами через представлення (Representations): Клієнт взаємодіє з ресурсами через їхні представлення (зазвичай у форматі JSON або XML). У нашому випадку буде використовуватися JSON.
- Самодостатні повідомлення (Self-descriptive messages): Кожен запит та відповідь містить достатньо інформації для їх інтерпретації (наприклад, використання HTTP-заголовків для вказання типу контенту Content-Type: application/json).

- Багаторівнева система (Layered System): Дозволяє використовувати проміжні сервери (наприклад, для балансування навантаження, кешування, безпеки), при цьому клієнт не знає про їх існування і взаємодіє лише з кінцевим URI.

Обґрунтування вибору REST API:

- Простота та стандартизація: REST використовує стандартні HTTP методи (GET для отримання даних, POST для створення, PUT для оновлення, DELETE для видалення), що робить API інтуїтивно зрозумілим та легким для інтеграції.
- Незалежність клієнта та сервера: Чіткий контракт API дозволяє незалежно розробляти та еволюціонувати фронтенд (наприклад, на Next.js/React) та бекенд (на ASP.NET Core). Це також відкриває можливість у майбутньому створювати інші клієнти (наприклад, мобільний застосунок) для того ж самого API.
- Широка підтримка та інструментарій: Існує велика кількість бібліотек та інструментів для розробки, тестування та документування REST API на різних платформах та мовах програмування.
- Ефективність передачі даних: Використання легко_вагового формату JSON для передачі даних зменшує обсяг трафіку та прискорює обмін інформацією між клієнтом та сервером.
- Масштабованість: Принцип відсутності стану (statelessness) спрощує горизонтальне масштабування серверної частини.

Поєднання клієнт-серверної архітектури з принципами REST API забезпечує гнучку, масштабовану та підтримувану основу для розробки електронного курсу "Вступ до програмування". Клієнтська частина на Next.js/React буде відповідати за зручний та інтерактивний користувацький інтерфейс, тоді як серверна частина на ASP.NET Core Web API буде надійно обробляти запити, керувати даними та реалізовувати основну бізнес-логіку курсу.

2.2.2 N-Tier Architecture (Багаторівнева архітектура)

Для забезпечення гнучкості, підтримуваності та масштабованості серверної частини (Backend) електронного курсу "Вступ до програмування" буде застосована N-Tier Architecture (багаторівнева архітектура) [7]. Цей архітектурний підхід передбачає логічний поділ застосунку на кілька незалежних рівнів (шарів), кожен з яких відповідає за специфічний набір функцій та обов'язків. Такий поділ дозволяє зменшити зв'язність між компонентами системи та підвищити її загальну якість.

У контексті нашого проєкту, серверна частина, що реалізується за допомогою ASP.NET Core Web API, буде структурована за наступними основними рівнями:

- Presentation Layer (Рівень представлення / Контролери):
  - Призначення: Цей рівень є точкою входу для всіх запитів від клієнтської частини (Frontend). Його основне завдання – приймати HTTP-запити, розбирати їх, валідувати вхідні дані (зазвичай Data Transfer Objects - DTOs), викликати відповідні методи бізнес-логіки для обробки запиту та формувати HTTP-відповідь (найчастіше у форматі JSON) для клієнта.
  - Компоненти: В ASP.NET Core цей рівень представлений контролерами (Controllers). Кожен контролер відповідає за обробку запитів, пов'язаних з певним ресурсом або

функціональним блоком системи (наприклад, LessonsController, UsersController, TestsController).

- Взаємодія: Контролери отримують дані від клієнта, передають їх на обробку сервісам з рівня бізнес-логіки та повертають результат клієнту. Вони не містять складної бізнес-логіки або логіки доступу до даних.

- Business Logic Layer (BLL) / Application Layer (Рівень бізнес-логіки / Сервіси):
  - Призначення: Це серце застосунку, де реалізується вся основна бізнес-логіка та правила. Саме тут відбувається обробка даних, виконання розрахунків, прийняття рішень та координація взаємодії між іншими рівнями. Наприклад, логіка перевірки відповідей на тестові завдання, розрахунок прогресу учня, управління навчальним контентом, забезпечення виконання бізнес-правил курсу.
  - Компоненти: Цей рівень складається з сервісів (Services). Кожен сервіс інкапсулює певну частину бізнес-логіки (наприклад, LessonService, UserService, TestManagementService).
  - Взаємодія: Сервіси отримують дані від контролерів, виконують необхідні операції, використовуючи для доступу до даних методи з рівня доступу до даних (Repositories), та повертають результат контролерам. Цей шар не залежить від конкретного способу представлення даних (наприклад, веб-інтерфейсу) і може бути перевикористаний.

- Data Access Layer (DAL) (Рівень доступу до даних / Репозиторії):
  - Призначення: Цей рівень відповідає за всю взаємодію з джерелом даних (базою даних). Він абстрагує логіку роботи з базою даних від решти застосунку, надаючи бізнес-логіці простий та зрозумілий інтерфейс для виконання операцій CRUD (Create, Read, Update, Delete) над сутностями.

- Компоненти: Основними компонентами цього рівня є репозиторії (Repositories), які реалізують однойменний патерн проєктування (Repository Pattern). Кожен репозиторій зазвичай працює з одним типом сутності (наприклад, LessonRepository, UserRepository).
- Взаємодія: Репозиторії отримують запити від сервісів рівня бізнес-логіки та трансформують їх у конкретні запити до бази даних (наприклад, SQL-запити або LINQ-запити при використанні Entity Framework Core). Вони також відповідають за перетворення даних з формату бази даних у доменні об'єкти, зрозумілі для рівня бізнес-логіки.

- Database Layer (Рівень бази даних):
  - Призначення: Це фізичний рівень зберігання даних. У нашому випадку це буде реляційна база даних Microsoft SQL Server 2022.
  - Компоненти: Включає саму СУБД, схему бази даних (таблиці, зв'язки, індекси тощо). Для взаємодії з цим рівнем з коду .NET буде використовуватися Entity Framework Core (EF Core) 9.0.5 – сучасний Object-Relational Mapper (ORM). EF Core дозволяє розробникам працювати з базою даних, використовуючи об'єкти .NET, абстрагуючись від написання SQL-запитів вручну (хоча така можливість залишається). DbContext та класи-сутності (Entities) є ключовими елементами EF Core на цьому рівні, що використовуються репозиторіями.

Переваги багаторівневої архітектури для проєкту:

- Покращена підтримуваність (Maintainability):
  - Чіткий поділ відповідальності: Кожен рівень має свою чітко визначену роль. Зміни в одному рівні (наприклад, оновлення логіки в сервісі) мають мінімальний вплив на інші рівні, за умови дотримання контрактів (інтерфейсів) між ними.

- Спрощене розуміння коду: Кодова база стає більш організованою та легкою для розуміння, оскільки логіка згрупована за функціональними областями.
- Полегшене командне розроблення: Різні розробники або команди можуть паралельно працювати над різними рівнями системи.

- Висока масштабованість (Scalability):
  - Незалежне масштабування рівнів: Хоча на початковому етапі система може бути розгорнута як єдиний процес (моноліт з логічними рівнями), така архітектура закладає основу для фізичного розподілу рівнів на окремі сервери у майбутньому. Наприклад, рівень бізнес-логіки або рівень доступу до даних можуть бути винесені на окремі сервери та масштабуватися незалежно, якщо вони стають вузьким місцем.
  - Оптимізація ресурсів: Можливість оптимізувати ресурси для кожного рівня окремо.
- Підвищена гнучкість та можливість повторного використання (Flexibility and Reusability):
  - Заміна компонентів: Легше замінити реалізацію одного рівня, не зачіпаючи інші. Наприклад, можна змінити СУБД, адаптувавши лише рівень доступу до даних, або додати новий тип клієнтського застосунку (наприклад, мобільний додаток), який буде використовувати той самий рівень бізнес-логіки.
  - Повторне використання сервісів: Сервіси рівня бізнес-логіки можуть бути використані різними типами клієнтів або в інших частинах системи.
- Покращена тестувальність (Testability):
  - Ізольоване тестування: Кожен рівень можна тестувати окремо. Наприклад, можна провести unit-тестування сервісів, імітуючи залежності від репозиторіїв, або тестувати репозиторії, використовуючи тестову базу даних.

Застосування N-Tier архітектури є обґрунтованим рішенням для створення надійної, гнучкої та готової до розвитку освітньої платформи. Цей підхід сприятиме створенню якісного програмного продукту, який буде легко підтримувати та розвивати в довгостроковій перспективі.

2.2.3 Об'єктно-орієнтоване програмування (ООП)

В основі розробки програмного забезпечення електронного курсу лежить парадигма об'єктно-орієнтованого програмування (ООП). Цей підхід дозволяє моделювати предметну область у вигляді взаємодіючих об'єктів, кожен з яких є екземпляром певного класу та має свої властивості (дані) та поведінку (методи) [7]. Застосування ООП сприяє створенню модульної, гнучкої та легко підтримуваної системи.

Фундаментальні принципи ООП, які будуть використані при моделюванні сутностей курсу (таких як користувачі, уроки, питання тощо), включають [7]:

- Абстракція (Abstraction): виділення ключових характеристик об'єкта та ігнорування несуттєвих деталей. Для сутності Урок (Lesson), наприклад, важливими будуть властивості назва, зміст, порядок_відображення та методи відобразити_зміст(), позначити_як_пройдений(), тоді як внутрішні деталі зберігання контенту можуть бути приховані. Це дозволяє зосередитись на "що" об'єкт робить, а не "як" він це робить.
- Інкапсуляція (Encapsulation): об'єднання даних (атрибутів) та методів, що ними маніпулюють, в єдину структуру – клас, а також приховування внутрішньої реалізації об'єкта від зовнішнього світу. Доступ до даних об'єкта контролюється через його публічний інтерфейс (методи). Наприклад, об'єкт Користувач (User) інкапсулюватиме такі дані, як логін, хеш_паролю, роль, та надаватиме методи типу змінити_пароль() або отримати_роль(), при цьому прямий доступ до хеш_паролю ззовні буде обмежений. Це підвищує безпеку та цілісність даних.

