Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Програмна система для інтерактивного вивчення іноземних мов

(тема)

Виконав:
здобувач 4 року навчання
групи ПЗПІ-21-3

Костянтин БОНДАРЕНКО
(Власне ім'я, ПРІЗВИЩЕ)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Програмна інженерія
(повна назва освітньої програми)

Керівник ст.викл. кафедри ПІ Віталій ЛЯПОТА
(посада, Власне ім'я, ПРІЗВИЩЕ)

Допускається до захисту
Зав. кафедри _____ Кирило СМЕЛЯКОВ
(підпис) (Власне ім'я, ПРІЗВИЩЕ)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерних наук
Кафедра програмної інженерії
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми Освітньо-професійна
Освітня програма Програмна Інженерія
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Бондаренку Костянтину Тарасовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Програмна система для інтерактивного вивчення іноземних мов

Затверджена наказом по університету від 19.05. 2025р. № 397 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 23.06.2025

3. Вихідні дані до роботи Розробити веб-застосунок для інтерактивного вивчення іноземних мов, що надає користувачам можливість вибору з різних типів завдань (текстові, аудіо, переклад, побудова речень тощо) з підтримкою гейміфікації, автоматичним створенням карток для запам'ятовування слів і досягнень. Мови програмування: C#, Typescript.

4. Перелік питань, що потрібно опрацювати в роботі

Вступ, аналіз предметної галузі, формування вимог до програмної системи, архітектура та проектування програмного забезпечення, опис прийнятих програмних рішень, тестування розробленого програмного забезпечення, висновки, додатки.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	20.05.2025	*виконано*
2	Створення специфікації ПЗ	21.05.2025	*виконано*
3	Проектування ПЗ	23.05.2025	*виконано*
4	Розробка ПЗ	24.05.2025	*виконано*
5	Тестування ПЗ	30.05.2025	*виконано*
6	Оформлення пояснювальної записки	05.06.2025	*виконано*
7	Підготовка презентації та доповіді	18.06.2025	*виконано*
8	Попередній захист	20.06.2025	*виконано*
9	Нормоконтроль, рецензування	20.06.2025	*виконано*
10	Здача роботи у електронний архів	20.06.2025	*виконано*
11	Допуск до захисту у зав. кафедри	23.06.2025	*виконано*

Дата видачі завдання « 19 » «травня» 2025р.

Здобувач _____
(підпис)

Керівник роботи _____ ст.викл. кафедри ПІ Віталій ЛЯПОТА
(підпис) (посада, Власне ім’я, ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи бакалавра, 52 стор., 17 рис., 2 додатки, 1 табл., 12 джерел.

ВЕБ-ДОДАТОК, ВИВЧЕННЯ МОВ, ГЕЙМІФІКАЦІЯ, КАРТКИ, КАСТОМІЗАЦІЯ, ПЕРЕКЛАД, ANGULAR, ASP.NET, AZURE

Об’єктом розробки є програмна система для інтерактивного вивчення іноземних мов.

Метою роботи є створення веб-застосунку, який дозволяє користувачам адаптувати навчальний процес відповідно до власних потреб, використовуючи інтерактивні завдання, картки, систему досягнень.

Для реалізації рішення використано технології ASP.NET Core Web API для серверної частини, Angular для клієнтської частини, MSSQL для зберігання даних, хмарну інфраструктуру Microsoft Azure та інструменти контейнеризації Docker.

У результаті розробки створено функціональний веб-застосунок, що підтримує індивідуалізацію навчання, автоматичне формування завдань і карток, імпорт даних із зовнішніх джерел, а також елементи гейміфікації, які підвищують мотивацію користувачів.

ABSTRACT

WEB APPLICATION, LANGUAGE LEARNING, GAMIFICATION, FLASHCARDS, CUSTOMIZATION, TRANSLATION, ANGULAR, ASP.NET, AZURE

The object of the development is a software system for interactive foreign language learning.

The purpose of the project is to develop a web application that enables users to customize the learning process according to their individual needs by using interactive exercises, flashcards, and an achievement system.

The solution is implemented using ASP.NET Core Web API for the backend, Angular for the frontend, MSSQL for data storage, Microsoft Azure for cloud infrastructure, and Docker for containerization.

As a result, a functional web application was developed that supports personalized learning, automatic generation of tasks and flashcards, data import from external sources, and gamification elements that enhance user motivation.

ЗМІСТ

ВСТУП

Сучасні тенденції у сфері вивчення іноземних мов вказують на зростаючий попит на інтерактивні, персоналізовані та зручні у використанні програмні рішення. В умовах глобалізації, міграції та міжнародної співпраці знання іноземних мов є ключовим чинником особистого та професійного розвитку. Тому все більше користувачів звертаються до онлайн-платформ, які дозволяють ефективно опановувати мови у зручному темпі.

Популярні платформи, такі як Duolingo, Babbel та Rosetta Stone, дійсно зробили вивчення мов доступнішим. Однак вони мають обмеження, зокрема: недостатню кастомізацію навчального процесу, обмежений вибір типів завдань, відсутність підтримки імпорту власних матеріалів або синхронізації з іншими сервісами. Багатьом користувачам бракує інструментів для налаштування системи відповідно до індивідуальних цілей, стилю навчання та рівня володіння мовою.

Метою цієї кваліфікаційної роботи є розробка програмної системи для інтерактивного вивчення іноземних мов, яка дозволить користувачам створювати індивідуальні навчальні плани, додавати власні матеріали, автоматично генерувати картки для запам’ятовування нових слів, а також виконувати різні типи завдань (текстові, аудіо, переклад, побудова речень тощо). Також система буде містити елементи гейміфікації, такі як бали, досягнення, лідерборди та можливість змагань з друзями.

У межах проєкту буде реалізовано повноцінний веб-застосунок з використанням сучасного стеку технологій: Angular (Frontend), ASP.NET Web API (Backend), MSSQL (база даних), Microsoft Azure (хмарна інфраструктура), а також Docker (контейнеризація).

Розроблена система має на меті не лише забезпечити функціональність, подібну до існуючих платформ, а й надати додаткові можливості для гнучкого та ефективного навчання, орієнтованого на конкретного користувача.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі

На сьогоднішній день ринок цифрових рішень для вивчення іноземних мов представлений великою кількістю онлайн-платформ, мобільних застосунків та веб-сервісів. Найбільш популярними серед них є Duolingo [1], Babbel [2], Rosetta Stone [3], Busuu, Memrise та інші. Кожна з цих платформ має свої переваги, проте також і суттєві обмеження, що стримують персоналізацію навчального процесу.

Duolingo – один з найпоширеніших безкоштовних застосунків, який використовує гейміфікований підхід до навчання (див. рис. 1.1). Його сильними сторонами є простий інтерфейс, велика кількість мов, адаптивна система завдань та інтерактивні вправи. Водночас, Duolingo має обмежені можливості щодо кастомізації навчальних цілей, створення власного контенту та глибокого занурення у граматику.

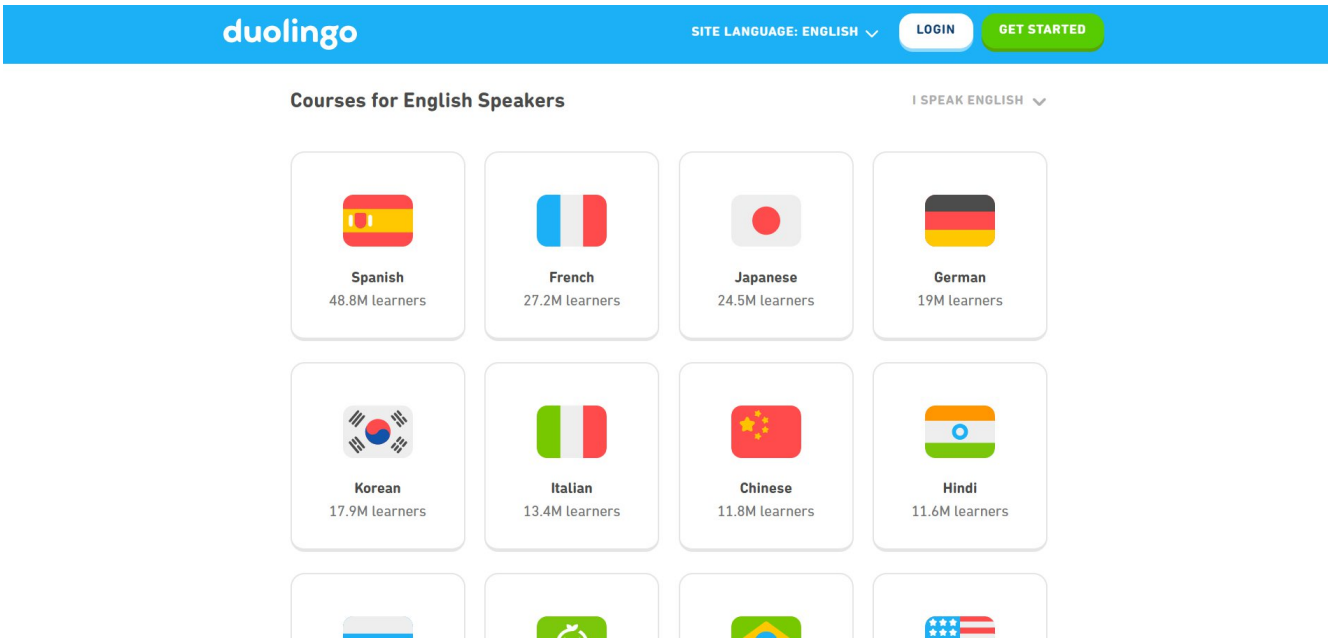

Рисунок 1.1 – Інтерфейс програмної системи “Duolingo” (на основі даних [1])

Babbel (див. рис. 1.2) орієнтується більше на структуроване навчання з наголосом на граматику і побудову речень. Цей сервіс має платну модель, що

забезпечує вищу якість контенту, але також обмежує доступність для широкого кола користувачів. Водночас, Babbel не підтримує глибоку персоналізацію або взаємодію з іншими платформами.

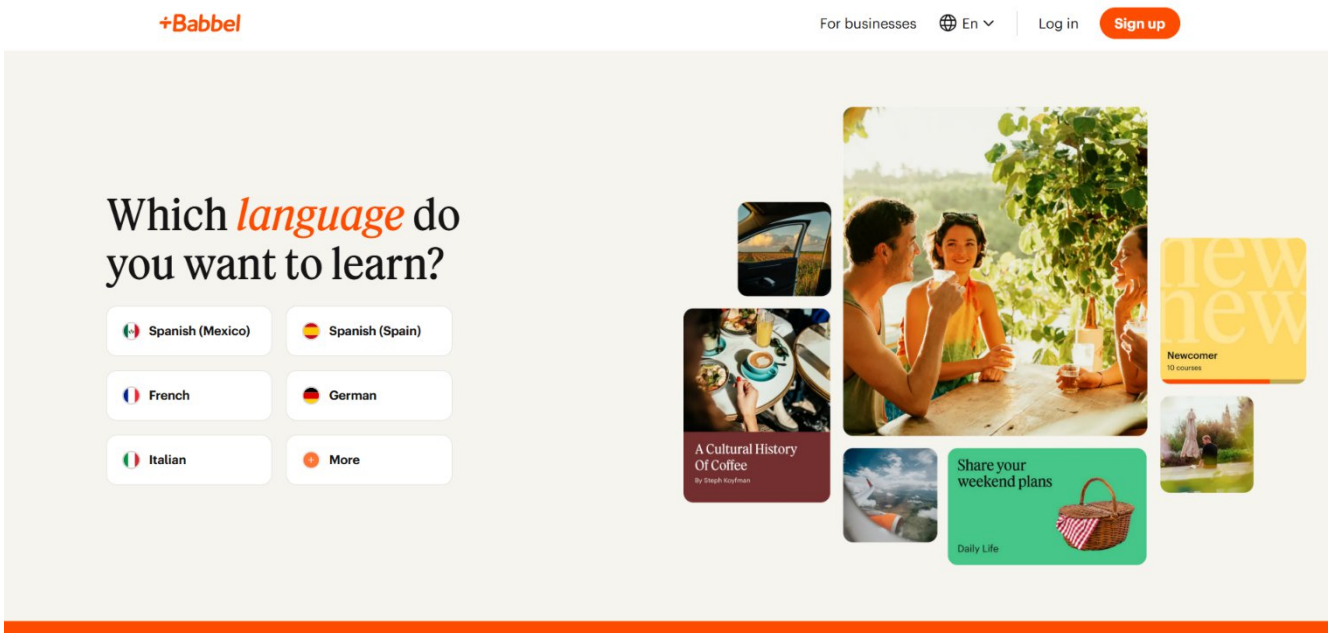

Рисунок 1.2 – Інтерфейс програмної системи “Babbel” (на основі даних [2])

Rosetta Stone (див. рис. 1.3) відомий своїм методом повного занурення – весь контент подається цільовою мовою без перекладів. Хоча цей підхід ефективний для деяких типів користувачів, він не завжди підходить для початківців. Також Rosetta Stone не має функцій створення власних навчальних матеріалів чи імпорту словникових баз.

