МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет

Факультет електроніки та інформаційних технологій

Кафедра комп'ютерних наук

«До захисту допущено»

В.о. завідувача кафедри

_____ Ігор ШЕЛЕХОВ
(підпис)

« » червня 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавр

зі спеціальності 122 – Комп'ютерних наук,
освітньо-професійної програми «Інформатика»
на тему: «Інформаційна система аудіо-візуальної підтримки процесу навчання осіб з особливими освітніми потребами»
здобувача групи ІН-01 Червякова Максима Олександровича

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ Максим ЧЕРВЯКОВ
(підпис)

Керівник,
старший викладач кафедри
комп'ютерних наук,
к.т.н., доцент Борис КУЗІКОВ _____
(підпис)

Суми – 2024

Сумський державний університет

Факультет електроніки та інформаційних технологій

Кафедра комп'ютерних наук

«Затверджую»

В.о. завідувача кафедри

_____ Ігор ШЕЛЕХОВ
(підпис)

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавра

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»
здобувача групи ІН-01 Червякова Максима Олександровича

1. Тема роботи: «Інформаційна система аудіо-візуальної підтримки процесу навчання осіб з особливими освітніми потребами»
   затверджую наказом по СумДУ від *«22» квітня 2024 р. № 0414-VI*
2. Термін здачі здобувачем кваліфікаційної роботи *до 29 травня 2024 року*
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) _____
6. Консультанти до проекту (роботи), із зазначенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «__» _____ 20__ р.

Завдання прийняв на виконання _____ Керівник _____
(підпис) (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз проблеми предметної області, постановка й формування завдань дослідження*		
2	*Огляд технологій, що використовуються для забезпечення аудіо-візуальної підтримки процесу навчання*		
3	*Розробка інформаційної системи аудіо-візуальної підтримки процесу навчання*		
4	*Аналіз отриманих результатів*		
5	*Оформлення пояснювальної записки до кваліфікаційної роботи*		

Здобувач вищої освіти _____ Керівник _____
(підпис) (підпис)

АНОТАЦІЯ

Записка: 76 стр., 26 рис., 5 табл., 1 додаток, 33 використаних джерела.

Обґрунтування актуальності теми роботи – Тема кваліфікаційної роботи є актуальною, оскільки присвячена розв’язанню важливої практичної задачі створення додатку аудіо-візуальної підтримки процесу навчання осіб з особливими освітніми потребами шляхом розробки інформаційною системи сучасними засобами проектування та розробки.

Об’єкт дослідження – процес навчання осіб з особливими освітніми потребами.

Мета роботи – дослідження, планування та розробка інформаційної системи аудіо-візуальної підтримки процесу навчання осіб з особливими освітніми потребами з використанням карток.

Методи дослідження – інструменти проектування та розробки багатофункціональних інформаційних систем.

Результати – розроблено інформаційну систему, яка може ефективно підтримувати навчальний процес за допомогою карток, забезпечуючи зручний інтерфейс для користувачів та можливість вивчення матеріалу в інтерактивному форматі.

АУДІО-ВІЗУАЛЬНА ПІДТРИМКА, ІНФОРМАЦІЙНА СИСТЕМА, НАВЧАННЯ, ASP.NET CORE, REACT.

ЗМІСТ

ВСТУП

Сучасний світ невпинно змінюється та розвивається, з ним змінюються і вимоги до освіти. Інформаційні технології надали освітнім процесам нові можливості, дозволяючи ефективно адаптувати навчання до потреб та індивідуальних особливостей кожного студента. Однією з інноваційних технологій, що значно впливає на навчальний процес, є використання карток як інструменту аудіо-візуальної підтримки процесу навчання та підвищення його ефективності.

Інформаційні системи аудіо-візуальної підтримки процесу навчання осіб з особливими освітніми потребами з використанням карток відіграють важливу роль у покращенні якості освіти та забезпеченні інклюзивного та індивідуалізованого підходу до кожного студента. Картки можуть бути використані для різних цілей, таких як узагальнення матеріалу, тестування знань, стимулювання критичного мислення та розвитку творчості. Їхня варіативність та можливість швидкого доступу до інформації робить їх незамінним інструментом для сучасного освітнього процесу.

Метою даної роботи є дослідження, планування та розробка інформаційної системи аудіо-візуальної підтримки процесу навчання осіб з особливими освітніми потребами з використанням карток. У процесі роботи буде здійснено аналіз сучасних підходів до навчання, розглянуто теоретичні аспекти використання карток у навчальному процесі, розроблено архітектуру інформаційної системи, визначено платформи для розробки та розроблено інформаційну систему.

Об’єкт дослідження. Процес навчання осіб з особливими освітніми потребами.

Предмет дослідження. Методи, технології та засоби подання навчальних матеріалів у вигляді карток з аудіо-візуальною підтримкою для осіб з особливими освітніми потребами.

Гіпотеза. Розроблена інформаційна система аудіо-візуальної підтримки процесу навчання осіб з особливими освітніми потребами з використанням карток зробить процес навчання більш ефективним, доступнішим та зручним для самостійного навчання.