- Наслідування (Inheritance): механізм, що дозволяє створювати нові класи (класи-нащадки) на основі існуючих (базових класів), успадковуючи їхні властивості та методи. Це сприяє повторному використанню коду та побудові ієрархій класів.
- Поліморфізм (Polymorphism): здатність об'єктів різних класів реагувати на однаковий виклик методу по-різному. Буквально "багато форм". Наприклад, якщо у нас є масив об'єктів різних типів питань (успадкованих від базового Питання), ми можемо викликати для кожного з них метод відобразити_питання(). Кожен об'єкт виконає цей метод відповідно до своєї власної реалізації: питання з однією відповіддю відобразиться з радіокнопками, а питання з множинним вибором – з чекбоксами. Це дозволяє писати більш гнучкий та узагальнений код.

Застосування цих принципів ООП при розробці класів для моделювання сутностей курсу, таких як Користувач (User), Урок (Lesson), Питання (Question), Відповідь (Answer) забезпечить створення добре структурованої, модульної та розширюваної системи.

2.2.4 Принципи об'єктно-орієнтованого дизайну (SOLID) та патерни проектування

Для створення якісної архітектури серверної частини електронного курсу, яка буде гнучкою, розширюваною та легкою у підтримці, будуть застосовані ключові принципи об'єктно-орієнтованого дизайну SOLID та ряд відомих патернів проєктування [8].

Принципи SOLID [8]:

- S – Single Responsibility Principle (Принцип єдиної відповідальності): Кожен клас матиме одну, чітко визначену зону відповідальності. Наприклад, UserService відповідатиме за бізнес-логіку користувачів, а LessonRepository – за доступ до даних уроків.

- – Open/Closed Principle (Принцип відкритості/закритості): Система буде відкрита для розширення (додавання нового функціоналу через нові класи/модулі, наприклад, нових типів питань через реалізацію IQuestionTypeStrategy), але закрита для модифікації існуючого, стабільного коду.
- L – Liskov Substitution Principle (Принцип підстановки Барбари Лісков): Похідні класи будуть повністю сумісні з базовими класами, гарантуючи, що заміна об'єкта базового типу на об'єкт похідного типу не порушить логіку програми (наприклад, Student та Teacher як підтипи BaseUser).
- I – Interface Segregation Principle (Принцип розділення інтерфейсу): Замість великих, універсальних інтерфейсів будуть використовуватися менші, більш спеціалізовані (наприклад, ILessonService, ITestService), щоб класи реалізовували лише ті методи, які їм дійсно потрібні.
- D – Dependency Inversion Principle (Принцип інверсії залежностей): Модулі вищого рівня (наприклад, сервіси) залежатимуть від абстракцій (інтерфейсів, як ILessonRepository), а не від конкретних реалізацій модулів нижчого рівня (SqlLessonRepository). Це досягатиметься через Dependency Injection.

Патерни проєктування:

- Service Layer Pattern (Шар сервісів) [9]: Інкапсулює бізнес-логіку в окремому шарі (LessonService, TestAttemptService), що слугує фасадом для контролерів.
- Repository Pattern (Репозиторій) [6]: Абстрагує доступ до даних (IUserRepository, IAnswerRepository), надаючи інтерфейс для CRUD-операцій та приховуючи деталі взаємодії з БД.
- Unit of Work Pattern (Одиниця роботи) [6]: Забезпечує атомарність операцій з базою даних. Реалізується через DbContext Entity Framework Core, де SaveChangesAsync() зберігає всі зміни в рамках однієї транзакції.
- Dependency Injection (DI) Pattern (Впровадження залежностей) [9]: Залежності (сервіси, репозиторії) надаються об'єктам ззовні (через конструктори), що сприяє слабкій зв'язності. Вбудований в ASP.NET Core.

- Controller Pattern (Контролер) [9]: Класи-контролери (AuthController, QuestionController) обробляють HTTP-запити, делегують обробку сервісам та повертають відповіді клієнту.

Дотримання цих принципів та патернів спрямоване на створення надійної, гнучкої та масштабованої архітектури, що є критично важливим для довгострокового розвитку та підтримки електронного курсу.

2.2.5 Патерни автентифікації та безпеки

Забезпечення безпеки взаємодії між клієнтською та серверною частинами є пріоритетом для освітньої платформи. Обрані архітектурні рішення в цій області спрямовані на реалізацію надійних механізмів автентифікації та авторизації.

Автентифікація на основі JSON Web Tokens (JWT)

JWT (RFC 7519) для автентифікації зумовлений його ефективністю в stateless архітектурах та придатністю для RESTful API. Після успішної перевірки облікових даних користувача, сервер генерує та підписує JWT, що містить необхідні твердження (claims) про суб'єкта (ідентифікатор, роль). Клієнтська частина надсилає цей токен у заголовку Authorization (як Bearer токен) кожного подальшого запиту до захищених ресурсів. Сервер валідує токен (підпис, термін дії), забезпечуючи ідентифікацію користувача та його прав без необхідності зберігання стану сесії. Критично важливим є використання HTTPS для передачі токенів та забезпечення їх цілісності за допомогою цифрового підпису.

Політики Cross-Origin Resource Sharing (CORS)

Взаємодія між відокремленими фронтендом (Next.js) та бекендом (ASP.NET Core), що потенційно функціонують на різних доменах/портах, вимагає коректного налаштування CORS. Це браузерний механізм безпеки, що дозволяє

серверу контролювати, які зовнішні джерела (origins) мають право доступу до його ресурсів, тим самим обходячи обмеження політики однакового походження (Same-Origin Policy). На серверній стороні буде реалізована сувора політика CORS, що чітко визначатиме дозволені джерела, HTTP-методи та заголовки, мінімізуючи ризики несанкціонованого доступу.

2.2.6 Обґрунтування вибору технологічного стеку

Вибір технологій для електронного курсу "Вступ до програмування" ґрунтувався на критеріях продуктивності, надійності, масштабованості та ефективності розробки.

Клієнтська частина (Frontend)

React (v19.0.0) та Next.js (v15.1.8): Обрано через компонентну архітектуру React для модульних UI та переваги Next.js, такі як SSR/SSG для оптимізації початкового завантаження та SEO, вбудовану маршрутизацію та розвинену екосистему.

TypeScript: Інтегровано для статичної типізації, що підвищує якість коду, полегшує його підтримку та зменшує кількість помилок на етапі виконання.

Tailwind CSS: Застосовано для швидкої та консистентної розробки адаптивних інтерфейсів за допомогою утилітарних класів.

Порівняння з Angular/Vue.js: Хоча Angular є потужним для великих корпоративних рішень, а Vue.js пропонує легкість інтеграції, поєднання React та Next.js надає оптимальний баланс гнучкості, широкої екосистеми та можливостей для ефективної реалізації освітньої платформи, враховуючи також існуючу експертизу.

Серверна частина (Backend)

.NET 8.0 (ASP.NET Core Web API): Вибір зумовлений високою продуктивністю .NET, строгістю типізації C# для розробки надійного коду, кросплатформеністю та екосистемою ASP.NET Core з вбудованими засобами для безпеки, DI та створення RESTful API.

Порівняння з Node.js (Express), Python (Django/Flask), PHP (Laravel): На відміну від Node.js, .NET часто забезпечує кращу продуктивність для CPU-інтенсивних задач. У порівнянні з Python-фреймворками та PHP/Laravel, строга типізація C# та архітектурні можливості ASP.NET Core визнані більш сприятливими для побудови складного та масштабованого бекенду, що вимагається для даного проєкту.

2.2.7 Архітектура фронтенд-частини: Feature-Sliced Design (FSD)

Для структурування кодової бази клієнтської частини (Frontend), реалізованої на Next.js та React, буде застосована архітектурна методологія Feature-Sliced Design (FSD). Цей підхід обрано задля забезпечення високого рівня підтримуваності, масштабованості та чіткої організації коду, що є особливо важливим для проєктів із потенціалом зростання та розширення функціональності, яким є освітня платформа.

Ключові аспекти та переваги FSD для проєкту:

- стандартизація та управління складністю
- ізоляція та перевикористання модулів
- масштабованість та підтримуваність

2.3 Проєктування моделі даних та ключових компонентів застосунку

Цей підрозділ присвячений детальному опису розробленої моделі даних, структури бази даних та архітектури основних компонентів серверної частини (Backend) електронного курсу "Вступ до програмування". Візуалізація цих аспектів здійснюється за допомогою відповідних діаграм (рис. 2.4.1-1,)*, що

ілюструють сутності, їх взаємозв'язки та структуру класів на різних рівнях архітектури.

2.3.1 Концептуальна модель даних та структура бази даних (ER-діаграма)

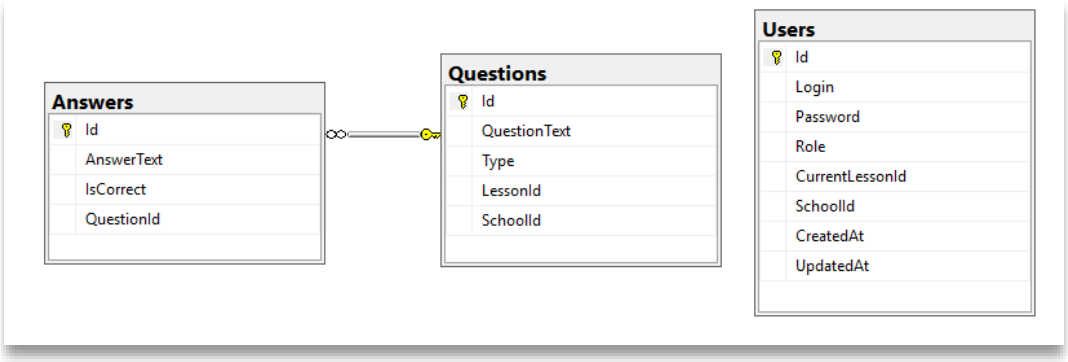

Рисунок 2.1

На Рисунку рис. 2.4.1-1 представлена концептуальна модель даних у вигляді ER-діаграми, що відображає ключові сутності та їх взаємозв'язки в базі даних проєкту.

Сутності бази даних:

- Users (Користувачі): Центральна сутність, що зберігає інформацію про зареєстрованих користувачів системи. Містить атрибути:
  - Id (Первинний ключ, ідентифікатор користувача).
  - Login (Логін користувача для входу в систему, передбачається унікальним).
  - Password (Хеш пароля користувача).
  - Role (Роль користувача в системі, наприклад, "студент", "викладач").
  - CurrentLessonId (Ідентифікатор поточного уроку, на якому зупинився користувач, може бути NULL).
  - SchoolId (Ідентифікатор навчального закладу, до якого належить користувач, може бути NULL).
  - CreatedAt (Дата та час створення запису про користувача).

- UpdatedAt (Дата та час останнього оновлення запису про користувача).