Серед платформ, орієнтованих на запам'ятовування слів, можна виокремити Memrise [4] (див. рис. 1.4) та Anki [5] (див. рис. 1.5). Memrise використовує картки та відео з носіями мови, що покращує вимову та сприйняття. Anki, у свою чергу, дозволяє створювати власні картки з використанням принципів інтервального повторення, але не має централізованого навчального контенту та підтримки новачків.

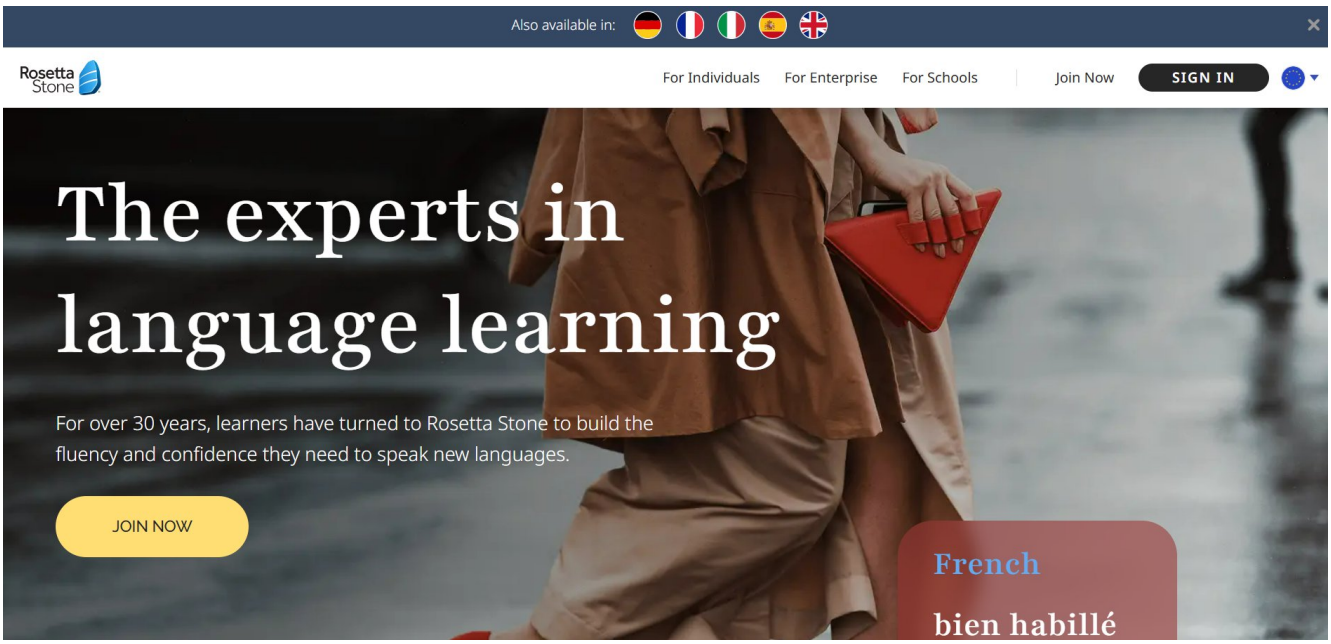

Рисунок 1.3 – Інтерфейс програмної системи “Rosetta Stone” (на основі даних [3])

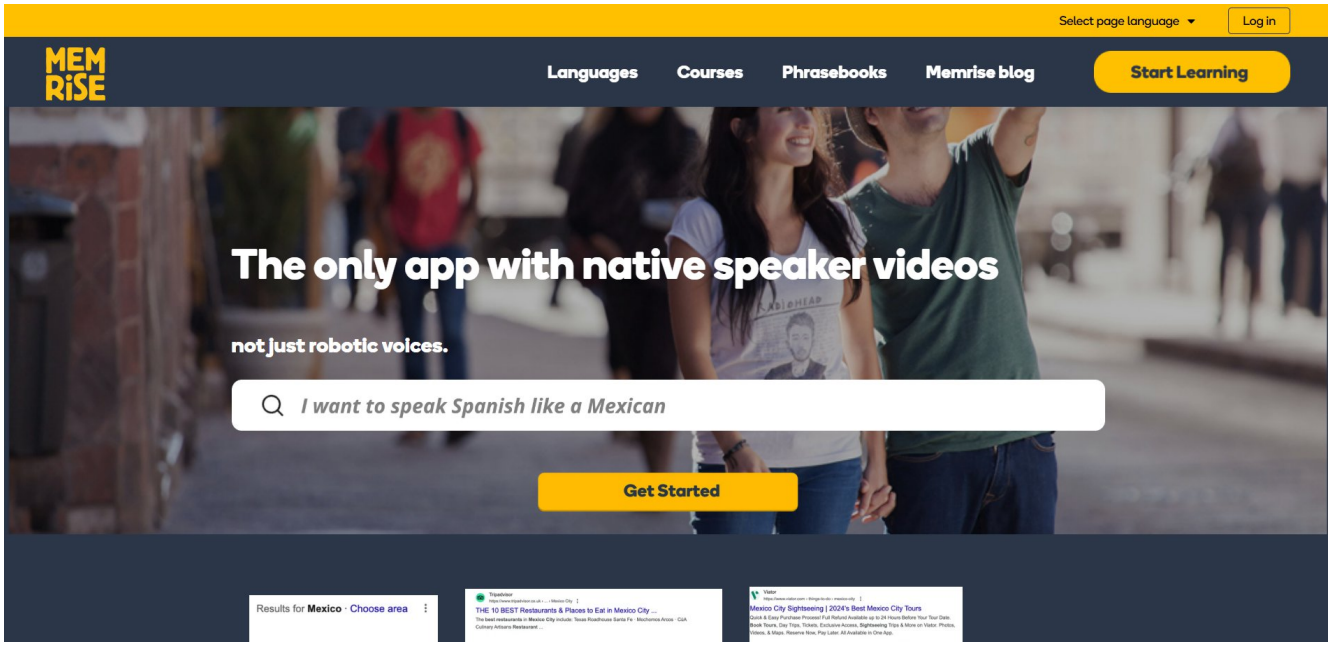

Рисунок 1.4 – Інтерфейс програмної системи “Memrise” (на основі даних [4])

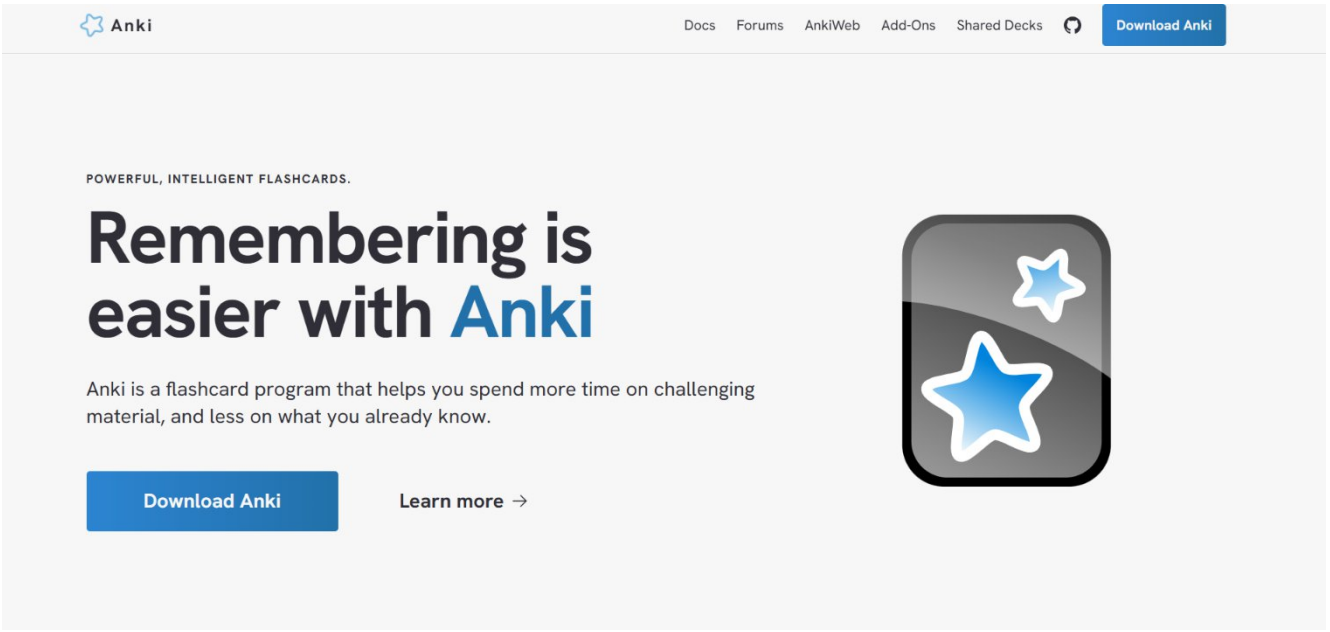

Рисунок 1.5 – Інтерфейс програмної системи “Anki” (на основі даних [5])

Ринок цифрових рішень для вивчення іноземних мов пропонує різноманітні платформи з унікальними підходами, але більшість з них обмежена в можливостях персоналізації навчання та створення власного контенту.

1.2 Виявлення та вирішення проблем

Сучасний ринок електронних засобів для вивчення іноземних мов охоплює велику кількість рішень, таких як Duolingo, Babbel, Rosetta Stone, Memrise, Anki та інші. Хоча вони забезпечують базовий рівень взаємодії та підтримують навчання, користувачі часто зіштовхуються з рядом проблем, які значно знижують ефективність або зручність використання таких платформ:

- недостатня кастомізація контенту – користувачі не мають змоги створювати власні теми, завдання чи курси. Платформи пропонують фіксовану програму, що не завжди відповідає індивідуальним цілям (наприклад, підготовка до роботи або спеціалізована лексика);
- обмежений набір типів завдань – більшість сервісів пропонують переважно прості вправи: вибір правильної відповіді, переклад одного речення, співставлення слів. Відсутні вправи з аудіювання, побудови речень, диктанти тощо;

- відсутність підтримки імпорту або синхронізації даних з інших платформ – користувачі, які вже мають словникові бази чи пройшли навчання на інших ресурсах, не можуть перенести результати в нову систему;
- відсутність інтегрованої допомоги або нагадувань – користувачі легко втрачають навчальну динаміку, оскільки більшість сервісів не надає нагадувань чи мотиваційних повідомлень, і тим більше – не підтримує ботів або інтелектуальних підказок.

Для візуалізації цієї проблематики проведено порівняльну оцінку (табл. 1), у якій співставлено ключові потреби користувачів та можливості популярних платформ.