Новизна. Полягає у розробці навчальної платформи для ефективного вивчення нового матеріалу за допомогою карток з використанням засобів аудіо-візуальної підтримки.

Структура. Дана робота складається зі вступу, детального аналітичного огляду, постановки задачі, вибору методу розв’язання поставленої задачі, опису програмного забезпечення інформаційної системи, висновків, списку використаних джерел та додатків.

1. АНАЛІТИЧНИЙ ОГЛЯД

1.1. Існуючі онлайн сервіси для навчання за допомогою карток

У сучасному світі, де зростає значення неперервного самовдосконалення та освіти, необхідно шукати ефективні методи навчання, які будуть зручними у використанні кожному, незалежно від освітніх потреб. Одним з таких методів є використання карток для навчання, який зарекомендував себе як ефективний інструмент для запам'ятовування та організації знань. Навчання за допомогою карток - це метод, що полягає у використанні карток з питаннями на одній стороні та відповідями на іншій для активного запам'ятовування та перевірки знань. Цей метод використовується в освітніх цілях для поліпшення запам'ятовування та вивчення нової інформації. Вивчаючи інформацію на картках, студент має можливість самостійно перевіряти свої знання, швидко виявляти прогалини та зосереджуватись на найважливіших аспектах матеріалу. За останні роки з появою онлайн сервісів та мобільних додатків, навчання за допомогою карток стало більш зручним та доступним.

Багато досліджень показують, що навчання за допомогою карток є ефективним методом для підвищення запам'ятовування та розуміння навчального матеріалу. Наприклад, дослідження проведене Джоном Данлоскі та іншими 2013 році [1] показало, що студенти, які використовували флеш-картки, показали набагато кращі результати на контрольних випробуваннях у порівнянні з студентами, які задля запам’ятовування матеріалу використовували перечитування та переказування матеріалу.

На сьогоднішній день існує багато онлайн сервісів, які надають можливість навчатися за допомогою карток. Розглянемо деякі з них детальніше.

1.1.1. Quizlet

Quizlet [2] — це популярний додаток, який використовується для навчання за допомогою карток. Він допомагає студентам та викладачам створювати, обмінюватися та вивчати навчальні матеріали у різних формах.

Даний додаток має наступний функціонал:

- Основна функція Quizlet — це створення та використання карток для навчання. Картки можуть містити текст, зображення, аудіо та навіть певні типи кодування, що робить їх універсальними для різних навчальних цілей.
- Додаток пропонує різноманітні тести та ігри, щоб допомогти студентам вивчати матеріал. Наприклад, можна використовувати режими "Випробування" та "Поєднання", щоб перевірити знання.
- Користувачі можуть обмінюватися своїми наборами карток з іншими, що сприяє спільному навчанню та співпраці.
- Quizlet використовує алгоритми, щоб визначити, які питання є складними для користувача, і автоматично повторює їх, допомагаючи закріпити знання.
- Додаток підтримує аудіо та візуальний контент, що може бути корисним для студентів з різними особливими потребами. Наприклад, додаток може озвучувати текст, що робить його доступним для людей з проблемами зору.
- Quizlet підтримує багато мов, що робить його зручним для різноманітної аудиторії.
- Quizlet доступний на різних платформах, включаючи веб-браузери, iOS та Android, що дозволяє використовувати його на різних пристроях.

Загальні переваги додатку:

- Quizlet має інтуїтивний інтерфейс, що робить його легким у використанні для широкого кола користувачів.

- Додаток дозволяє створювати контент різного типу, що допомагає адаптувати його під різні навчальні потреби.

Недоліки додатку:

- Безкоштовна версія Quizlet містить рекламу, що може відволікати увагу користувачів і створювати дискомфорт під час навчання.
- Деякі корисні функції, такі як додавання зображень або аудіо, обмежені в безкоштовній версії, що може впливати на якість навчання.

1.1.2. Anki

Anki [3] – це ще один досить популярний додаток для навчання за допомогою карток, який надає користувачам потужний інструмент для ефективного запам'ятовування та відтворення інформації.

Даний додаток має наступний функціонал:

- Anki дозволяє користувачам створювати картки з текстом, зображеннями, звуком та навіть відео, а також надає можливість налаштувати структуру карток з використанням HTML та CSS.
- Anki має можливість синхронізувати дані між різними пристроями, що дозволяє користувачам навчатися будь-де. Додаток доступний для Windows, macOS, Linux, iOS та Android, з можливістю імпорту та експорту даних.
- Anki підтримує додатки та надбудови, які додають нові функції та можливості.
- Anki використовує алгоритм повторення з інтервалами, який оптимізує час вивчення, повторюючи складні картки частіше, а прості рідше. Це допомагає ефективніше закріплювати інформацію в довгостроковій пам'яті.
- Anki дозволяє користувачам налаштовувати планування навчання, встановлювати щоденні цілі та відстежувати прогрес. Можливо

створити індивідуальні розклади та переглядати статистику, яка допоможе налаштувати навчальні звички.