- Questions (Питання): Сутність для зберігання тестових питань. Містить атрибути:
  - Id (Первинний ключ, ідентифікатор питання).
  - QuestionText (Текст питання).
  - Type (Тип питання, наприклад, "одиночний вибір", "множинний вибір", "відкрита відповідь").
  - LessonId (Зовнішній ключ, що вказує на урок, до якого належить питання).
  - SchoolId (Ідентифікатор навчального закладу, для якого створено це питання, може вказувати на кастомні питання викладачів).
- Answers (Відповіді): Сутність для зберігання варіантів відповідей до питань. Містить атрибути:
  - Id (Первинний ключ, ідентифікатор відповіді).
  - AnswerText (Текст варіанту відповіді).
  - IsCorrect (Булевий прапорець, що вказує, чи є відповідь правильною).
  - QuestionId (Зовнішній ключ, що вказує на питання, до якого належить цей варіант відповіді).

Взаємозв'язки:

Між сутностями Questions та Answers існує зв'язок один-до-багатьох: одне питання (Questions) може мати багато варіантів відповідей (Answers), але кожен варіант відповіді належить лише одному питанню. Це реалізовано через зовнішній ключ QuestionId в таблиці Answers, що посилається на Id таблиці Questions.

Дана структура бази даних забезпечує зберігання основної інформації, необхідної для функціонування системи тестування та управління користувачами. Атрибути, такі як LessonId та SchoolId в таблиці Questions,

вказують на можливість групування питань за уроками та створення кастомного контенту для різних навчальних закладів або викладачів.

2.3.2 Архітектура шару доступу до даних (Data Access Layer)

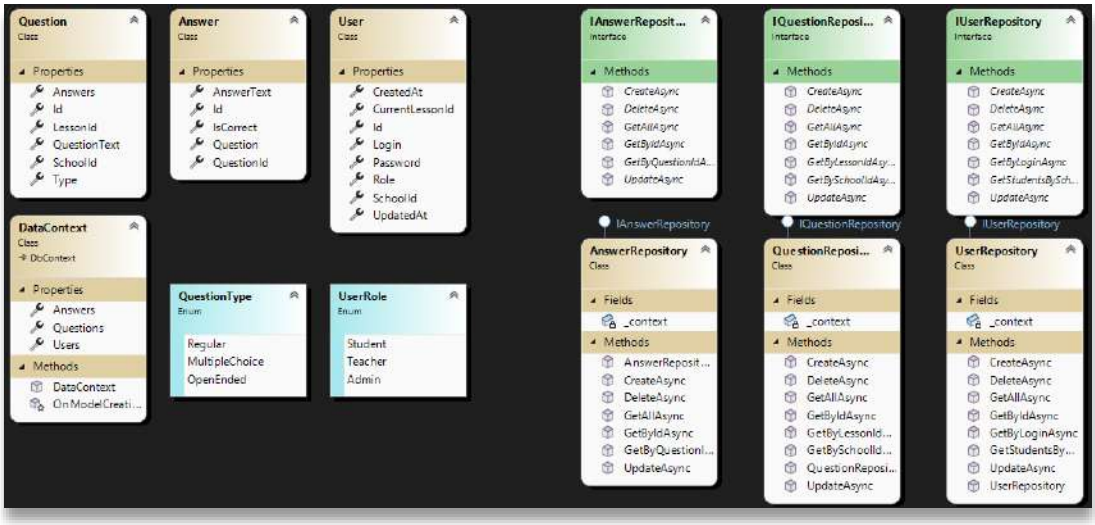

Рисунок 2.2

На Рисунку 2.3.2-1 відображено структуру класів, що належить до шару доступу до даних (DAL).

- Бізнес-сутності (Entities):
  - Question: Представляє питання. Властивості включають Id, LessonId, SchoolId, QuestionText, Type (типу QuestionType) та колекцію Answers.
  - Answer: Представляє варіант відповіді. Властивості: Id, QuestionId, AnswerText, IsCorrect.
  - User: Представляє користувача. Властивості: Id, Login, Password, Role (типу UserRole), SchoolId, CurrentLessonId, CreatedAt, UpdatedAt.
- Перелічення (Enums):
  - QuestionType: Визначає типи питань (Regular, MultipleChoice, OpenEnded).
  - UserRole: Визначає ролі користувачів (Student, Teacher, Admin).
- Контекст даних (DbContext):

- DataContext: Клас, що успадковується від DbContext Entity Framework Core. Він є мостом між доменними моделями та базою даних. Містить властивості типу DbSet<TEntity> для кожної сутності (Questions, Answers, Users), що дозволяє виконувати CRUD-операції. Метод OnModelCreating використовується для конфігурації моделі даних та зв'язків.

- Інтерфейси та реалізації репозиторіїв (Repositories):
  - IQuestionRepository, IAnswerRepository, IUserRepository (Інтерфейси): Визначають контракти для доступу до даних відповідних сутностей. Включають методи, такі як CreateAsync, DeleteAsync, GetAllAsync, GetByIdAsync, GetByLessonIdAsync (для питань), GetByQuestionIdAsync (для відповідей), GetByLoginAsync (для користувачів), UpdateAsync тощо.
  - QuestionRepository, AnswerRepository, UserRepository (Класи): Конкретні реалізації вищезазначених інтерфейсів. Кожен репозиторій інкапсулює логіку взаємодії з DataContext для маніпулювання даними відповідної сутності. Вони містять приватне поле _context типу DataContext, яке впроваджується через конструктор (Dependency Injection).

2.3.3 Архітектура шару бізнес логіки (Business Logic Layer)

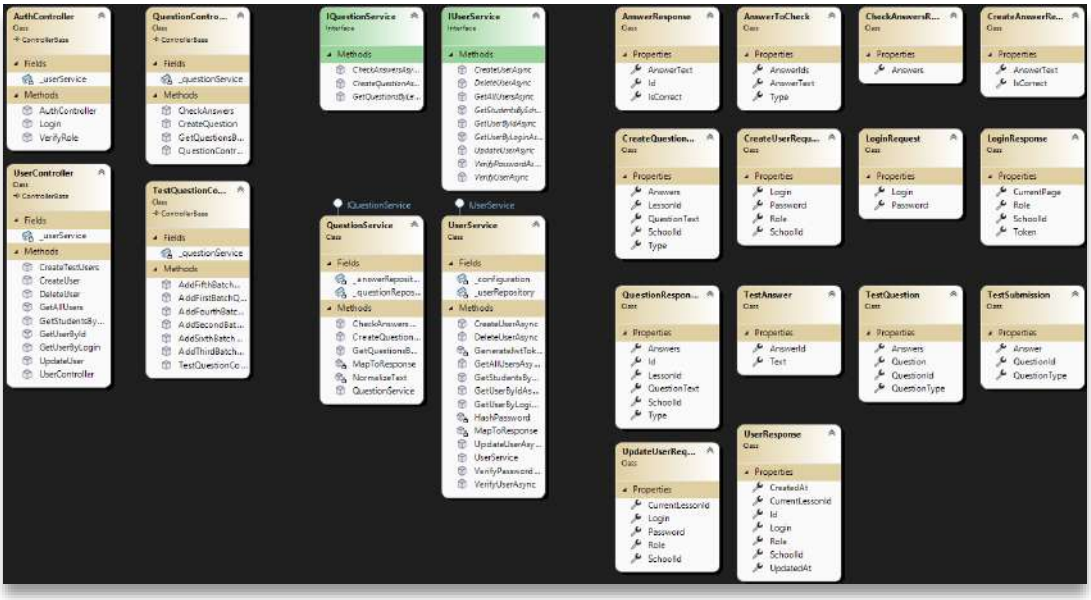

Рисунок 2.3

На Рисунку 2.3.3-1 показано класи, що відноситься до шару бізнес логіки.

- Інтерфейси та реалізації сервісів (Services):
  - IQuestionService, IUserService (Інтерфейси): Визначають контракти для бізнес-логіки, пов'язаної з питаннями та користувачами відповідно.
  - QuestionService, UserService (Класи): Реалізації сервісних інтерфейсів. Вони інкапсулюють бізнес-логіку, таку як створення питань, отримання питань за різними критеріями, створення користувачів, автентифікація, валідація даних тощо. QuestionService залежить від IAnswerRepository та IQuestionRepository. UserService залежить від IConfiguration та IUserRepository.
- Контролери API (API Controllers):
  - AuthController: Відповідає за обробку запитів, пов'язаних з автентифікацією (наприклад, Login, VerifyRole). Залежить від IUserService.
  - UserController: Обробляє запити, пов'язані з користувачами (наприклад, CreateUser, DeleteUser, GetAllUsers, GetUserById, GetUserByLogin, UpdateUser). Залежить від IUserService.

- TestQuestionController: Керує запитами стосовно тестових питань та відповідей (наприклад, AddQuestionBatch, AddThirdBatch, CheckAnswers, CreateQuestion, GetQuestionById, GetQuestionList, UpdateQuestion). Залежить від IQuestionService. Всі контролери успадковуються від базового класу ControllerBase ASP.NET Core.

- Об'єкти передачі даних (Data Transfer Objects - DTOs):
  - Запити (Requests): Наприклад, CreateQuestionRequest (містить LessonId, QuestionText, SchoolId, Type, колекцію Answers), CreateUserRequest (Login, Password, Role, SchoolId), LoginRequest (Login, Password), UpdateUserRequest (CurrentLessonId, Login, Password, Role, SchoolId). Ці об'єкти використовуються для передачі даних від клієнта до сервера при створенні або оновленні ресурсів.
  - Відповіді (Responses): Наприклад, AnswerResponse (AnswerText, Id, IsCorrect), CheckAnswersResponse (Properties), CreateAnswerResponse (Properties), QuestionResponse (Answers, Id, LessonId, QuestionText, SchoolId, Type), LoginResponse (CurrentPage, Role, SchoolId, Token), UserResponse (CreatedAt, CurrentLessonId, Id, Login, Password, Role, SchoolId, UpdatedAt). Ці об'єкти використовуються для структурування даних, що повертаються сервером клієнту.
- Інші DTO: AnswerToCheck (для перевірки відповідей), TestAnswer (для представлення відповіді в тесті), TestQuestion (для представлення питання в тесті), TestSubmission (для відправки результатів тесту).