Таблиця 1.1 – Порівняння можливостей популярних мовних сервісів і потреб користувачів (виконана самостійно)

Потреба / Платформа	Duolingo	Babbel	Rosetta Stone	Memrise	Anki
Кастомізація навчального контенту	-	-	-	+-	+
Різні типи завдань	+-	+-	+-	-	-
Імпорт/синхронізація	-	-	-	-	+
Нагадування / бот-помічник	+-	+-	-	-	+-

Сучасні платформи для вивчення мов мають ряд обмежень, які знижують їхню ефективність і зручність використання, включаючи недостатню кастомізацію контенту, обмежені типи завдань, відсутність підтримки імпорту даних та інтегрованої допомоги, що створює необхідність у розробці рішень, які задовольняють індивідуальні потреби користувачів і забезпечують більше можливостей для персоналізації навчального процесу.

1.3 Постановка задачі

Зважаючи на виявлені проблеми та потреби користувачів, метою даної розробки є створення інтерактивної платформи для вивчення іноземних мов, яка

дозволить максимально персоналізувати навчальний процес, зберігаючи при цьому зручність використання та високу мотивацію до навчання через гейміфікацію та різноманітність типів завдань.

1.3.1 Цільова аудиторія

Цільовою аудиторією для даного проєкту є:

- студенти, які прагнуть вивчити іноземну мову для особистого розвитку, професійних цілей або подорожей;
- працівники, яким необхідно покращити свої мовні навички для кар'єрного росту;
- вчителі та репетитори, які хочуть забезпечити своїм учням індивідуалізовану платформу для навчання;
- люди, які вивчають мови для специфічних цілей (наприклад, професійна лексика, терміни, специфіка роботи в іншій країні).

1.3.2 Монетизація

Монетизація платформи може здійснюватися за допомогою наступних механізмів:

- підписка: користувачі можуть отримати доступ до преміум-функцій, таких як персоналізовані плани навчання, розширені типи завдань та додаткові ресурси, через підписку на місяць або рік;
- реклама: безкоштовні користувачі можуть мати доступ до основних функцій платформи з підтримкою реклами;
- інтеграція з іншими сервісами: можливість імпортувати курси або матеріали з інших платформ (наприклад, LinkedIn Learning, Coursera), що дозволить монетизувати контент через партнерства.

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

2.1 Функціональні вимоги

Функціональні вимоги визначають основні можливості програмного продукту, які забезпечують задоволення потреб користувачів і реалізацію основних цілей системи [6]. У розробленому веб-застосунку для інтерактивного вивчення іноземних мов ці вимоги стосуються таких ключових аспектів, як реєстрація користувачів, персоналізація навчального процесу [7], наявність різноманітних типів завдань, можливості інтерактивного навчання через гейміфікацію, а також підтримка імпорту даних та нагадувань. Основні функціональні вимоги:

- реєстрація та аутентифікація користувачів: користувачі повинні мат можливість реєструватися на платформі за допомогою електронної пошти;
- персоналізація навчального матеріалу: користувач може обирати типи завдань. Користувач може вибирати теми для вивчення (наприклад, професійна лексика, подорожі, бізнес тощо). Система повинна адаптувати складність завдань залежно від прогресу користувача;
- типи завдань: Платформа повинна підтримувати різноманітні типи завдань, такі як: переклад речень та слів, побудова правильних речень, аудіовправи (прослуховування та вибір правильного варіанту), ігрові завдання (matching words, fill in the blanks);
- гейміфікація та мотивація: Реалізація системи досягнень, балів та лідербордів. Користувачі повинні мати можливість змагатися з іншими учасниками та бачити свій прогрес у порівнянні з іншими;
- імпорт та синхронізація даних: Платформа повинна дозволяти імпорт даних;
- нагадування та підказки: Користувачі можуть налаштовувати нагадування для виконання завдань. Боти повинні надсилати мотиваційні повідомлення та нагадування для виконання завдань.

2.2 Нефункціональні вимоги

Для забезпечення високої якості системи для інтерактивного вивчення іноземних мов, необхідно врахувати різноманітні нефункціональні вимоги. Це дозволить забезпечити стабільну роботу платформи при високих навантаженнях, забезпечити захист даних користувачів та можливість подальшого масштабування системи, а також надати зручний і інтуїтивно зрозумілий інтерфейс, який буде адаптований до різних пристроїв і умов використання. Основні нефункціональні вимоги:

- продуктивність: система повинна підтримувати одночасну роботу до 1000 користувачів без значних затримок. Час відповіді сервера не повинен перевищувати 2 секунд для основних операцій;
- безпека: всі особисті дані користувачів повинні зберігатися в зашифрованому вигляді. Система повинна підтримувати двофакторну аутентифікацію для підвищення рівня безпеки;
- масштабованість: система повинна бути здатною обробляти великий обсяг даних, оскільки з часом кількість користувачів та обсяг навчального контенту зростатимуть. Можливість розширення функціоналу в майбутньому (наприклад, підтримка нових мов або типів завдань);
- портативність: веб-версія повинна бути адаптивною для використання на різних пристроях (комп'ютери, планшети, смартфони);
- інтерфейс користувача: інтерфейс повинен бути інтуїтивно зрозумілим і зручним для користувачів усіх рівнів;
- підтримка мов: спочатку підтримка лише основних мов для вивчення, але з можливістю розширення до більшої кількості мов у майбутньому.

3 АРХІТЕКТУРА ТА ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 UML проектування програмного забезпечення

UML (Unified Modeling Language) є стандартом для візуального моделювання програмних систем. Для проектування системи для вивчення іноземних мов було використано різні діаграми UML, щоб відобразити структуру, поведінку та взаємодію компонентів. Це допомагає створити чітке уявлення про систему, що полегшує комунікацію серед команди розробників, а також покращує підтримуваність і розширюваність системи.

Діаграма прецедентів (Use-case діаграма) (див. рис. 3.1) відображає взаємодію користувачів із системою. Вона допомагає визначити, які функції має виконувати система з точки зору користувача, і як ці функції збудовані у вигляді взаємодій.

Діаграма компонентів (див. рис. 3.2) описує компоненти системи та їх взаємодії. Це дозволяє зрозуміти, як система розподіляється на окремі частини, що виконують різні функції, і як ці частини взаємодіють між собою.

Основні компоненти системи:

- frontend: клієнтська частина, що відповідає за відображення інтерфейсу користувача;
- backend: серверна частина, що обробляє запити від користувача, виконує бізнес-логіку та взаємодіє з базою даних;
- database: система зберігання даних (MSSQL), де зберігаються дані користувачів, завдань, результатів та досягнень.

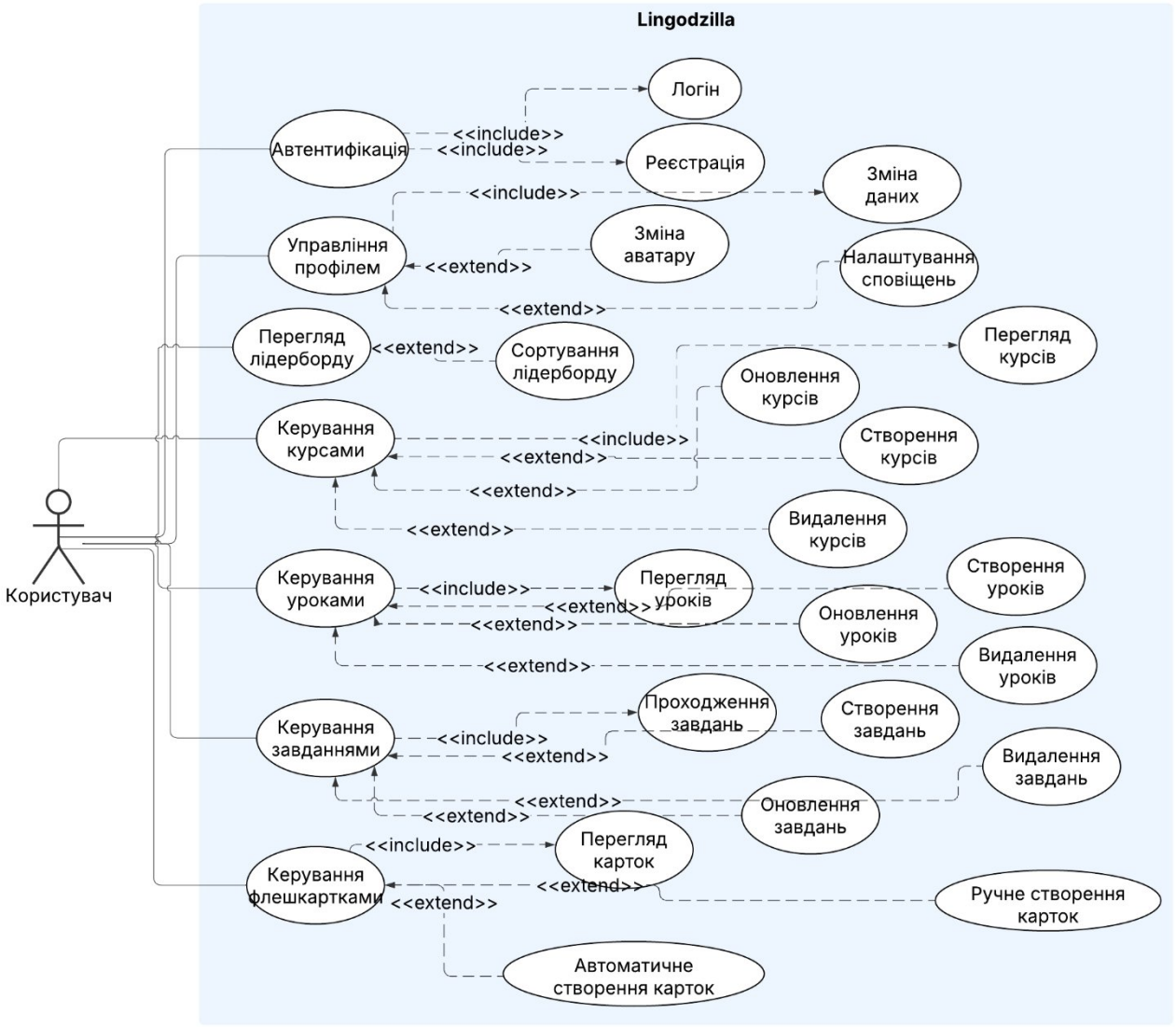

Рисунок 3.1 – Use-case діаграма застосунку (рисунок виконано самостійно)

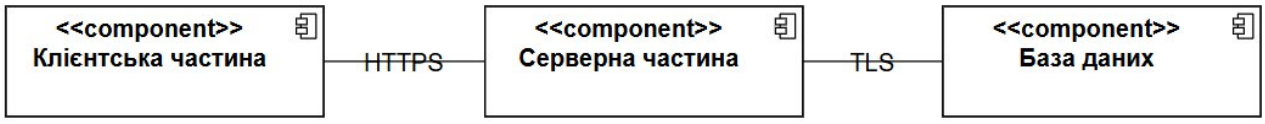

Рисунок 3.2 – Діаграма компонентів (рисунок виконано самостійно)

3.2 Проектування архітектури програмного забезпечення

3.2.1 Загальна архітектура програмного забезпечення

Загальна архітектура програмного забезпечення для інтерактивної платформи вивчення іноземних мов побудована з урахуванням принципів модульності, масштабованості та високої продуктивності. Основними компонентами архітектури є клієнтська частина та серверна частина, які взаємодіють через API [8] для забезпечення безперебійної роботи системи.

Клієнтська частина реалізована як веб-застосунок, розроблений на платформі Angular. Він взаємодіє з серверною частиною через REST API. Веб-інтерфейс надає користувачам доступ до навчальних матеріалів, прогресу, завдань, досягнень та елементів гейміфікації.

Серверна частина побудована з використанням Onion Architecture, яка дозволяє чітко розділяти логіку програми на шари для забезпечення масштабованості та підтримуваності системи. Архітектура включає:

- core (ядро): містить бізнес-логіку та сутності домену, незалежні від технологій і зовнішніх інтерфейсів;
- application layer: відповідає за реалізацію сервісів та операцій, таких як обробка даних, виконання бізнес-правил та взаємодія з репозиторіями;
- infrastructure layer: забезпечує доступ до бази даних, зовнішніх API, а також інші технічні аспекти взаємодії з зовнішнім світом;

- presentation layer: відповідає за прийом запитів від користувачів і відправку відповідей через API. Веб-інтерфейс взаємодіє з цією частиною за допомогою REST API.