2.4 Висновки до розділу 2

У цьому розділі було закладено теоретичний фундамент для розробки електронного курсу. Сформульовано функціональні та нефункціональні вимоги

до системи, а також обґрунтовано вибір технологічного стеку та архітектурних рішень. Детальний розгляд архітектурних патернів та принципів проектування (включаючи ООП та SOLID) підкреслює науковий підхід до створення надійного та масштабованого програмного забезпечення. Розроблена концептуальна модель даних дозволила створити міцну основу для подальшої практичної реалізації.

РОЗДІЛ 3. ДЕМОНСТРАЦІЯ, ВИСНОВКИ ТА РЕКОМЕНДАЦІЇ ЩОДО ВИКОРИСТАННЯ ЕЛЕКТРОННОГО КУРСУ “ВСТУП ДО ПРОГРАМУВАННЯ”

У даному розділі підсумовуються основні результати, отримані в ході виконання магістерської роботи, що полягала у розробці інтерактивного електронного курсу "Вступ до програмування" на базі JavaScript для учнів загальноосвітніх шкіл. Також надаються рекомендації щодо практичного використання розробленого програмного продукту та окреслюються перспективи його подальшого розвитку.

3.1 Демонстрація функціоналу електронного курсу

Для наочної ілюстрації результатів розробки та ключових можливостей створеного електронного курсу "Вступ до програмування", у цьому підрозділі представлено серію скріншотів основних інтерфейсів користувача. Ці зображення демонструють як процес взаємодії учня з навчальним контентом, так і функціонал, доступний для викладача.

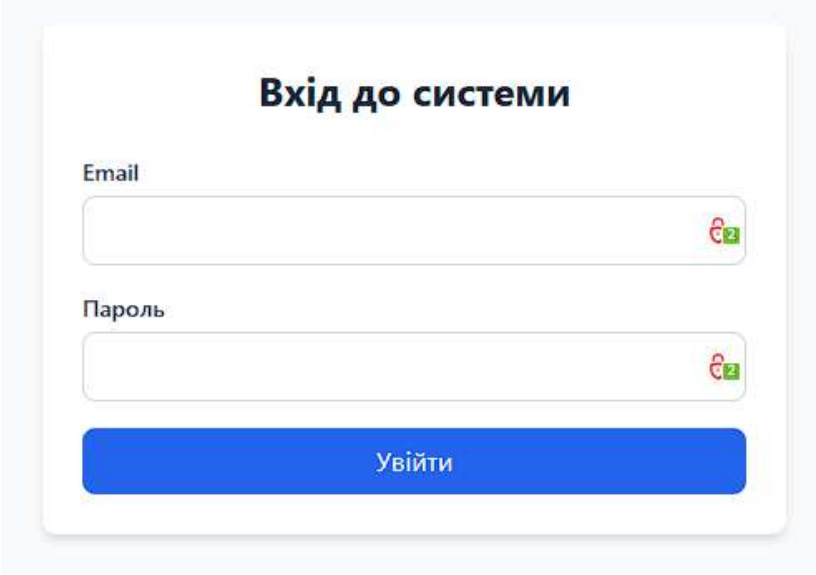

Рисунок 3.1

На Рисунку 3.1 представлено сторінку автентифікації користувача. Інтерфейс є мінімалістичним та інтуїтивно зрозумілим. Користувач має ввести свої облікові дані (логін та пароль) для отримання доступу до системи.

Рисунок 3.2

На Рисунку 3.2 представлена структура навчального матеріалу, розбита на розділи, модулі та уроки.

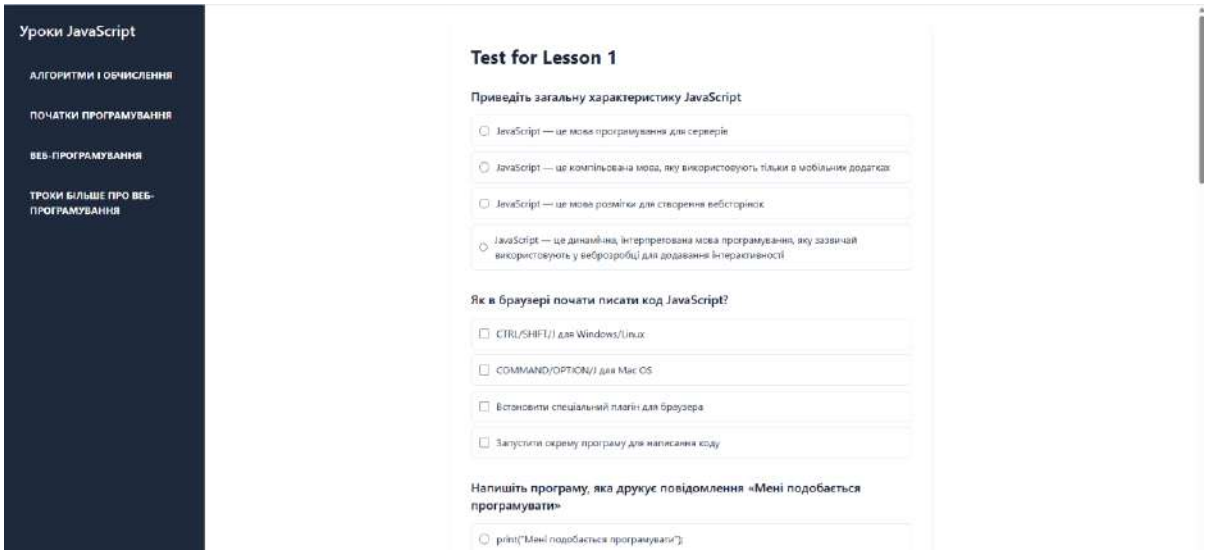

Рисунок 3.3

На Рисунку 3.3 показано інтерфейс проходження тесту. На екрані відображається текст питання та варіанти відповідей.

Рисунок 3.4

Після завершення тесту система автоматично перевіряє відповіді та надає миттєвий зворотний зв'язок. На Рисунку 3.4 продемонстровано екран з результатами, де учень може побачити загальний результат, а також які відповіді були правильними, а які – хибними. Це сприяє кращому засвоєнню матеріалу та роботі над помилками.

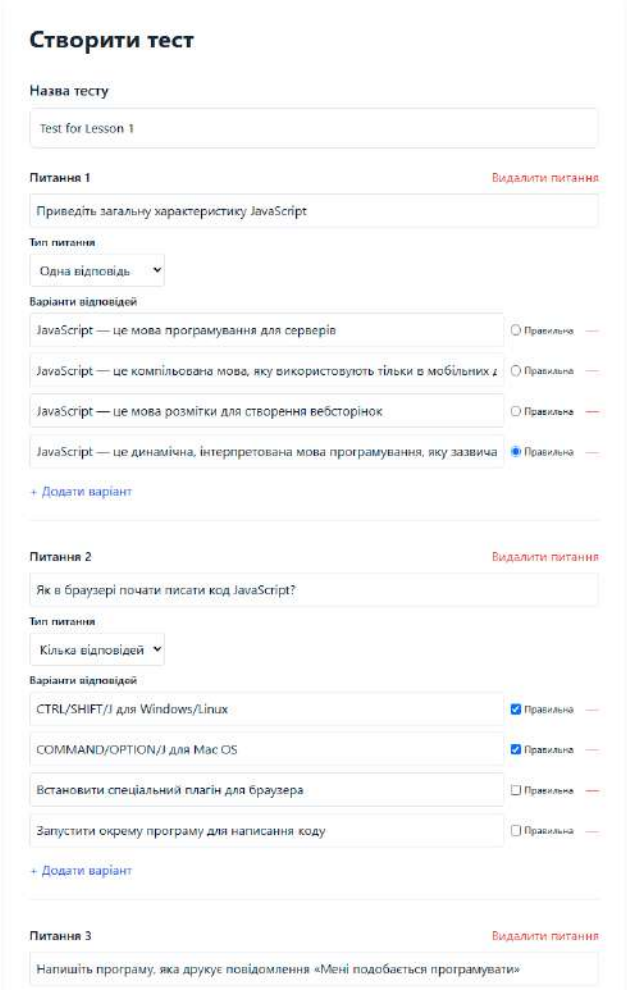

Рисунок 3.5

Для викладачів та адміністраторів розроблено спеціальний інтерфейс для управління навчальним контентом. На Рисунку 3.5 показано функціонал редагування/створення тестових питань. Викладач може вводити текст питання, додавати варіанти відповідей, позначати правильні з них та зберігати зміни. Цей інструмент дозволяє гнучко адаптувати та розширювати базу тестових завдань курсу.

3.2 Рекомендації щодо розгортання системи

Для успішного впровадження та функціонування розробленого електронного курсу необхідно виконати розгортання його програмних

компонентів. Система складається з трьох основних частин: серверна частина (Backend), клієнтська частина (Frontend) та база даних (Database). Існує два основних підходи до їх розгортання: традиційний (ручний) та з використанням контейнеризації Docker.

3.2.1 Ручне розгортання без контейнеризації

Цей підхід передбачає налаштування кожного компонента системи окремо безпосередньо на серверному обладнанні.

- Серверна частина (Backend - .NET): Потребує встановлення на сервері .NET 8.0 Runtime та налаштування веб-сервера (наприклад, IIS для Windows або Nginx як зворотний проксі для Kestrel на Linux) для обробки запитів.
- База даних (MS SQL Server): Вимагає доступу до працюючого екземпляра Microsoft SQL Server 2022. Схема бази даних створюється шляхом застосування міграцій Entity Framework Core (dotnet ef database update).

Клієнтська частина (Frontend - Next.js): Існує кілька способів розгортання:

  - На спеціалізованих платформах (Vercel, Netlify): Найпростіший та рекомендований спосіб. Платформа автоматично розгортає та оптимізує застосунок після підключення до Git-репозиторію.
  - На власному сервері з Node.js: Вимагає встановлення Node.js, збірки проєкту (npm run build) та запуску у продуктивному режимі (npm start).

- Конфігурація: Необхідно вручну відредагувати файли конфігурації (appsettings.json для бекенду та .env для фронтенду), вказавши коректні адреси, рядки підключення та секретні ключі.

3.2.2 Розгортання з використанням Docker

Цей підхід є рекомендованим для більшості сучасних проєктів, оскільки він значно спрощує розгортання, забезпечує узгодженість середовища та портативність. Технологія Docker дозволяє "запакувати" кожен компонент системи (бекенд, фронтенд, база даних) у свій ізольований контейнер, який містить усі необхідні для роботи залежності.