Для зберігання даних про користувачів, завдання, результати та досягнення використовується MSSQL.

Для забезпечення високої доступності та масштабованості використовується Microsoft Azure. Хмарна платформа надає:

- автоматичне масштабування в залежності від навантаження;
- безпечне зберігання даних;
- інтеграцію з різними сервісами для моніторингу та аналітики.

Для забезпечення стабільності та портативності програмного забезпечення в різних середовищах використовуються Docker-контейнери. Вони дозволяють швидко розгортати та тестувати різні версії програмного продукту, а також спрощують масштабування сервісів в хмарному середовищі.

Всі компоненти взаємодіють через REST API, що дозволяє забезпечити незалежність між ними та дозволяє гнучко розвивати кожен з компонентів системи (веб-застосунок, сервер) без значних змін в інших частинах системи.

Ця архітектура дозволяє побудувати гнучку, масштабовану і ефективну платформу для вивчення мов, яка буде здатна працювати з великими обсягами даних і підтримувати велику кількість користувачів.

3.2.2 Архітектура серверної частини

Серверна частина системи для інтерактивного вивчення іноземних мов реалізована з використанням Onion Architecture, що дозволяє створити чисту і підтримувану структуру, що легко масштабується та інтегрується з іншими системами. Вона чітко розділяє різні рівні логіки і забезпечує високий рівень ізоляції та гнучкості в роботі з даними, що робить систему стійкою до змін у майбутньому.

Шари архітектури:

- core (ядро): це основний шар, який містить доменні моделі (сущності) та бізнес-логіку програми. Він не залежить від зовнішніх технологій чи інтерфейсів і є незалежним від інших шарів архітектури. Основні компоненти: моделі для користувачів, завдань, досягнень, статистики прогресу;
- application layer (шар застосунків): цей шар відповідає за організацію бізнес-операцій, таких як обробка запитів від користувачів, виконання бізнес-правил і комунікація між різними частинами системи. Він надає інтерфейси для роботи з даними та обробки запитів. Основні компоненти: сервіси для роботи з користувачами (реєстрація, авторизація), завданнями (формування та перевірка завдань), звітністю (моніторинг прогресу), а також реалізація механізмів гейміфікації (лідерборди, досягнення);
- infrastructure layer (шар інфраструктури): цей шар відповідає за реалізацію конкретних технологій і зовнішніх інтерфейсів, таких як бази даних, файлові системи, зовнішні API (наприклад, для перекладів або роботи з текстами), а також інші інтеграції. Основні компоненти: репозиторії для збереження даних у базі даних (MSSQL), інтеграція з зовнішніми сервісами для отримання лексичних даних або тестів;
- presentation layer (шар презентації): цей шар відповідає за прийом запитів від користувачів через веб-застосунок і надання відповідей. Він забезпечує взаємодію з клієнтською частиною системи через REST API, формує відповіді у вигляді JSON або іншого зручного формату. Основні компоненти: контролери, що відповідають за обробку HTTP-запитів, валідатори вхідних даних, а також логіка для формування відповідей для фронтенду.

Взаємодія компонентів:

- presentation layer (шар презентації) взаємодіє з Application layer (шар застосунків), передаючи запити користувача, такі як реєстрація, авторизація, вибір завдань, збереження результатів тощо;
- application layer використовує Infrastructure layer для доступу до бази даних, зовнішніх API та інших сервісів, необхідних для виконання операцій;
- core layer є основою, яка забезпечує бізнес-логіку та визначає, як обробляються дані. Інші шари взаємодіють з ним через сервіси та інтерфейси.

Переваги такої архітектури:

- модульність: Кожен шар має чітко визначену відповідальність і може бути змінений або розширений без великого впливу на інші частини системи;
- масштабованість: Завдяки чіткому поділу на шари система легко масштабується — можна окремо працювати з кожним компонентом;
- тестування: Легкість тестування кожного шару окремо, оскільки вони ізольовані від зовнішніх залежностей.

Ця архітектура дозволяє зберігати чистоту коду, забезпечує високу підтримуваність і гнучкість для майбутніх змін, а також дозволяє знижувати складність розробки завдяки чітко визначеним межам відповідальності кожного шару.

3.2.3 Архітектура клієнтської частини

Клієнтська частина системи для вивчення іноземних мов розроблялась з використанням Angular [9], що є потужним фреймворком для створення односторінкових веб-додатків (SPA). Архітектура клієнтської частини побудована

таким чином, щоб забезпечити високу продуктивність, зручність для користувача та масштабованість додатку.

Компоненти є основними будівельними блоками в Angular додатку. Вони відповідають за відображення інтерфейсу користувача та взаємодію з ним. Кожен компонент містить:

- HTML шаблон для розмітки,
- CSS для стилізації,
- TypeScript для логіки компонента.

Сервіси відповідають за бізнес-логіку додатку і взаємодію з сервером через HTTP запити. Вони дозволяють зберігати централізовану логіку і зручну взаємодію з API.

Для навігації між різними частинами додатку використовується Angular Router. Це дозволяє створювати багато сторінок в одному веб-додатку без необхідності перезавантажувати сторінку.

Для покращення безпеки клієнтської частини застосовуються стандартні підходи безпеки веб-додатків, включаючи захист від CSRF атак, використання HTTPS для всіх запитів і налаштування правильних політик CORS (Cross-Origin Resource Sharing).

3.3 Проектування структури бази даних

Проектування структури бази даних для системи вивчення іноземних мов передбачає створення ефективної та масштабованої моделі для зберігання даних користувачів, курсів, уроків, вправ, прогресу та інших важливих аспектів. База даних була розроблена на основі Microsoft SQL Server (MSSQL) [10], з використанням Entity Framework Core (EF Core) для взаємодії з даними.

Основними сутностями системи є:

- користувачі (Users): містить основну інформацію про користувачів, їхній рівень знань та дані для аутентифікації;
- запити до друзів (FriendRequests): зберігає інформацію про запити на додавання друзів між користувачами;

- вподобання користувача (UserPreferences): зберігає дані про вибір курсів та рівні складності, що подобаються користувачам;
- курси (Courses): містить дані про доступні курси для користувачів;
- уроки (Lessons): зберігає інформацію про уроки, які є частиною курсів;
- вправи (Exercises): містить дані про вправи, що користувачі виконують для закріплення знань;
- прогрес (Progress): зберігає інформацію про прогрес користувачів у виконанні завдань та уроків.

Між сутностями встановлені взаємозв'язки:

- користувачі можуть мати запити до друзів та вподобання;
- курси містять кілька уроків, а уроки, в свою чергу, мають вправи;
- користувачі відслідковують свій прогрес у виконанні уроків та вправ.

Для забезпечення безпеки даних використовуються методи хешування паролів та захищені з'єднання між клієнтом і сервером. Всі чутливі дані, такі як паролі користувачів, зберігаються у зашифрованому вигляді. На рисунку 3.3 наведена ER-діаграма бази даних.

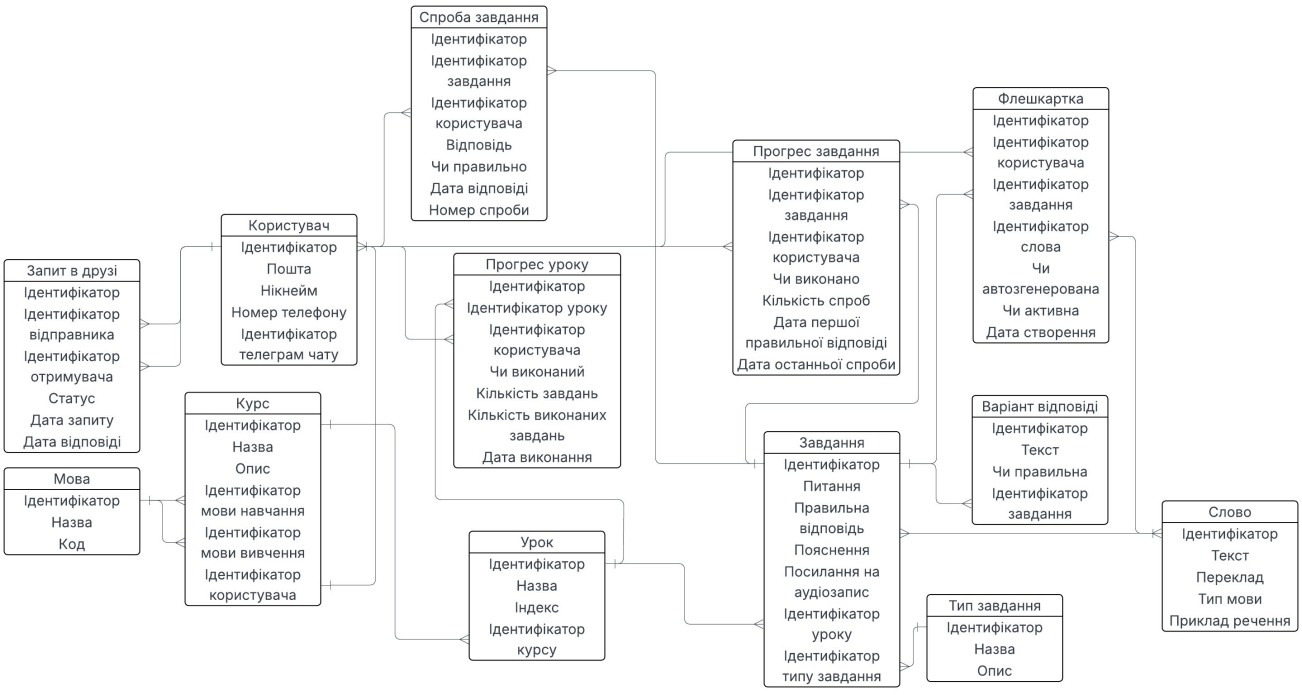

Рисунок 3.3 – ER-діаграма (рисунок виконано самостійно)

3.4 Створення UI/UX дизайну

Перша сторінка, з якою зустрічається користувач після відкриття застосунку – це сторінка для входу (див. рис. 3.4) та реєстрації (див. рис. 3.5). Цей екран має два основні блоки: форма логіну та форма реєстрації. Використано просту форму з великими полями для введення даних, щоб забезпечити зручність використання як на мобільних пристроях, так і на десктопах.

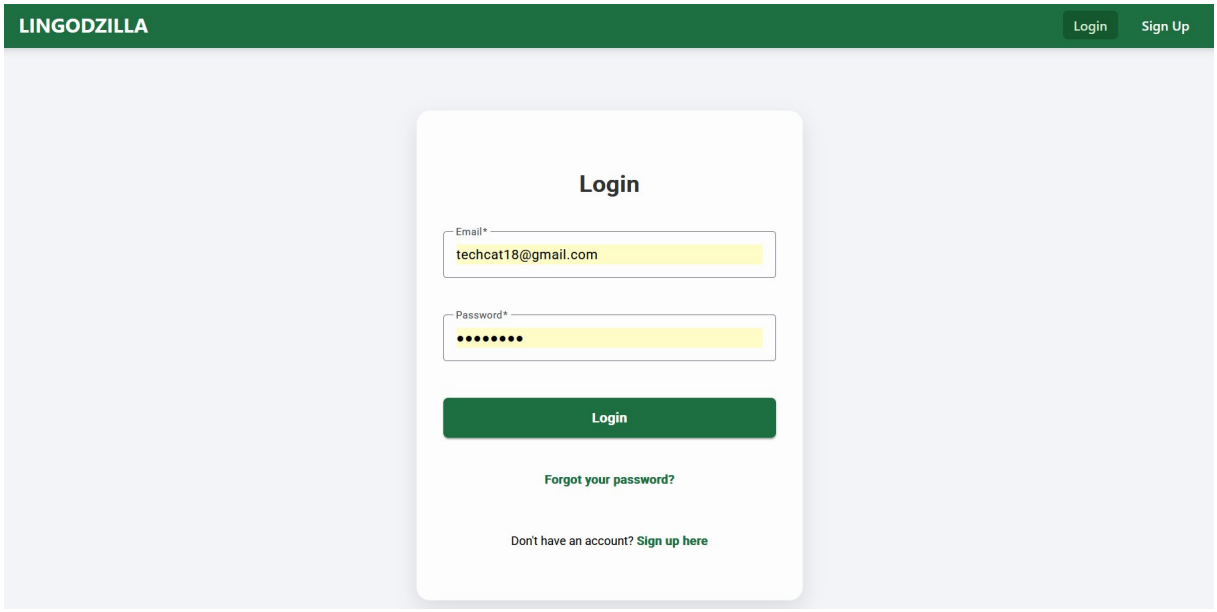

Рисунок 3.4 – Сторінка для входу (рисунок виконано самостійно)

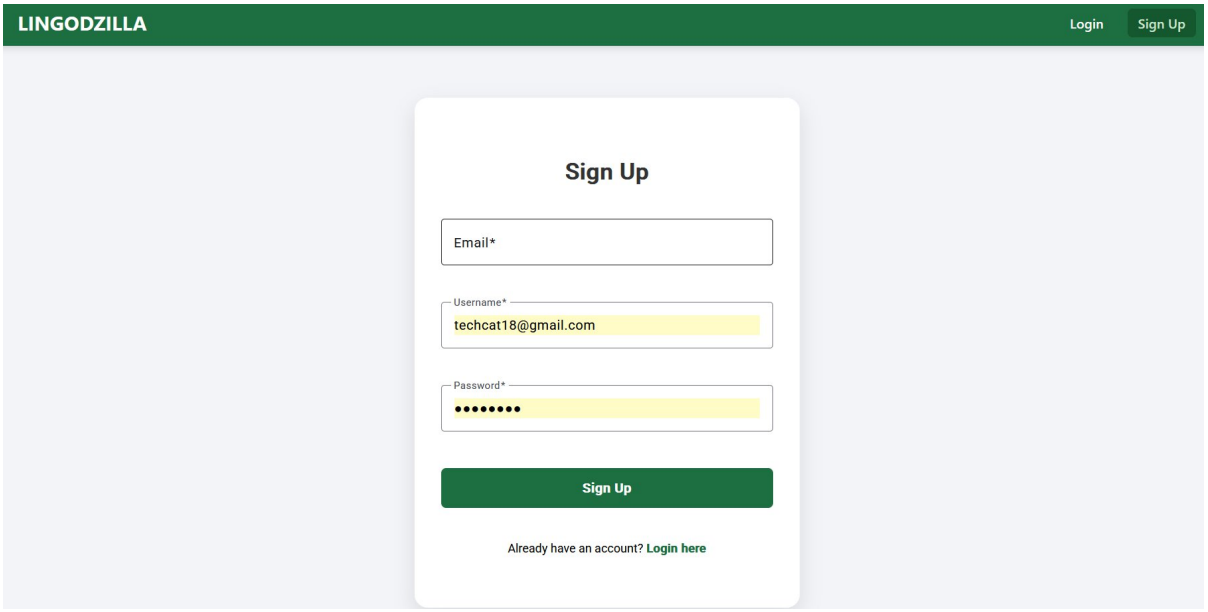

Рисунок 3.5 – Сторінка реєстрації (рисунок виконано самостійно)

Після входу користувач потрапляє на сторінку профілю (див. рис. 3.6), де можна переглянути основну інформацію про себе.

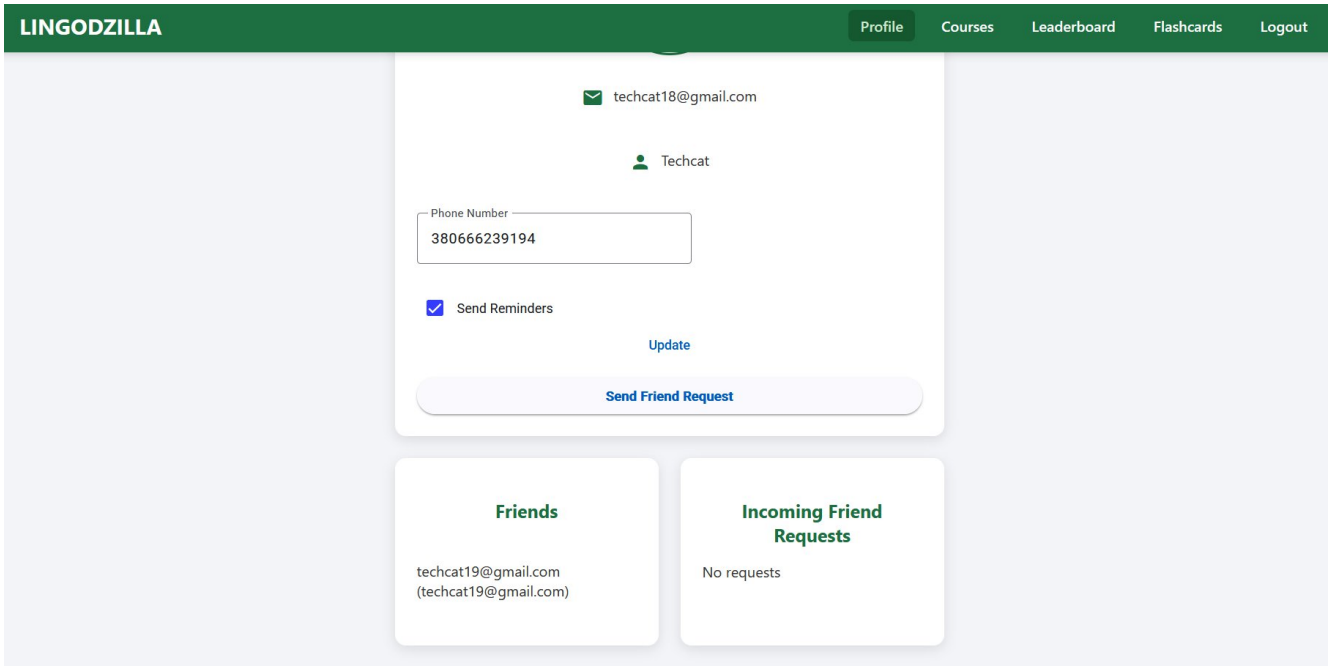

Рисунок 3.6 – Сторінка профілю (рисунок виконано самостійно)

На сторінці Список курсів (див. рис. 3.7) користувач може вибрати курс для подальшого вивчення. Інтерфейс адаптований для мобільних пристроїв, де картки курсів автоматично переставляються в стовпчики в залежності від ширини екрану.

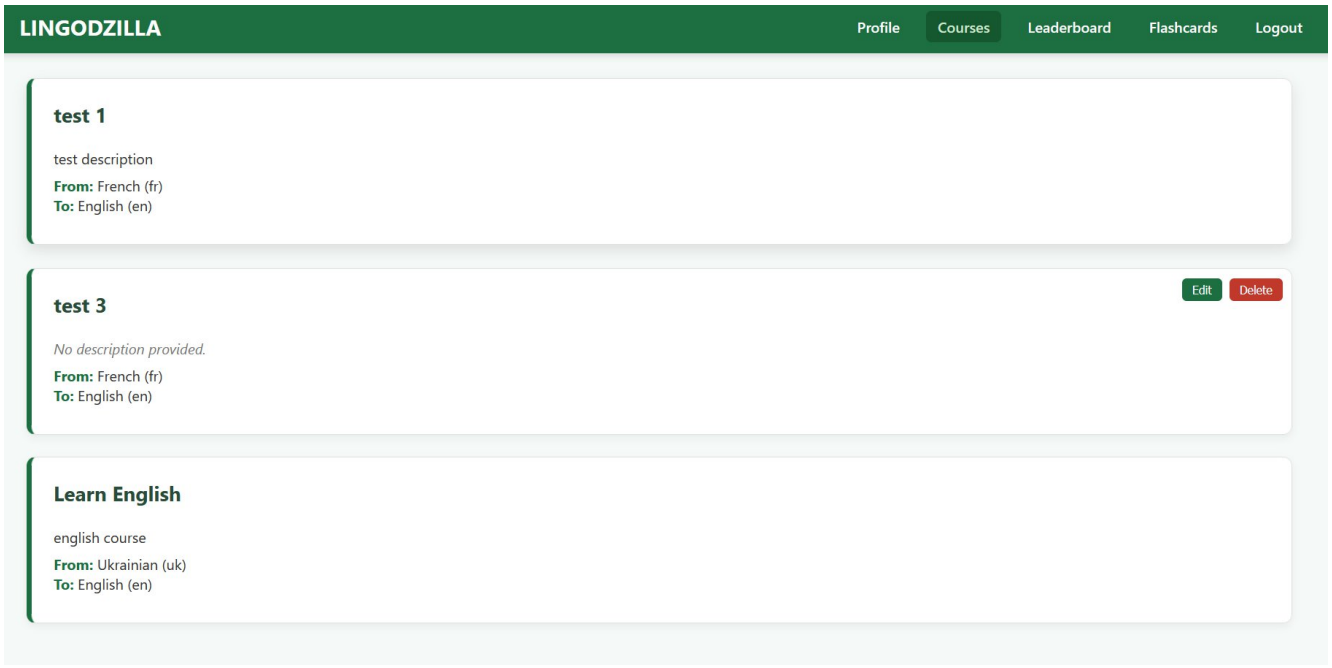

Рисунок 3.7 – Сторінка курсів (рисунок виконано самостійно)

На сторінці Карток (див. рис. 3.8) відображаються флеш-картки для запам'ятовування нових слів та виразів. Використано анімації для зворотного зв'язку при перевірці правильності відповіді, що підвищує інтерактивність.

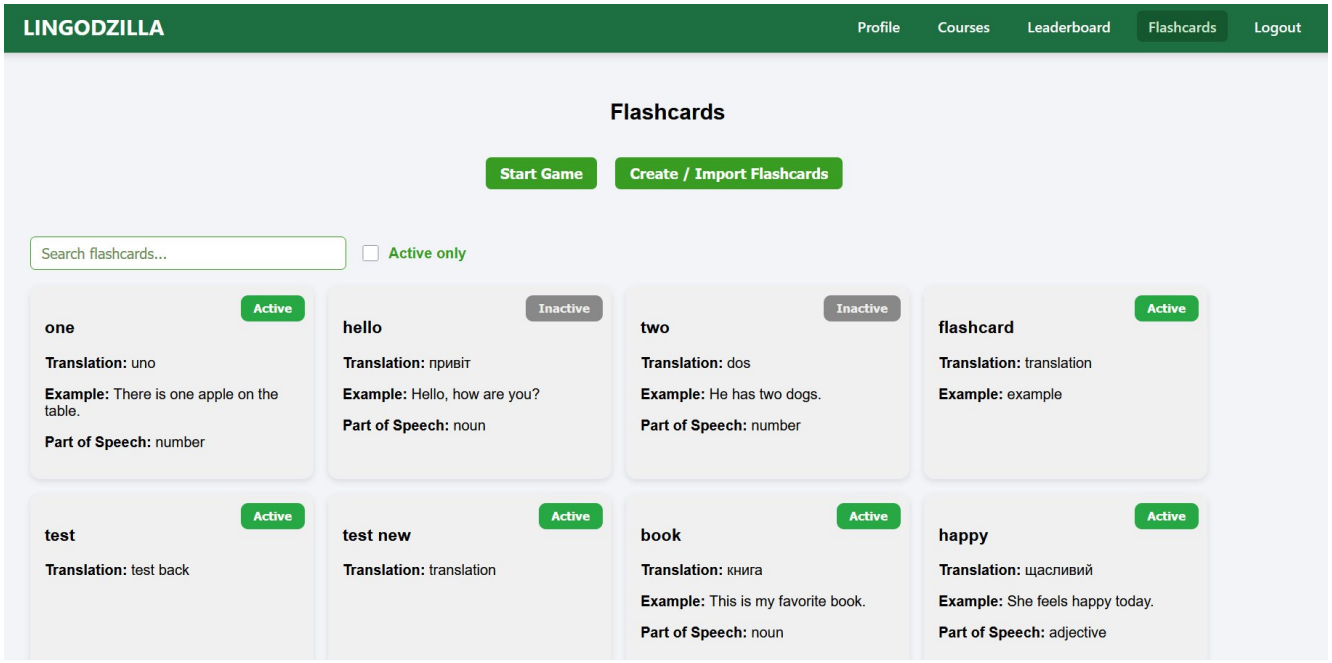

Рисунок 3.8 – Сторінка карток (рисунок виконано самостійно)

Лідерборд (таблиця лідерів) (див. рис. 3.9) показує рейтинг користувачів за кількістю отриманих балів, досягнень чи завершених завдань. Ця сторінка дозволяє користувачам порівнювати свій прогрес з іншими.