Етапи розгортання

- Створення Dockerfile: Для серверної (.NET) та клієнтської (Next.js) частин створюються окремі файли Dockerfile. У них містяться інструкції для Docker, як правильно зібрати образи застосунків.
- Створення docker-compose.yml: У цьому файлі описуються всі сервіси системи:
    - Сервіс бази даних: Базується на офіційному образі (напр., mcr.microsoft.com/mssql/server), отримує налаштування через змінні середовища та використовує том (volume) для зберігання даних.
    - Сервіси Backend та Frontend: Збираються на основі відповідних Dockerfile. Усі конфігураційні дані (адреси, ключі) передаються їм через змінні середовища.
    - Всі сервіси об'єднуються в єдину віртуальну мережу для взаємодії.
- Запуск та ініціалізація:
    - Весь програмний комплекс запускається однією командою: docker-compose up -d.
    - Після першого запуску необхідно застосувати міграції Entity Framework Core для створення структури бази даних у контейнері (в коді реалізована перевірка на відповідність бази, тому цей крок є необов’язковим).

3.3 Основні висновки за результатами виконаної роботи

3.3.1 Коротке узагальнення проведених досліджень, проектування та реалізації електронного курсу.

Магістерська робота була спрямована на вирішення актуальної проблеми нестачі якісних україномовних інтерактивних освітніх ресурсів для раннього навчання школярів основам програмування. Результатом проведеної роботи стало створення та впровадження готового до використання електронного курсу "Вступ до програмування".

На етапі дослідження було здійснено комплексний аналіз сучасних підходів до навчання програмуванню школярів, включаючи методики візуального блочного та текстового програмування. Було вивчено та узагальнено функціонал і методики провідних світових та українських освітніх онлайн-платформ (таких як Kahoot, Mentimeter, Quizlet, Nearpod, Socrative), що дозволило виявити ефективні практики інтерактивного навчання та сформувати вимоги до власної розробки. Теоретичною основою для педагогічно-методичних аспектів курсу слугував навчальний посібник "Вступ до дитячого програмування. Мова JavaScript" проф. М. Глибовця[1], а також стандарти мови JavaScript (ECMAScript [2], MDN Web Docs [3]).

На етапі проєктування на основі аналізу педагогічно-методичних основ створення україномовного інтерактивного курсу було спроєктовано структуру онлайн-курсу, а також принципи організації перевірки знань. Ключовим етапом стало проєктування архітектури програмного забезпечення електронного курсу відповідно до сучасних стандартів веб-розробки. Було обґрунтовано вибір клієнт-серверної моделі, N-Tier архітектури для серверної частини, принципів REST API, а також технологічного стеку, що включає Next.js (React, TypeScript, Tailwind CSS) для фронтенду, .NET 8.0 (ASP.NET Core Web API) для бекенду та Microsoft SQL Server з Entity Framework Core для бази даних. Для фронтенд-частини було обрано архітектурну методологію Feature-Sliced Design. Також було розроблено концептуальну модель даних та деталізовано архітектуру ключових

компонентів системи, включаючи патерни автентифікації (JWT) та безпеки (CORS).

На етапі реалізації, відповідно до розробленої архітектури та проєктних рішень, було створено бекенд- та фронтенд-частини електронного курсу. Реалізовано основний функціонал, що включає реєстрацію та автентифікацію користувачів, доступ до навчальних матеріалів, інтерактивну систему тестування з різними типами завдань, а також унікальний механізм для адміністраторів/викладачів щодо додавання та редагування кастомних тестових питань. Курс було наповнено оригінальним україномовним контентом, що охоплює основи JavaScript, на базі посібника проф. М. ГЛИБОВЦЯ[1]. Було проведено комплексне тестування розробленого програмного продукту, що підтвердило його працездатність та відповідність поставленим вимогам.

3.3.2 Підтвердження досягнення поставлених мети та завдань магістерської роботи.

Метою даної магістерської роботи була розробка та впровадження інтерактивного електронного курсу "Вступ до програмування" на базі JavaScript, адаптованого для учнів загальноосвітніх шкіл, з акцентом на оригінальний україномовний контент та можливість розширення навчального матеріалу викладачами. Для досягнення поставленої мети було послідовно виконано наступні завдання:

- Проаналізовано та узагальнено сучасні підходи до навчання програмуванню для школярів, а також існуючі онлайн-платформи та їхні методики. У рамках першого розділу було проведено огляд актуальних педагогічних методик, включаючи візуальне та текстове програмування, та проаналізовано функціонал освітніх платформ, що дало змогу визначити ефективні практики та сформувати вимоги до власної розробки.
- Розроблено педагогічно-методичні основи створення україномовного інтерактивного курсу з програмування, що враховуватиме

особливості сприйняття інформації школярами на основі навчального посібника проф. М. ГЛИБОВЦЯ[1].

- Спроектовано архітектуру та функціональні модулі програмного забезпечення електронного курсу відповідно до сучасних стандартів веб-розробки. У другому розділі детально описано та обґрунтовано вибір клієнт-серверної моделі, N-Tier архітектури, принципів REST API, технологічного стеку (.NET, Next.js, React, SQL Server), архітектури фронтенд-частини (Feature-Sliced Design), а також розроблено модель даних та архітектуру ключових компонентів системи.
- Реалізовано бекенд- та фронтенд-частини електронного курсу, забезпечивши інтерактивність та зручність користувацького інтерфейсу. Відповідно до спроєктованої архітектури, було розроблено програмний код серверної та клієнтської частин системи. Реалізовано ключові інтерактивні елементи, включаючи навігацію по уроках, відображення навчального контенту та систему тестування, з акцентом на інтуїтивно зрозумілий інтерфейс для цільової аудиторії.
- Наповнено курс оригінальним україномовним контентом, що охоплює основи JavaScript, а також розроблено інтерактивні тестові завдання різних типів. Навчальні матеріали курсу, що базуються на посібнику проф. М. ГЛИБОВЦЯ[1], були адаптовані та інтегровані в систему. Розроблено та впроваджено базу тестових завдань різних типів для перевірки засвоєння знань учнями.
- Впроваджено функціонал для адміністраторів/викладачів щодо додавання та редагування кастомних тестових питань. Реалізовано спеціалізований інтерфейс та відповідну логіку на серверній стороні, що дозволяє викладачам розширювати базу тестових завдань власними питаннями, адаптуючи курс до потреб конкретних учнів або навчальних програм.
- Проведено тестування розробленого програмного продукту та оцінено його відповідність поставленим вимогам. Було здійснено тестування системи, а також оцінка юзабіліті інтерфейсу. Результати тестування підтвердили

працездатність реалізованого функціоналу та його відповідність функціональним і нефункціональним вимогам, визначеним на етапі проєктування.

3.4 Рекомендації щодо можливого використання результатів магістерської роботи

Результати, отримані в ході виконання даної магістерської роботи, зокрема розроблений електронний курс "Вступ до програмування" та його програмна платформа, мають значний потенціал для практичного застосування в освітньому процесі та для подальших розробок. Нижче наведено ключові рекомендації щодо їх використання.

- Впровадження електронного курсу в освітню практику:
  - Для загальноосвітніх шкіл, ліцеїв та гімназій: Розроблений курс рекомендується до використання як основний або допоміжний навчальний ресурс при викладанні інформатики у середніх класах для ознайомлення учнів з основами програмування на мові JavaScript. Він може слугувати ефективним інструментом для проведення уроків, організації самостійної роботи учнів та виконання практичних завдань.
  - Для позашкільних навчальних закладів: Курс може бути успішно інтегрований у програми гуртків з програмування, комп'ютерних клубів та центрів дитячої творчості, надаючи структурований та інтерактивний матеріал для початківців.
  - Для самостійного вивчення програмування школярами: Завдяки інтуїтивно зрозумілому інтерфейсу, україномовному контенту та інтерактивним тестам, курс може бути ефективно використаний учнями для самостійного опанування базових концепцій програмування, розвитку логічного мислення та підготовки до більш поглибленого вивчення IT-дисциплін.

- Використання розробленої архітектури та програмного коду як основи для подальших освітніх проєктів:
    - Спроєктована N-Tier архітектура серверної частини на ASP.NET Core, клієнтська частина на Next.js/React з використанням Feature-Sliced Design, а також реалізовані модулі (автентифікації, управління контентом, тестування) є гнучкими та масштабованими. Ці компоненти та архітектурні рішення можуть бути адаптовані та використані як основа для створення інших електронних навчальних курсів з різних предметних областей, особливо тих, що потребують інтерактивності та можливості кастомізації контенту.
    - Відкрита (за умови відповідного ліцензування) або частково доступна кодова база може стати корисним ресурсом для освітніх установ або команд розробників, які планують створювати власні україномовні освітні платформи.
- Рекомендації для викладачів щодо ефективного використання функціоналу кастомізації тестових питань:
    - Персоналізація навчання: Викладачам рекомендується активно використовувати можливість додавання власних тестових питань для адаптації курсу до індивідуального рівня підготовки учнів у класі або групі, а також для врахування специфіки навчальної програми конкретного закладу.
    - Поглиблене вивчення тем: Створення додаткових, більш складних або творчих питань до окремих тем допоможе стимулювати учнів до глибшого розуміння матеріалу та розвитку навичок вирішення нестандартних завдань.

3.5 Перспективи подальшого розвитку проєкту

Розроблений електронний курс "Вступ до програмування" є функціональною та готовою до використання платформою, однак існує низка перспективних напрямків для її подальшого вдосконалення та розширення функціоналу, що дозволить підвищити її ефективність та привабливість для користувачів.

- Розширення навчального контенту:
  - Додавання нових модулів та уроків з JavaScript: Включення більш поглиблених тем з JavaScript для учнів, які успішно засвоїли основи.
  - Розробка курсів з інших мов програмування: Адаптація платформи для створення вступних курсів з інших популярних мов, таких як Python, що розширить освітні можливості для школярів.
  - Інтеграція проєктно-орієнтованого навчання: Додавання модулів з розробкою невеликих практичних проєктів (наприклад, прості ігри, інтерактивні веб-сторінки), що дозволить учням застосовувати отримані знання на практиці.
- Впровадження елементів гейміфікації:
  - Розробка системи досягнень, балів, віртуальних нагород та рейтингів для підвищення мотивації учнів та залучення їх до активного навчання.
  - Введення ігрових сценаріїв та персонажів, що супроводжуватимуть учнів протягом курсу.
- Покращення інструментів для викладачів:
  - Розширення функціоналу кастомізації: надання можливості викладачам не лише додавати тестові питання, але й створювати власні уроки, змінювати послідовність існуючих матеріалів.
  - Розробка інструментів для аналітики та моніторингу успішності учнів: надання викладачам детальної статистики щодо прогресу окремих учнів та груп, виявлення складних тем.
- Проведення апробації та збір зворотного зв'язку:

- Організація пілотного впровадження курсу в декількох навчальних закладах для збору детального зворотного зв'язку від учнів та викладачів.
- Використання отриманих даних для ітераційного вдосконалення контенту, функціоналу та користувацького досвіду платформи.