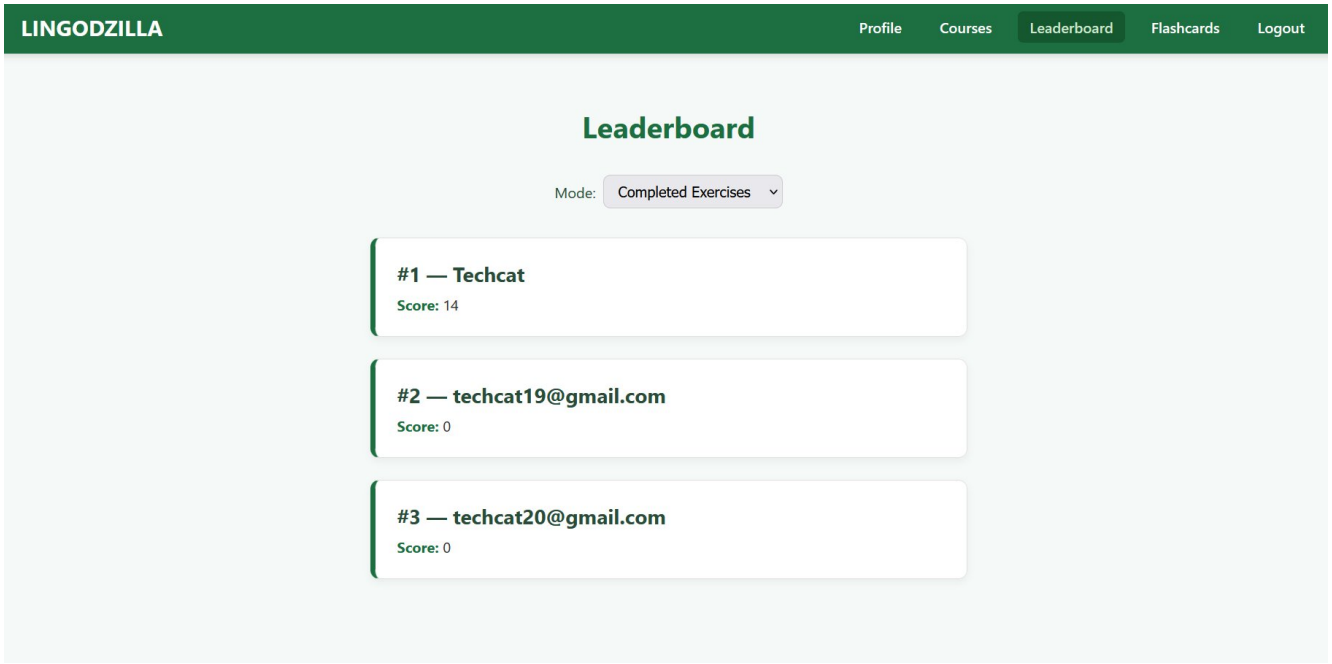

Рисунок 3.9 – Сторінка лідерборду (рисунок виконано самостійно)

Сторінка Уроків (див. рис. 3.10) є однією з основних частин системи, де користувач може пройти уроки відповідно до обраного курсу.

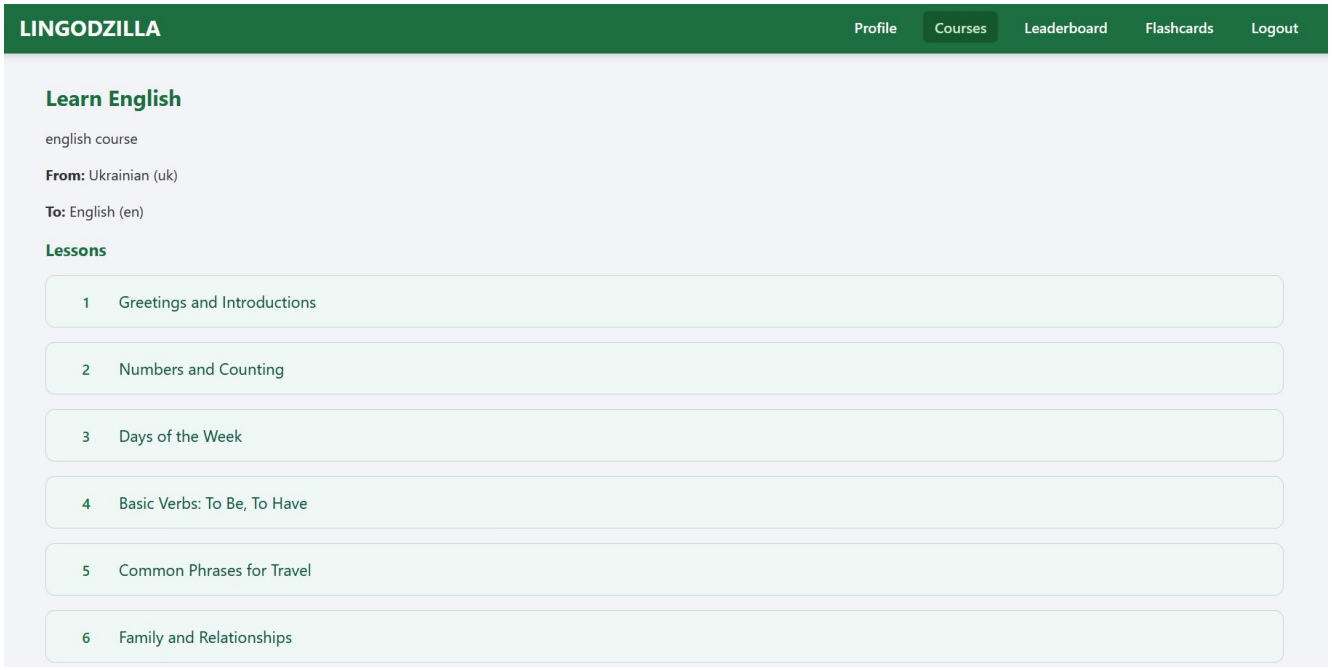

Рисунок 3.10 – Сторінка уроків (рисунок виконано самостійно)

Сторінка Завдання (див. рис. 3.11) містить список завдань, які користувач повинен виконати. Завдання можуть бути різних типів: текстові, аудіо, переклад, побудова речень тощо.

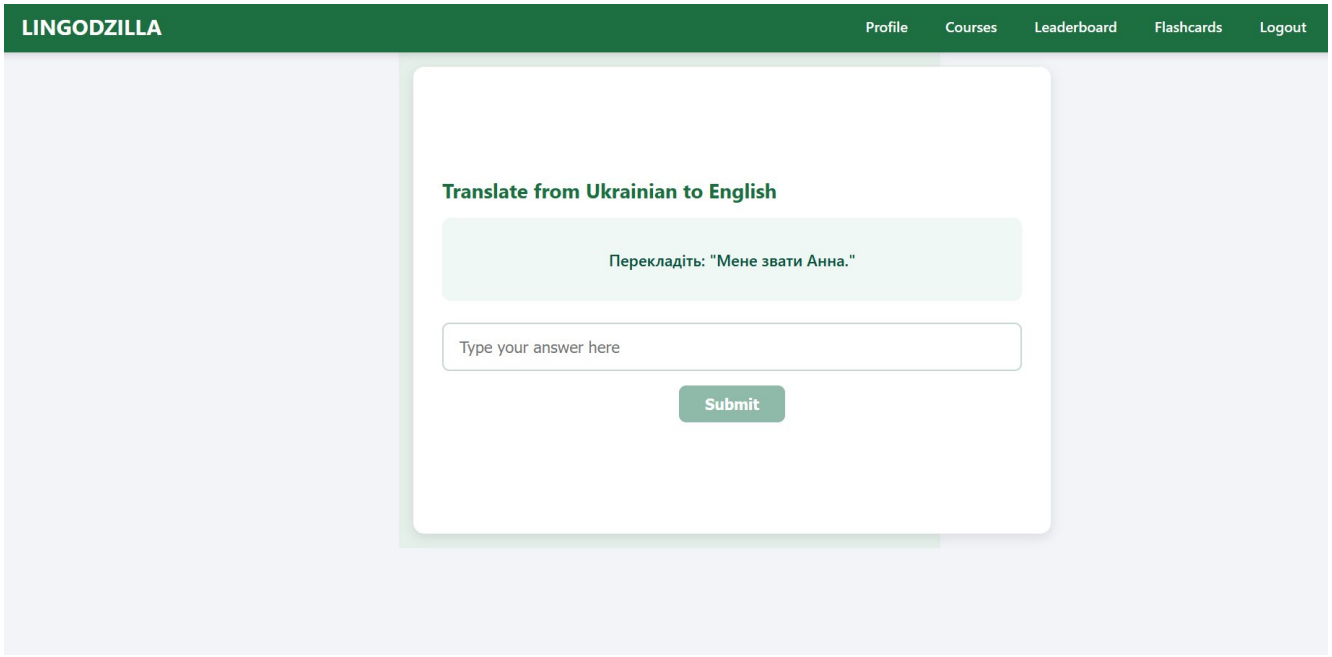

Рисунок 3.11 – Сторінка завдання (рисунок виконано самостійно)

Процес створення UI/UX дизайну для системи вивчення іноземних мов базувався на принципах зручності для користувача та оптимізації взаємодії з додатком. Метою було створити інтуїтивно зрозумілий інтерфейс, який забезпечить приємний досвід користувача на кожному етапі взаємодії з системою. Для цього було використано сучасні підходи до дизайну, зокрема адаптивний дизайн для підтримки різних розмірів екранів та пристроїв.

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

4.1 Робота з базою даних

Для зберігання даних у розробленій системі для вивчення іноземних мов використовувалась реляційна база даних MSSQL. Це забезпечує високу надійність і ефективне зберігання даних, таких як інформація про користувачів, завдання, результати виконання завдань, досягнення, та інші компоненти платформи.

Серверна частина системи використовує Entity Framework Core (EF Core) як ORM (Object-Relational Mapper), що дозволяє легко взаємодіяти з базою даних через об'єктно-орієнтовані моделі. EF Core автоматично генерує SQL-запити на основі LINQ-запитів, що дозволяє працювати з базою даних за допомогою звичних об'єктів, не пишучи складні SQL-запити вручну.

Для організації та зручної роботи з базою даних я створив окремий SQL Database Project у Visual Studio (див. рис. 4.1). Це дозволило зберігати всі скрипти для створення та модифікації бази даних у вигляді версійованих файлів, що спрощує керування змінами бази даних під час розробки. Проєкт забезпечує можливість автоматичного створення та оновлення бази даних при кожному розгортанні.