- Забезпечення доступності (Accessibility):
  - Проведення аудиту доступності та впровадження необхідних змін для відповідності стандартам WCAG (Web Content Accessibility Guidelines), щоб зробити курс доступним для учнів з особливими освітніми потребами.

3.6 Висновки до розділу 3

У даному розділі було представлено ключові висновки, що підсумовують результати виконаної магістерської роботи з розробки електронного курсу "Вступ до програмування". Було підтверджено досягнення поставленої мети та виконання всіх завдань дослідження, а також детально висвітлено наукову новизну та практичне значення отриманих результатів. На основі проведеної роботи сформульовано конкретні рекомендації щодо впровадження розробленого програмного продукту в освітній процес загальноосвітніх шкіл та для самостійного навчання, а також щодо використання його архітектури для подальших розробок. Окреслено також перспективні напрямки для подальшого розвитку та вдосконалення створеного електронного курсу, що відкриває можливості для його довгострокового використання та розширення функціоналу. Представлені матеріали логічно завершують виклад дослідження та розробки, проведених у рамках магістерської роботи.

Список використаних джерел

1. Глибовець М. Вступ до дитячого програмування. Мова JavaScript : навч. посіб. Київ, 2019. 530 с.
2. ECMAScript® 2026 Language Specification. 17th edition (May 2025) [Електронний ресурс] / ECMA International. URL: https://tc39.es/ecma262/
3. JavaScript | MDN [Електронний ресурс] / Mozilla Developer Network. URL: https://developer.mozilla.org/uk/docs/Web/JavaScript
4. Sommerville I. Software Engineering. 10th ed. Harlow : Pearson Education Limited, 2016. 816 p.
5. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures : Doctoral dissertation. University of California, Irvine, 2000. 180 p. URL: https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf
6. Fowler M. Patterns of Enterprise Application Architecture. Boston : Addison-Wesley, 2003. 533 p.
7. Booch G., Maksimchuk R. A., Engle M. W., Young B. J., Conallen J., Houston K. A. Object-Oriented Analysis and Design with Applications. 3rd ed. Upper Saddle River, NJ : Addison-Wesley Professional, 2007. 720 p.
8. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Upper Saddle River, NJ : Prentice Hall, 2017. 432 p.
9. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Boston : Addison-Wesley, 1995. 395 p.

Додаток А. Основні логічні файли backend коду

Контроллери:

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Authorization;
using API.Models.Auth;
using API.Services.Interfaces;
using System.Security.Claims;

namespace API.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class AuthController : ControllerBase
    {
        private readonly IUserService _userService;

        public AuthController(IUserService userService)
        {
            _userService = userService;
        }

        [HttpPost("login")]
        [AllowAnonymous]
        public async Task<ActionResult<LoginResponse>> Login(LoginRequest request)
        {
            var response = await _userService.VerifyUserAsync(request);
            if (response == null)
                return Unauthorized(new { message = "Invalid login or password" });

            return Ok(response);
        }

        [HttpGet("verify-role")]
        [Authorize]
        public ActionResult<object> VerifyRole()
        {
            var roleClaim = User.FindFirst(ClaimTypes.Role);
            var userIdClaim = User.FindFirst(ClaimTypes.NameIdentifier);
            var userNameClaim = User.FindFirst(ClaimTypes.Name);
            var schoolIdClaim = User.FindFirst("SchoolId");

            return Ok(new
            {
                Role = roleClaim?.Value,
                UserId = userIdClaim?.Value,
                UserName = userNameClaim?.Value,
                SchoolId = schoolIdClaim?.Value,
                AllClaims = User.Claims.Select(c => new { c.Type, c.Value })
            });
        }
    }
}
```

```
using API.Models.Question;
using API.Services.Interfaces;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using System.Security.Claims;
```

```
namespace API.Controllers
{
    /// <summary>
    /// Controller for managing questions and answers.
    /// </summary>
    [ApiController]
    [Route("api/[controller]")]
    [Authorize]
    public class QuestionController : ControllerBase
    {
        private readonly IQuestionService _questionService;

        /// <summary>
        /// Initializes a new instance of the <see cref="QuestionController"/> class.
        /// </summary>
        /// <param name="questionService">The question service for handling business
logic.</param>
        public QuestionController(IQuestionService questionService)
        {
            _questionService = questionService;
        }

        /// <summary>
        /// Creates a new question with its answers.
        /// </summary>
        /// <param name="request">The question creation request.</param>
        /// <returns>The created question response.</returns>
        [HttpPost]
        [Authorize(Roles = "Teacher,Admin")]
        public async Task<ActionResult<QuestionResponse>>
CreateQuestion(CreateQuestionRequest request)
        {
            var schoolIdClaim = User.FindFirst("SchoolId");
            if (schoolIdClaim == null)
                return BadRequest(new { message = "SchoolId not found in token" });

            request.SchoolId = int.Parse(schoolIdClaim.Value);
            var question = await _questionService.CreateQuestionAsync(request);
            return Ok(question);
        }

        /// <summary>
        /// Updates an existing question with its answers.
        /// </summary>
        /// <param name="request">The question update request.</param>
        /// <returns>The updated question response.</returns>
        [HttpPut]
        [Authorize(Roles = "Teacher,Admin")]
        public async Task<ActionResult<QuestionResponse>>
UpdateQuestion(CreateQuestionRequest request)
        {
            var schoolIdClaim = User.FindFirst("SchoolId");
            if (schoolIdClaim == null)
                return BadRequest(new { message = "SchoolId not found in token" });

            request.SchoolId = int.Parse(schoolIdClaim.Value);
            var question = await _questionService.UpdateQuestionAsync(request);
            return Ok(question);
        }

        /// <summary>
        /// Creates or updates multiple questions.
        /// </summary>
        /// <param name="requests">The list of questions to create or update.</param>
        /// <param name="lessonId">The id of the lesson</param>
        /// <returns>A collection of created/updated question responses.</returns>
```

```
        [HttpPost("list/{lessonId}")]
        [Authorize(Roles = "Teacher,Admin")]
        public async Task<ActionResult<IEnumerable<QuestionResponse>>>
CreateOrUpdateQuestions(List<CreateQuestionRequest> requests, double lessonId)
        {
            var schoolIdClaim = User.FindFirst("SchoolId");
            if (schoolIdClaim == null)
                return BadRequest(new { message = "SchoolId not found in token" });

            var schoolId = int.Parse(schoolIdClaim.Value);

            // Delete all existing questions for this lesson and school
            await _questionService.DeleteQuestionsByLessonAndSchoolIdAsync(lessonId,
schoolId);

            if (!requests.Any())
                return Ok();

            // Create new questions
            var results = new List<QuestionResponse>();
            foreach (var request in requests)
            {
                request.SchoolId = schoolId;
                var result = await _questionService.CreateQuestionAsync(request);
                results.Add(result);
            }

            return Ok(results);
        }

        /// <summary>
        /// Retrieves all questions for a specific lesson and school.
        /// </summary>
        /// <param name="lessonId">The ID of the lesson.</param>
        /// <returns>A collection of questions with their answers (without correct
answer information).</returns>
        [HttpGet("lesson/{lessonId}")]
        public async Task<ActionResult<IEnumerable<QuestionResponse>>>
GetQuestionsByLessonId(double lessonId)
        {
            var schoolIdClaim = User.FindFirst("SchoolId");
            if (schoolIdClaim == null)
                return BadRequest(new { message = "SchoolId not found in token" });

            var schoolId = int.Parse(schoolIdClaim.Value);

            var standardQuestions = await
_questionService.GetQuestionsByLessonAndSchoolIdAsync(lessonId, 0);
            var schoolQuestions = await
_questionService.GetQuestionsByLessonAndSchoolIdAsync(lessonId, schoolId);

            var allQuestions = standardQuestions.Concat(schoolQuestions);

            return Ok(allQuestions);
        }

        /// <summary>
        /// Retrieves questions for a specific lesson and school without standart
questions.
        /// </summary>
        /// <param name="lessonId">The ID of the lesson.</param>
        /// <returns>A collection of questions with their answers (without correct
answer information).</returns>
        [HttpGet("lesson/edit/{lessonId}")]
        public async Task<ActionResult<IEnumerable<QuestionResponse>>>
GetQuestionsByLessonIdForEdit(double lessonId)
```

```
        {
            var schoolIdClaim = User.FindFirst("SchoolId");
            if (schoolIdClaim == null)
                return BadRequest(new { message = "SchoolId not found in token" });

            var schoolId = int.Parse(schoolIdClaim.Value);

            var schoolQuestions = await
_questionService.GetQuestionsByLessonAndSchoolIdAsync(lessonId, schoolId);

            return Ok(schoolQuestions);
        }

        /// <summary>
        /// Checks student's answers and returns the score.
        /// </summary>
        /// <param name="request">The request containing answers to check.</param>
        /// <returns>The number of correct answers.</returns>
        [HttpPost("check")]
        public async Task<ActionResult<int>> CheckAnswers(CheckAnswersRequest request)
        {
            var score = await _questionService.CheckAnswersAsync(request);
            return Ok(score);
        }
    }
}



using API.Models.User;

using API.Services.Interfaces;
using Database.Models;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

namespace API.Controllers
{
    /// <summary>
    /// Controller for managing system users
    /// </summary>
    [ApiController]
    [Route("api/[controller]")]
    [Authorize(Roles = "Admin")]
    public class UserController : ControllerBase
    {
        private readonly IUserService _userService;

        /// <summary>
        /// Initializes a new instance of the UserController class
        /// </summary>
        /// <param name="userService">Service for working with users</param>
        public UserController(IUserService userService)
        {
            _userService = userService;
        }

        /// <summary>
        /// Gets a list of all users in the system
        /// </summary>
        /// <returns>Collection of users</returns>
        /// <response code="200">Returns the list of users</response>
        /// <response code="401">User is not authenticated</response>
        /// <response code="403">User does not have required permissions</response>
        [HttpGet]
        public async Task<ActionResult<IEnumerable<UserResponse>>> GetAllUsers()
```