У проекті були створені скрипти для створення таблиць. Приклад скрипту:

```
CREATE TABLE [dbo].[Exercises]
(
    [Id] UNIQUEIDENTIFIER PRIMARY KEY DEFAULT NEWID(),
    [Question] NVARCHAR(1000) NULL,
    [AudioUrl] NVARCHAR(1000) NULL,
    [CorrectAnswer] NVARCHAR(MAX) NOT NULL,
    [Explanation] NVARCHAR(MAX) NULL,

    [LessonId] UNIQUEIDENTIFIER NOT NULL,
    [ExerciseTypeId] UNIQUEIDENTIFIER NOT NULL,

   CONSTRAINT FK_Exercises_Lessons FOREIGN KEY ([LessonId]) REFERENCES
[dbo].[Lessons]([Id]),
    CONSTRAINT FK_Exercises_ExerciseTypes FOREIGN KEY
([ExerciseTypeId]) REFERENCES [dbo].[ExerciseTypes]([Id])
);
```

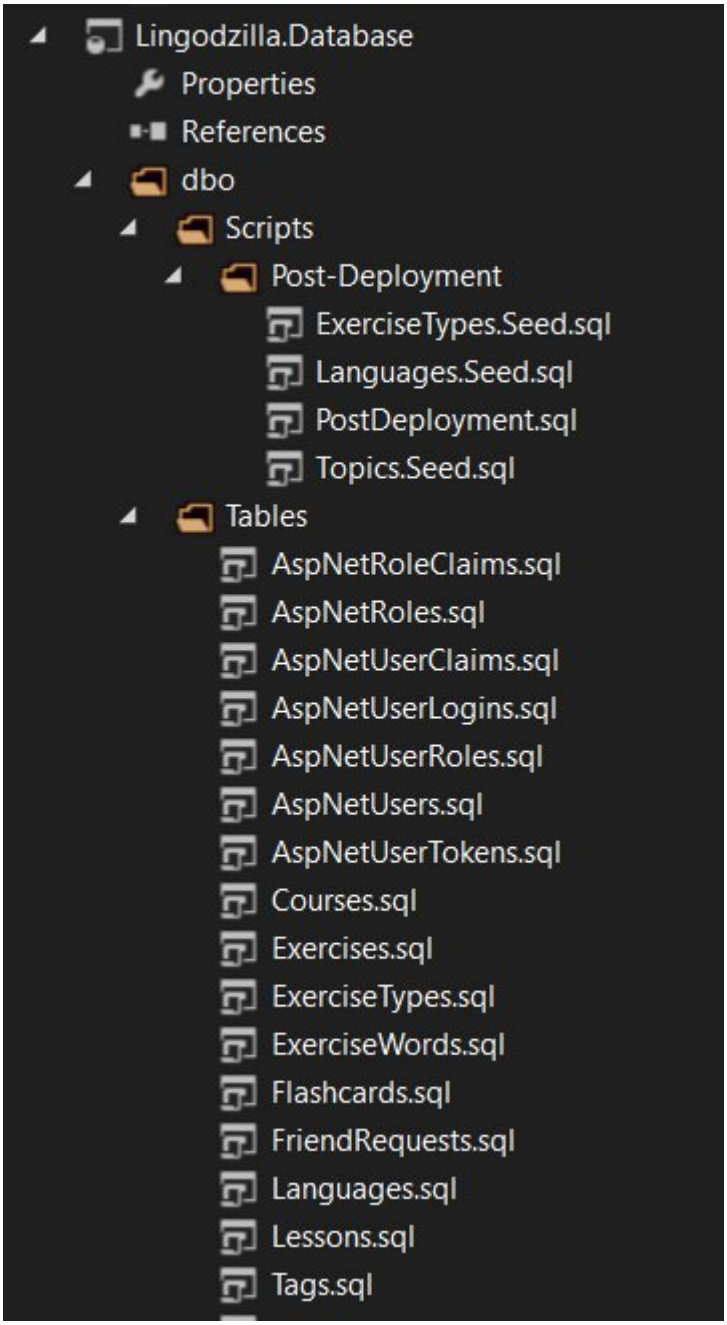

Рисунок 4.1 – SQL Database Project (рисунок виконано самостійно)

Також були створені скрипти, що запускаються одразу після деплою бази даних. Приклад скрипту:

```
    INSERT INTO [dbo].[ExerciseTypes] ([Id], [Name], [Description])
    SELECT NEWID(), 'Multiple Choice', 'Choose the correct answer from several options.'
    WHERE NOT EXISTS (SELECT 1 FROM [dbo].[ExerciseTypes] WHERE [Name] = 'Multiple Choice');

    INSERT INTO [dbo].[ExerciseTypes] ([Id], [Name], [Description])
    SELECT NEWID(), 'Fill in the Blank', 'Complete the sentence by entering the missing word or phrase.'
```

```
    WHERE NOT EXISTS (SELECT 1 FROM [dbo].[ExerciseTypes] WHERE [Name] =
'Fill in the Blank');

    INSERT INTO [dbo].[ExerciseTypes] ([Id], [Name], [Description])
    SELECT NEWID(), 'Matching', 'Match words or phrases with their
corresponding translations or images.'
    WHERE NOT EXISTS (SELECT 1 FROM [dbo].[ExerciseTypes] WHERE [Name] =
'Matching');

    INSERT INTO [dbo].[ExerciseTypes] ([Id], [Name], [Description])
    SELECT NEWID(), 'Speaking Practice', 'Practice pronunciation by
repeating or constructing sentences.'
    WHERE NOT EXISTS (SELECT 1 FROM [dbo].[ExerciseTypes] WHERE [Name] =
'Speaking Practice');

    INSERT INTO [dbo].[ExerciseTypes] ([Id], [Name], [Description])
    SELECT NEWID(), 'Listening Comprehension', 'Listen to audio and answer
questions about what you hear.'
    WHERE NOT EXISTS (SELECT 1 FROM [dbo].[ExerciseTypes] WHERE [Name] =
'Listening Comprehension');

    INSERT INTO [dbo].[ExerciseTypes] ([Id], [Name], [Description])
    SELECT NEWID(), 'Translation', 'Translate a sentence or phrase into
another language.'
    WHERE NOT EXISTS (SELECT 1 FROM [dbo].[ExerciseTypes] WHERE [Name] =
'Translation');
```

4.2 Валідація на сервері

Для валідації даних на серверній стороні системи для вивчення іноземних мов використано бібліотеку Fluent Validation. Цей інструмент дозволяє реалізувати зручну та гнучку валідацію моделей та даних, забезпечуючи високу якість перевірки вхідних даних від користувачів та зниження ймовірності помилок під час обробки.

Fluent Validation дозволяє визначати правила валідації через об’єкти, що описують бізнес-логіку. Це дозволяє не тільки перевіряти типові умови, такі як довжина рядків або формат email-адрес, а й створювати складніші правила валідації, орієнтуючись на специфічні вимоги системи.

Коли користувач надсилає дані (наприклад, під час реєстрації або зміни паролю), дані перевіряються через відповідні валідатори перед тим, як вони потраплять до бізнес-логіки чи бази даних. Це дозволяє відразу відсіяти некоректні або неповні дані, зменшуючи навантаження на сервер та підвищуючи безпеку системи.

У разі помилок валідації система повертає відповідне повідомлення користувачу, що дозволяє йому виправити помилки вводу. Наприклад, якщо користувач введе некоректний email або пароль, система повідомить про це, що покращує користувацький досвід.

4.3 Валідація користувацького введення

Валідація користувацького введення є важливим етапом для забезпечення правильності та безпеки даних, які надходять від користувача через веб-інтерфейс. На відміну від валідації на сервері, яка перевіряє дані після їх надсилання до сервера, валідація на клієнтській стороні дозволяє швидко виявити помилки на етапі введення даних, зменшуючи навантаження на сервер та покращуючи користувацький досвід.

Для валідації користувацького введення на клієнтській стороні використовується JavaScript-бібліотека Angular Reactive Forms. Цей підхід дозволяє зручно перевіряти введені дані, не надсилаючи їх на сервер до того, як вони будуть перевірені. Angular надає потужні можливості для роботи з формами, дозволяючи перевіряти їх на правильність через вбудовані валідатори та кастомні функції перевірки.

Хоча валідація на клієнтській стороні покращує користувацький досвід, важливо, щоб ці дані також перевірялись на сервері для забезпечення їхньої цілісності та безпеки. Тому серверна валідація завжди є необхідною, навіть якщо дані були перевірені на клієнті.

4.4 Авторизація

Для авторизації користувачів у системі було реалізовано механізм JWT (JSON Web Tokens). Це дозволяє забезпечити безпечний процес аутентифікації та авторизації, зокрема для взаємодії з сервером через API.

JWT є стандартом для обміну даними, який дозволяє безпечне передавання інформації між клієнтом і сервером у вигляді JSON-об'єкта. Токен містить

зашифровану інформацію про користувача, що дозволяє серверу перевірити його права доступу до захищених ресурсів. Основними складовими JWT є:

- header: вказує тип токену (звичайно "JWT") і алгоритм підпису (наприклад, HMAC SHA256 або RSA);
- payload: містить заявлені дані (наприклад, ідентифікатор користувача та роль);
- signature: використовується для перевірки того, що токен не був змінений.

Після успішної авторизації користувача система генерує JWT, який містить інформацію про користувача. Цей токен надсилається назад клієнту, який використовує його для подальших запитів до серверу.

Токен зберігається на клієнтській стороні в localStorage, і додається до кожного запиту в заголовок Authorization при доступі до захищених ендпоїнтів.

При отриманні запиту з JWT, сервер перевіряє токен на валідність і достовірність за допомогою секретного. Якщо токен дійсний, користувач отримує доступ до запитуваних ресурсів. Реалізація JWT сервісу:

```
public class JwtService : IJwtService
{
    private readonly JwtSettings _jwtSettings;
    private readonly UserManager<User> _userManager;

    public JwtService(
        IOptions<JwtSettings> jwtSettings,
        UserManager<User> userManager)
    {
        _jwtSettings = jwtSettings.Value;
        _userManager = userManager;
    }

    public SigningCredentials GetSigningCredentials()
    {
        var key = Encoding.UTF8.GetBytes(_jwtSettings.Key);
        var secret = new SymmetricSecurityKey(key);

        return new SigningCredentials(secret,
SecurityAlgorithms.HmacSha256);
    }

    public async Task<List<Claim>> GetClaimsAsync(string userId)
    {
        var user = await _userManager.FindByIdAsync(userId);
        if (user is null)
        {
```

```
            throw new ApplicationException("User was not found");
        }

        var claims = new List<Claim>
        {
            new(JwtRegisteredClaimNames.Sub, user.Id.ToString()),
            new(JwtRegisteredClaimNames.Email, user.Email ??
string.Empty),
            new(ClaimTypes.Role, "User")
        };

        var userRoles = await _userManager.GetRolesAsync(user);

        claims.AddRange(userRoles.Select(role => new
Claim(ClaimsIdentity.DefaultRoleClaimType, role)));

        return claims;
    }

    public JwtSecurityToken GenerateToken(SigningCredentials
signingCredentials,
        IEnumerable<Claim> claims)
    {
        var token = new JwtSecurityToken(
            issuer: _jwtSettings.Issuer,
            audience: _jwtSettings.Audience,
            claims: claims,
            expires: DateTime.Now.AddDays(30),
            signingCredentials: signingCredentials);

        return token;
    }
}
```

4.5 Використання патернів проектування та принципів програмування

У процесі розробки системи для вивчення іноземних мов були використані кілька важливих патернів проектування, які допомогли забезпечити модульність, чистоту коду та ефективну взаємодію компонентів. Серед основних патернів, що застосовувалися, є Dependency Injection (DI), Repository, Unit of Work, а також принципи програмування, такі як DRY, KISS та YAGNI.

Dependency Injection (DI) є одним з основних патернів, що використовувався в проєкті для зниження зв'язності між компонентами системи та забезпечення легкості тестування. Використання DI дозволяє легко інжектувати залежності в класи через їх конструктори, а також підвищує гнучкість і підтримуваність коду.