```
        {
            var users = await _userService.GetAllUsersAsync();
            return Ok(users);
        }

        /// <summary>
        /// Gets a user by their ID
        /// </summary>
        /// <param name="id">User ID</param>
        /// <returns>User information</returns>
        /// <response code="200">Returns the user information</response>
        /// <response code="404">User not found</response>
        /// <response code="401">User is not authenticated</response>
        /// <response code="403">User does not have required permissions</response>
        [HttpGet("{id}")]
        public async Task<ActionResult<UserResponse>> GetUserById(int id)
        {
            var user = await _userService.GetUserByIdAsync(id);
            if (user == null)
                return NotFound(new { message = $"User with ID {id} not found" });

            return Ok(user);
        }

        /// <summary>
        /// Gets a user by their login
        /// </summary>
        /// <param name="login">User login</param>
        /// <returns>User information</returns>
        /// <response code="200">Returns the user information</response>
        /// <response code="404">User not found</response>
        /// <response code="401">User is not authenticated</response>
        /// <response code="403">User does not have required permissions</response>
        [HttpGet("login/{login}")]
        public async Task<ActionResult<UserResponse>> GetUserByLogin(string login)
        {
            var user = await _userService.GetUserByLoginAsync(login);
            if (user == null)
                return NotFound(new { message = $"User with login {login} not found"
});

            return Ok(user);
        }

        /// <summary>
        /// Gets a list of students for a specific school
        /// </summary>
        /// <param name="schoolId">School ID</param>
        /// <returns>Collection of students</returns>
        /// <response code="200">Returns the list of students</response>
        /// <response code="401">User is not authenticated</response>
        /// <response code="403">User does not have required permissions</response>
        [HttpGet("school/{schoolId}")]
        public async Task<ActionResult<IEnumerable<UserResponse>>>
GetStudentsBySchool(int schoolId)
        {
            var students = await _userService.GetStudentsBySchoolIdAsync(schoolId);
            return Ok(students);
        }

        /// <summary>
        /// Creates a new user
        /// </summary>
        /// <param name="request">User creation data</param>
        /// <returns>Created user information</returns>
        /// <response code="201">User successfully created</response>
```

```
        /// <response code="400">Invalid request data</response>
        /// <response code="401">User is not authenticated</response>
        /// <response code="403">User does not have required permissions</response>
        [HttpPost]
        public async Task<ActionResult<UserResponse>> CreateUser(CreateUserRequest
request)
        {
            try
            {
                var user = await _userService.CreateUserAsync(request);
                return CreatedAtAction(nameof(GetUserById), new { id = user.Id },
user);
            }
            catch (Exception ex)
            {
                return BadRequest(new { message = ex.Message });
            }
        }

        /// <summary>
        /// Updates user information
        /// </summary>
        /// <param name="id">User ID</param>
        /// <param name="request">Update data</param>
        /// <returns>Updated user information</returns>
        /// <response code="200">User successfully updated</response>
        /// <response code="400">Invalid request data</response>
        /// <response code="404">User not found</response>
        /// <response code="401">User is not authenticated</response>
        /// <response code="403">User does not have required permissions</response>
        [HttpPut("{id}")]
        public async Task<ActionResult<UserResponse>> UpdateUser(int id,
UpdateUserRequest request)
        {
            try
            {
                var user = await _userService.UpdateUserAsync(id, request);
                return Ok(user);
            }
            catch (KeyNotFoundException)
            {
                return NotFound(new { message = $"User with ID {id} not found" });
            }
            catch (Exception ex)
            {
                return BadRequest(new { message = ex.Message });
            }
        }

        /// <summary>
        /// Deletes a user
        /// </summary>
        /// <param name="id">User ID</param>
        /// <returns>Operation result</returns>
        /// <response code="204">User successfully deleted</response>
        /// <response code="404">User not found</response>
        /// <response code="401">User is not authenticated</response>
        /// <response code="403">User does not have required permissions</response>
        [HttpDelete("{id}")]
        public async Task<IActionResult> DeleteUser(int id)
        {
            var result = await _userService.DeleteUserAsync(id);
            if (!result)
                return NotFound(new { message = $"User with ID {id} not found" });

            return NoContent();
```

```
        }

    }
}
```

Сервіси:

```
using API.Models.Question;
using Database.Models;

namespace API.Services.Interfaces
{
    /// <summary>
    /// Interface for managing question-related operations.
    /// Defines methods for question creation and retrieval.
    /// </summary>
    public interface IQuestionService
    {
        /// <summary>
        /// Creates a new question with its answers in the system.
        /// </summary>
        /// <param name="request">The question creation request model containing
question text, type, and answers.</param>
        /// <returns>The created question response model.</returns>
        Task<QuestionResponse> CreateQuestionAsync(CreateQuestionRequest request);

        /// <summary>
        /// Updates an existing question with its answers.
        /// </summary>
        /// <param name="request">The question update request model containing the ID
and updated data.</param>
        /// <returns>The updated question response model.</returns>
        Task<QuestionResponse> UpdateQuestionAsync(CreateQuestionRequest request);

        /// <summary>
        /// Retrieves all questions for a specific lesson and school without exposing
correct answers.
        /// </summary>
        /// <param name="lessonId">The unique identifier of the lesson.</param>
        /// <param name="schoolId">The unique identifier of the school.</param>
        /// <returns>A collection of question response models without correct
answers.</returns>
        Task<IEnumerable<QuestionResponse>> GetQuestionsByLessonAndSchoolIdAsync(double
lessonId, int schoolId);

        /// <summary>
        /// Checks student's answers and calculates the score.
        /// </summary>
        /// <param name="request">The request containing answers to check.</param>
        /// <returns>The number of correct answers.</returns>
        Task<int> CheckAnswersAsync(CheckAnswersRequest request);

        /// <summary>
        /// Deletes all questions for a specific lesson and school.
        /// </summary>
        /// <param name="lessonId">The unique identifier of the lesson.</param>
        /// <param name="schoolId">The unique identifier of the school.</param>
        /// <returns>True if the operation was successful.</returns>
        Task<bool> DeleteQuestionsByLessonAndSchoolIdAsync(double lessonId, int
schoolId);
    }
}
```

```
using API.Models.Auth;
using API.Models.User;
using Database.Models;

namespace API.Services.Interfaces
{
    /// <summary>
    /// Interface for managing user-related operations.
    /// Defines methods for user authentication, creation, and management.
    /// </summary>
    public interface IUserService
    {
        /// <summary>
        /// Verifies user credentials and returns authentication response if
successful.
        /// </summary>
        /// <param name="request">The login request containing user
credentials.</param>
        /// <returns>The login response containing user information and JWT token if
authentication is successful; otherwise, null.</returns>
        Task<LoginResponse?> VerifyUserAsync(LoginRequest request);

        /// <summary>
        /// Retrieves all users from the database.
        /// </summary>
        /// <returns>A collection of user response models.</returns>
        Task<IEnumerable<UserResponse>> GetAllUsersAsync();

        /// <summary>
        /// Retrieves a specific user by their unique identifier.
        /// </summary>
        /// <param name="id">The unique identifier of the user to retrieve.</param>
        /// <returns>The user response model if found; otherwise, null.</returns>
        Task<UserResponse?> GetUserByIdAsync(int id);

        /// <summary>
        /// Retrieves a user by their login identifier.
        /// </summary>
        /// <param name="login">The login identifier of the user to retrieve.</param>
        /// <returns>The user response model if found; otherwise, null.</returns>
        Task<UserResponse?> GetUserByLoginAsync(string login);

        /// <summary>
        /// Retrieves all students associated with a specific school.
        /// </summary>
        /// <param name="schoolId">The unique identifier of the school.</param>
        /// <returns>A collection of student response models belonging to the specified
school.</returns>
        Task<IEnumerable<UserResponse>> GetStudentsBySchoolIdAsync(int schoolId);

        /// <summary>
        /// Creates a new user in the system with the specified credentials.
        /// </summary>
        /// <param name="request">The user creation request model.</param>
        /// <returns>The created user response model.</returns>
        Task<UserResponse> CreateUserAsync(CreateUserRequest request);

        /// <summary>
        /// Updates an existing user's information and optionally their password.
        /// </summary>
        /// <param name="id">The unique identifier of the user to update.</param>
        /// <param name="request">The user update request model.</param>
        /// <returns>The updated user response model.</returns>
        Task<UserResponse> UpdateUserAsync(int id, UpdateUserRequest request);
```

```
        /// <summary>
        /// Removes a user from the system by their unique identifier.
        /// </summary>
        /// <param name="id">The unique identifier of the user to delete.</param>
        /// <returns>True if the user was successfully deleted; otherwise,
false.</returns>
        Task<bool> DeleteUserAsync(int id);

        /// <summary>
        /// Verifies if the provided password matches the user's stored credentials.
        /// </summary>
        /// <param name="user">The user entity to verify credentials for.</param>
        /// <param name="password">The password to verify against the user's stored
credentials.</param>
        /// <returns>True if the password matches; otherwise, false.</returns>
        Task<bool> VerifyPasswordAsync(User user, string password);
    }
}
```

Репозиторії:

```
using Database.Models;

namespace Database.Repositories
{
    /// <summary>
    /// Interface for managing Answer entities in the database
    /// </summary>
    public interface IAnswerRepository
    {
        /// <summary>
        /// Gets all answers from the database
        /// </summary>
        /// <returns>A collection of all answers</returns>
        Task<IEnumerable<Answer>> GetAllAsync();

        /// <summary>
        /// Gets an answer by its ID
        /// </summary>
        /// <param name="id">The ID of the answer to retrieve</param>
        /// <returns>The answer if found, null otherwise</returns>
        Task<Answer?> GetByIdAsync(int id);

        /// <summary>
        /// Gets all answers for a specific question
        /// </summary>
        /// <param name="questionId">The ID of the question</param>
        /// <returns>A collection of answers for the specified question</returns>
        Task<IEnumerable<Answer>> GetByQuestionIdAsync(int questionId);

        /// <summary>
        /// Creates a new answer in the database
        /// </summary>
        /// <param name="answer">The answer entity to create</param>
        /// <returns>The created answer with updated ID</returns>
        Task<Answer> CreateAsync(Answer answer);

        /// <summary>
        /// Updates an existing answer in the database
        /// </summary>
        /// <param name="answer">The answer entity with updated data</param>
        /// <returns>The updated answer</returns>
        Task<Answer> UpdateAsync(Answer answer);
```

```
        /// <summary>
        /// Deletes an answer from the database
        /// </summary>
        /// <param name="id">The ID of the answer to delete</param>
        /// <returns>True if the answer was deleted, false if not found</returns>
        Task<bool> DeleteAsync(int id);
    }
}


using Database.Models;

namespace Database.Repositories
{
    /// <summary>
    /// Interface for managing Question entities in the database
    /// </summary>
    public interface IQuestionRepository
    {
        /// <summary>
        /// Gets all questions from the database
        /// </summary>
        /// <returns>A collection of all questions</returns>
        Task<IEnumerable<Question>> GetAllAsync();

        /// <summary>
        /// Gets a question by its ID
        /// </summary>
        /// <param name="id">The ID of the question to retrieve</param>
        /// <returns>The question if found, null otherwise</returns>
        Task<Question?> GetByIdAsync(int id);

        /// <summary>
        /// Gets all questions for a specific lesson
        /// </summary>
        /// <param name="lessonId">The ID of the lesson</param>
        /// <returns>A collection of questions for the specified lesson</returns>
        Task<IEnumerable<Question>> GetByLessonIdAsync(int lessonId);

        /// <summary>
        /// Gets all questions for a specific school
        /// </summary>
        /// <param name="schoolId">The ID of the school</param>
        /// <returns>A collection of questions for the specified school</returns>
        Task<IEnumerable<Question>> GetBySchoolIdAsync(int schoolId);

        /// <summary>
        /// Creates a new question in the database
        /// </summary>
        /// <param name="question">The question entity to create</param>
        /// <returns>The created question with updated ID</returns>
        Task<Question> CreateAsync(Question question);

        /// <summary>
        /// Updates an existing question in the database
        /// </summary>
        /// <param name="question">The question entity with updated data</param>
        /// <returns>The updated question</returns>
        Task<Question> UpdateAsync(Question question);

        /// <summary>
        /// Deletes a question from the database
        /// </summary>
        /// <param name="id">The ID of the question to delete</param>
        /// <returns>True if the question was deleted, false if not found</returns>
        Task<bool> DeleteAsync(int id);
```

```
    }
}


using Database.Models;

namespace Database.Repositories
{
    /// <summary>
    /// Interface for managing User entities in the database
    /// </summary>
    public interface IUserRepository
    {
        /// <summary>
        /// Gets all users from the database
        /// </summary>
        /// <returns>A collection of all users</returns>
        Task<IEnumerable<User>> GetAllAsync();

        /// <summary>
        /// Gets a user by their ID
        /// </summary>
        /// <param name="id">The ID of the user to retrieve</param>
        /// <returns>The user if found, null otherwise</returns>
        Task<User?> GetByIdAsync(int id);

        /// <summary>
        /// Gets a user by their login
        /// </summary>
        /// <param name="login">The login of the user to retrieve</param>
        /// <returns>The user if found, null otherwise</returns>
        Task<User?> GetByLoginAsync(string login);

        /// <summary>
        /// Gets all students from a specific school
        /// </summary>
        /// <param name="schoolId">The ID of the school</param>
        /// <returns>A collection of students from the specified school</returns>
        Task<IEnumerable<User>> GetStudentsBySchoolIdAsync(int schoolId);

        /// <summary>
        /// Creates a new user in the database
        /// </summary>
        /// <param name="user">The user entity to create</param>
        /// <returns>The created user with updated ID</returns>
        Task<User> CreateAsync(User user);

        /// <summary>
        /// Updates an existing user in the database
        /// </summary>
        /// <param name="user">The user entity with updated data</param>
        /// <returns>The updated user</returns>
        Task<User> UpdateAsync(User user);

        /// <summary>
        /// Deletes a user from the database
        /// </summary>
        /// <param name="id">The ID of the user to delete</param>
        /// <returns>True if the user was deleted, false if not found</returns>
        Task<bool> DeleteAsync(int id);
    }
}
```

Додаток Б. Основні логічні файли frontend коду

middleware.ts

```
import { NextResponse } from "next/server";
import type { NextRequest } from "next/server";
import { getSession } from "@/shared/lib/auth";
import { getUserFromToken } from "./shared/utils/getUserFromToken";

export async function middleware(request: NextRequest) {
  const token = await getSession();
  const user = await getUserFromToken(token);

  const publicRoutes = [
    "/login",
    "/register",
    "/api/auth/login",
    "/lesson",
    "/api/lessons",
    "/images",
    "/",
  ];

  // Перевіряємо чи це публічний роут
  const isPublicRoute = publicRoutes.some(
    (route) =>
      request.nextUrl.pathname === route ||
      request.nextUrl.pathname.startsWith(route + "/")
  );

  if (isPublicRoute) {
    return NextResponse.next();
  }

  if (!user) {
    const returnUrl = encodeURIComponent(request.nextUrl.pathname);
    return NextResponse.redirect(
      new URL(`/login?returnUrl=${returnUrl}`, request.url)
    );
  }

  if (
    request.nextUrl.pathname.startsWith("/test/") &&
    (user.role === "teacher" || user.role === "admin")
  ) {

    const match = request.nextUrl.pathname.match(/^\/test\/((?:\d+\/)*\d+)/);
    const testParams = match ? match[1] : null;
    if (testParams) {
      return NextResponse.redirect(
```

```
        new URL(`/test/edit/${testParams}`, request.url)
      );
    }
  }

  return NextResponse.next();
}

export const config = {
  matcher: [
    /*
     * Match all request paths except for the ones starting with:
     * - _next/static (static files)
     * - _next/image (image optimization files)
     * - favicon.ico (favicon file)
     * - public folder
     * - images folder
     */
    "/((?!_next/static|_next/image|favicon.ico|public|images).*)",
  ],
};
```

auth.ts

```
import { loginRoute } from "@/pages/Auth/config/routes";
import { FormState, SignInFormSchema } from "@/pages/Auth/definitions";
import { backendApiUrl } from "@/shared/config/backend";
import { createSession } from "@/shared/lib/auth";
import { User } from "@/shared/types/auth";
import { redirect } from "next/navigation";

export async function signin(state: FormState, formData: FormData) {
  const returnUrl = formData.get("returnUrl") as string;
  console.log("returnUrl", returnUrl);
  const validatedFields = SignInFormSchema.safeParse({
    email: formData.get("email"),
    password: formData.get("password"),
  });

  if (!validatedFields.success) {
    return {
      errors: validatedFields.error.flatten().fieldErrors,
    };
  }

  const email = formData.get("email");
  const password = formData.get("password");

  const response = await fetch(`${backendApiUrl}${loginRoute}`, {
    method: "POST",
    headers: {
```

```
      "Content-Type": "application/json",
    },
    body: JSON.stringify({ login: email, password }),
  });

  if (!response.ok) {
    return {
      errors: {
        email: ["Invalid email or password"],
      },
    };
  }

  const data = (await response.json()) as User;

  await createSession(data.token);

  redirect(returnUrl || "/");
}
```

lesson/[...params]/layout.tsx

```
import Sidebar from "@/shared/ui/Sidebar/Sidebar";

export default function LessonLayout({ children }: { children: React.ReactNode }) {
  return (
    <div className="flex h-screen">
      <div className="fixed top-0 left-0 bottom-0 z-10">
        <Sidebar />
      </div>
      <div className="flex-1 ml-80">
        {children}
      </div>
    </div>
  );
}
```

lesson/[...params]/page.tsx

```
import Lesson from "@/pages/Lesson/ui";

export default async function LessonPage({
  params,
}: {
  params: { params: string[] };
}) {
  const { params: routeParams } = await params;
  const [sectionId, subsectionId, topicId] = routeParams || [];
  return (
    <Lesson
```

```
      sectionId={sectionId}
      subsectionId={subsectionId}
      topicId={topicId}
    />
  );
}
```

test/[…params]/page.tsx

```
import Test from "@/pages/Test/Test";
import { api } from "@/shared/config/backend";
import { getSession } from "@/shared/lib/auth";
import { Suspense } from "react";
import { Test as TestType, QuestionResponse } from "@/pages/Test/edit/TestPage";

interface PageProps {
  params: { params: string[] };
}

export default async function TestPage({ params }: PageProps) {
  const { params: routeParams } = await params;
  const [sectionId, subsectionId, topicId] = routeParams || [];
  const testSlug = `${sectionId}/${subsectionId}/${topicId}`;
  const lessonId = `${sectionId}.${subsectionId}`;
  const token = await getSession();
  let testData: TestType | undefined = undefined; // Use the imported TestType

  try {
    const response = await fetch(api.question.get(lessonId), {
      method: "GET",
      headers: {
        "Content-Type": "application/json",
        Authorization: `Bearer ${token}`,
      },
    });

    if (!response.ok) {
      console.error(
        `Error fetching questions for lesson ${lessonId}: ${response.status}`
      );
    } else {
      const questions: QuestionResponse[] = await response.json();

      if (Array.isArray(questions)) {
        testData = {
          id: lessonId,
          slug: testSlug,
          title: `Тест до розділу ${lessonId}`,
          questions: questions,
        };
      } else {
```

```
        console.error("Fetched data is not an array of questions:", questions);
      }
    }
  } catch (error) {
    console.error("Error during question data fetch:", error);
  }

  return (
    <Suspense fallback={<div>Завантаження...</div>}>
      {/* Pass the potentially undefined testData */}
      <Test test={testData} />
    </Suspense>
  );
}
```

auth.ts

```
export type User = {
  token: string;
  "http://schemas.microsoft.com/ws/2008/06/identity/claims/role" : "Student" |
"Teacher" | "Admin";
  schoolId: number;
};

export type AuthError = {
  message: string;
  code: string;
};
```

domain.ts

```
type Image = {
  type: "image";
  src: string;
  alt: string;
};

type Heading = {
  type: "heading";
  text: string;
};

type Subheading = {
  type: "subheading";
  text: string;
};

type Paragraph = {
```

```
  type: "paragraph";
  text: string;
};

type List = {
  type: "list";
  items: string[];
};

export type CodeSnippet = {
  type: "code";
  code: string;
  language?: string;
};

export type Content = Heading | Subheading | Paragraph | Image | List |
CodeSnippet;

export type Topic = {
  id: number;
  title: string;
  content: Content[];
};

export type Section = {
  id: number;
  title: string;
  topics: Topic[];
};

export type ContentSection = {
  id: number;
  title: string;
  sections: Section[];
};
```

currentUser.ts

```
import { createContext } from "react";

interface User {
  id: string;
  role: number;
  login: string;
}

export const CurrentUser = createContext<User | null>(null);
```