У .NET цей патерн реалізовано через вбудовану систему DI, яка дозволяє автоматично керувати життєвим циклом об'єктів та їх залежностями. Це спрощує заміну компонентів під час тестування і в разі змін у бізнес-логіці. Приклад DI:

```
private readonly IUnitOfWork _unitOfWork;
private readonly IMapper _mapper;
private readonly IContextAccessor _contextAccessor;

public ExerciseManager(
    IUnitOfWork unitOfWork,
    IMapper mapper,
    IContextAccessor contextAccessor)
{
    _unitOfWork = unitOfWork;
    _mapper = mapper;
    _contextAccessor = contextAccessor;
}
```

Repository — це патерн, що абстрагує доступ до даних і дозволяє працювати з базою даних через об'єкти замість написання SQL-запитів вручну. Це дозволяє спростити логіку доступу до даних і покращити підтримуваність коду. Всі операції з базою даних централізовані в репозиторіях, що знижує залежність між бізнес-логікою і даними. Приклад репозиторію:

```
public class GenericRepository<T> where T : BaseEntity
{
    private readonly AppDbContext _context;
    protected readonly DbSet<T> DbSet;

    protected GenericRepository(AppDbContext context)
    {
        _context = context;
        DbSet = context.Set<T>();
    }

    public IQueryable<T> AsQueryable()
    {
        return DbSet.AsQueryable();
    }

    public virtual async Task<List<T>> GetAllAsync(CancellationToken
cancellationToken = default)
    {
        return await DbSet.ToListAsync(cancellationToken);
    }

    public Task<IQueryable<T>>
GetAllAsQueryableAsync(CancellationToken cancellationToken = default)
```

```
    {
        return Task.FromResult(DbSet.AsQueryable());
    }

    public virtual async Task<T?> GetByIdAsync(Guid id,
CancellationToken cancellationToken = default)
    {
        return await DbSet.FirstOrDefaultAsync(x => x.Id == id,
cancellationToken);
    }

    public virtual async Task AddAsync(T entity, CancellationToken
cancellationToken = default)
    {
        await DbSet.AddAsync(entity, cancellationToken);
    }

    public virtual void Update(T entity)
    {
        DbSet.Update(entity);
    }

    public virtual void Delete(T entity)
    {
        DbSet.Remove(entity);
    }

    public async Task SaveChangesAsync(CancellationToken
cancellationToken = default)
    {
        await _context.SaveChangesAsync(cancellationToken);
    }
}
```

Unit of Work — це патерн, що дозволяє управляти транзакціями та координувати зміни в кількох репозиторіях. Це дозволяє забезпечити консистентність даних у разі помилок, оскільки всі зміни обробляються в рамках єдиної транзакції. Завдяки цьому патерну, зміни в базі даних будуть зафіксовані лише після успішного виконання всіх операцій. Реалізація Unit Of Work:

```
public class UnitOfWork : IUnitOfWork
{
    private readonly AppDbContext _dbContext;
    private readonly IServiceProvider _serviceProvider;
    private readonly Dictionary<string, object> _repositories;

    public UnitOfWork(
        AppDbContext dbContext,
        IServiceProvider serviceProvider)
    {
        _dbContext = dbContext;
```

```
        _serviceProvider = serviceProvider;
        _repositories = new Dictionary<string, object>();
    }

    public int SaveChanges()
    {
        return _dbContext.SaveChanges();
    }

    public Task<int> SaveChangesAsync(CancellationToken
cancellationToken = default)
    {
        return _dbContext.SaveChangesAsync(cancellationToken);
    }

    public T GetRepository<T>()
    {
        var typeName = typeof(T).Name;

        if (!_repositories.ContainsKey(typeName))
        {
            var instance = _serviceProvider.GetService<T>();
            if (instance is not null)
            {
                _repositories.Add(typeName, instance);
            }
        }

        return (T)_repositories[typeName];
    }
}
```

Принципи програмування, що були дотримані:

- DRY (Don't Repeat Yourself): Цей принцип застосовувався у всіх частинах системи для уникнення дублювання коду. Всі загальні функції, такі як перевірка даних або обчислення балів, були винесені в окремі сервіси, що зменшує дублювання і покращує підтримуваність коду;
- KISS (Keep It Simple, Stupid): Принцип KISS допоміг уникнути надмірної складності в архітектурі та реалізації функціоналу. Завжди вибирався найпростіший підхід для досягнення мети, що дозволяло зробити систему зрозумілою і легкою для розширення;
- YAGNI (You Aren't Gonna Need It): Принцип YAGNI передбачає реалізацію лише необхідного функціоналу, що дозволяє уникати надмірних витрат часу на створення функцій, які в майбутньому можуть не знадобитися.

5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Тестування застосунку

Для тестування програмного забезпечення я використовував мануальне тестування, а також інструменти для тестування API, такі як Swagger, Postman та Insomnia. Це дозволило перевірити основну функціональність, взаємодію між компонентами системи та коректність роботи серверної частини через API [11].

Я проводив мануальне тестування основних функцій системи, таких як реєстрація користувачів, авторизація, виконання завдань, система досягнень і лідербордів. Тестував функціональність веб-застосунку вручну, перевіряючи правильність роботи кожної частини інтерфейсу та взаємодії з сервером.

Для тестування взаємодії між клієнтською частиною та сервером використовував такі інструменти:

- Swagger: використовував для документування та тестування API, що дозволяло перевіряти ендпоїнти на відповідність специфікаціям;
- Postman: цей інструмент використовував для перевірки правильності обробки запитів та автоматизації тестування API ендпоїнтів;
- Insomnia: також використовував для тестування запитів та перевірки результатів від серверу, що допомагало виявити можливі помилки в обробці даних.

Мануально перевіряв всі основні функції платформи, зокрема реєстрацію, виконання завдань та збереження результатів. Це дозволило переконатися, що система працює без помилок і коректно зберігає прогрес користувачів.

5.2 Приклади критичних помилок

Під час тестування програмного забезпечення я виявив кілька критичних помилок, які потребували термінового виправлення для забезпечення стабільної роботи системи.

Під час тестування функції збереження результатів виконання завдань виявлено, що в окремих випадках прогрес користувача не зберігається після виконання завдання. Це відбувалося через некоректну обробку запитів на сервері

при високому навантаженні, що призводило до втрати даних. Проблема була вирішена шляхом оптимізації запитів до бази даних та додавання механізмів перевірки транзакцій.

Під час тестування API за допомогою Postman та Swagger було виявлено, що деякі ендпоїнти не обробляли помилки правильно. Наприклад, у випадку неправильних даних, система не повертала коректний код помилки (HTTP 400), а замість цього система давала неспецифіковану відповідь або навіть падала. Цю проблему було виправлено шляхом додавання додаткових перевірок на сервері та покращення обробки виключень.

Виявлено, що при введенні некоректних даних у форму реєстрації (наприклад, неправильний формат електронної пошти) система не показувала належне повідомлення про помилку. Це могло призвести до того, що користувачі не могли коректно завершити реєстрацію. Виправлення цієї проблеми полягало в удосконаленні валідації введених даних на клієнтській частині та додаванні відповідних повідомлень для користувача.

Під час тестування системи досягнень і прогресу було виявлено, що рівень користувача не оновлюється в залежності від виконаних завдань і отриманих балів. Це сталося через неправильну логіку обчислення балів в серверній частині. Після аналізу та оновлення алгоритму обчислення, система почала коректно відображати прогрес користувачів.

Інколи при оновленні прогресу або досягнень, система показувала застарілі дані, оскільки кешування не оновлювалося належним чином. Ця помилка викликала непорозуміння у користувачів, оскільки вони не бачили актуальних результатів після виконання завдання. Було вирішено додати механізм примусового очищення кешу після важливих змін, щоб забезпечити актуальність даних.

6 ВПРОВАДЖЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

6.1 Визначення плану впровадження

Після завершення розробки та тестування програмного забезпечення, наступним етапом стало його впровадження. Застосунок був задеплоєний у хмарну інфраструктуру Microsoft Azure, що забезпечило його високу доступність та масштабованість. Використання Azure дозволило легко масштабувати систему в залежності від навантаження, а також забезпечити безпеку даних та стабільність роботи сервісу.

Процес впровадження складався з кількох етапів:

- розгортання серверної та клієнтської частин на хмарних серверах Azure з використанням App Services для хостингу ASP.NET Core Web API;
- розгортання бази даних в Azure SQL Database, що забезпечує надійне зберігання даних з автоматичним резервним копіюванням та високою доступністю;
- налаштування інтеграції з Azure Blob Storage для зберігання великих файлів, таких як медіа-контент для завдань, аудіофайли, картинки тощо.

Для автоматизації процесу розгортання був налаштований CI/CD процес з використанням Azure DevOps. Цей процес включає автоматичне тестування, створення артефактів та деплой на сервери Azure, що значно скоротило час на оновлення та підтримку системи.

ВИСНОВКИ

За час виконання роботи була спроєктована, імплементована та протестована програмна система для інтерактивного вивчення іноземних мов, що має на меті забезпечити персоналізований процес навчання, використання різноманітних завдань та гейміфікації для підвищення мотивації користувачів. Система надає можливість створювати індивідуальні навчальні плани, обирати типи завдань, а також відстежувати прогрес через систему досягнень та лідербордів.

У ході роботи був проведений аналіз існуючих рішень у сфері вивчення мов, виявлено їхні обмеження та проблеми, зокрема у сфері кастомізації навчальних матеріалів, інтеграції з іншими платформами та відсутності повноцінної гейміфікації. На основі цього було сформовано вимоги до функціоналу програмного продукту, що дозволили створити систему, яка відповідає сучасним вимогам та потребам користувачів.

Після аналізу предметної галузі та конкурентів були визначені ключові функції системи, створена специфікація продукту, що формалізувала функціональні та нефункціональні вимоги. З використанням UML діаграм було спроектовано архітектуру системи, розроблені діаграми прецедентів, діяльності та ER-діаграма бази даних.

У процесі реалізації були використані сучасні технології, такі як ASP.NET Core для серверної частини та Angular для клієнтської частини. Після завершення розробки було проведено тестування функціональності та продуктивності системи, що дозволило виявити і виправити критичні помилки.

Розроблена програмна система дозволяє користувачам ефективно вивчати іноземні мови, створюючи індивідуалізований підхід до навчання, а також підвищує мотивацію через елементи гейміфікації. Впровадження цієї платформи може значно полегшити процес вивчення мов для широкого кола користувачів і зробити його більш захоплюючим та інтерактивним.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Duolingo – офіційний сайт [Електронний ресурс] – URL: https://www.duolingo.com/ (дата звернення: 8.05.2025)

2. Babbel – офіційний сайт [Електронний ресурс] – URL: https://www.babbel.com/ (дата звернення: 8.05.2025)

3. Rosetta Stone – офіційний сайт [Електронний ресурс] – URL: https://eu.rosettastone.com/ (дата звернення: 8.05.2025)

4. Memrise – офіційний сайт [Електронний ресурс] – URL: https://www.memrise.com/ (дата звернення: 19.05.2025)

5. Anki – офіційний сайт [Електронний ресурс] – URL: https://apps.ankiweb.net/ (дата звернення: 20.05.2025)

6. How to create your own language app and succeed: Step-by-step guide [Електронний ресурс] – URL: https://rewisoft.com/blog/how-to-create-your-own-language-app-and-succeed-step-by-step-guide/ (дата звернення: 06.04.2025)

7. Reddit Post [Електронний ресурс] – URL: https://www.reddit.com/r/languagelearning/comments/1j8yryb/ive_spent_6_months_fulltime_building_a_language/ (дата звернення: 06.04.2025)

8. ASP.NET Core Web API – документація [Електронний ресурс] – URL: https://learn.microsoft.com/en-us/aspnet/core/web-api/?view=aspnetcore-5.0 (дата звернення: 01.06.2025)

9. Angular – офіційна документація [Електронний ресурс] – URL: https://angular.io/docs (дата звернення: 10.06.2025)

10. Microsoft SQL Server – документація [Електронний ресурс] – URL: https://learn.microsoft.com/en-us/sql/sql-server/?view=sql-server-ver15 (дата звернення: 18.05.2025)

11. Docker – офіційна документація [Електронний ресурс] – URL: https://www.docker.com/what-docker (дата звернення: 05.06.2025)

12. Посилання на GitHub репозиторій з вихідним кодом, відео та специфікацією. URL: https://github.com/techcat18/2025_B_PI_PZPI-21-3_Bondarenko_K_T (дата звернення: 21.06.2025)