Недоліки додатку:

- Через свою гнучкість і різноманіття функцій Anki може бути складним для новачків. Навчання може вимагати часу та зусиль, щоб повністю зрозуміти всі можливості програми та її систему управління колодами та картками.
- Користувачі іноді стикаються з проблемами синхронізації, помилками під час використання, а також проблемами з безпекою облікових записів.
- Система повторення з інтервалами вимагає регулярності в навчанні. Якщо користувач не дотримується регулярного графіка, ефективність програми знижується, і вона не дає таких самих результатів, як при регулярному використанні.

1.1.3. OmniSets

OmniSets [4] – це інноваційний додаток для навчання за допомогою карток, який використовує штучний інтелект, щоб полегшити та підвищити ефективність навчального процесу. Цей додаток має безліч функцій, які допомагають користувачам створювати, організовувати та використовувати картки для навчання.

Даний додаток має наступний функціонал:

- OmniSets дозволяє створювати картки кількома способами
  - Користувачі можуть вручну створювати картки, додаючи терміни та визначення, а також візуальний контент.
  - Функція "Create with AI" дозволяє генерувати картки на будь-яку тему, використовуючи алгоритми штучного інтелекту, які шукають інформацію в базі даних та в Інтернеті.
  - OmniSets може конвертувати ваші нотатки в картки за допомогою функції "Create from note", що дозволяє швидко створювати велику кількість карток з існуючих матеріалів.

- OmniSets пропонує різні режими навчання:
  - Співставлення: Цей режим включає ігри та вправи на швидкість, що допомагають підвищити короткочасну пам'ять.
  - Питання та відповіді: Можливість тестування за допомогою питання і відповіді, щоб перевірити, наскільки добре засвоєно матеріал.
  - Spell Mode: Режим, який допомагає практикувати написання термінів та визначень для закріплення довготривалої пам'яті.
- OmniSets дозволяє організувати картки за допомогою папок, тегів та інших інструментів для категоризації. Також можливо створювати плани навчання для структурованого вивчення, а також ділитися своїми картками з іншими користувачами для спільного навчання.

Недоліки додатку:

- OmniSets поки що не має власного мобільного додатку. Проте він доступний через Progressive Web App (PWA), це може бути менш зручним, ніж повноцінний мобільний додаток. Деякі користувачі можуть відчувати обмеження у функціональності або стабільності під час використання PWA.
- Функція "Create with AI" може створювати картки, але вони іноді можуть бути менш точними або деталізованими порівняно з вручну створеними картками. Це може вимагати додаткової перевірки та редагування, що може уповільнити процес підготовки матеріалів.

1.1.4. Brainscape

Brainscape [5] — це ще один досить популярний додаток для навчання за допомогою карток, що надає користувачам інструменти для ефективного навчання та запам'ятовування інформації.

Даний додаток має наступний функціонал:

- Brainscape дозволяє користувачам створювати та використовувати картки для вивчення різних тем. Картки можуть містити текст, зображення, аудіо, і їх можна організувати за категоріями.
- Додаток використовує технологію, яка допомагає користувачам повторювати інформацію через певні проміжки часу. Це підвищує ефективність запам'ятовування.
- Користувачі можуть оцінювати картки від 1 до 5, що впливає на те, як часто вони повторюватимуться в майбутньому.
- Brainscape дозволяє користувачам ділитися своїми наборами карток з іншими, а також використовувати картки, створені іншими користувачами.

Недоліки додатку:

- Інтерфейс може бути складним для деяких користувачів з особливими освітніми потребами. Додаток не має спеціальних налаштувань для людей з вадами зору.
- Більшість розширених функцій і готових наборів карток потребують платної підписки.

Окрім згаданих сервісів, існує ще безліч інших онлайн платформ, які надають можливість навчатися за допомогою карток, таких як Memrise, Cram, StudyBlue та багато інших. Кожен з цих сервісів має свої особливості та певні недоліки, тому ціллю даної роботи буде створення простого, інтуїтивно зрозумілого та зручного у використанні онлайн сервісу, в якому буде враховано плюси та мінуси існуючих систем аудіо-візуальної підтримки процесу навчання осіб з особливими освітніми потребами з використанням карток.

1.2. Хмарні сервіси та їх особливості

Створення онлайн сервісу аудіо-візуальної підтримки процесу навчання осіб з особливими освітніми потребами з використанням карток передбачає розміщення додатку в мережі інтернет. Це забезпечує глобальний доступ до сервісу, незалежно від місця знаходження користувачів. Для розміщення додатку існує кілька способів, кожен з яких має свої особливості, переваги та недоліки. Розглянемо їх детальніше:

1. Dedicated Server (Виділений сервер). Даний спосіб розміщення передбачає виділення окремого фізичного серверу для хостингу додатку. У такому випадку сервіс матиме високу продуктивність та безпеку, адже всі ресурси використовуються виключно одним застосунком. Однак управління та підтримка серверу, велика вартість оренди або придбання фізичного сервера може бути досить вагомим недоліком такого рішення, особливо для початківців [6].
2. Virtual Private Server (VPS) (Віртуальний приватний сервер). Віртуальний приватний сервер – це віртуальний сервер, що функціонує на фізичному сервері, але має окремо виділені ресурси та ізоляцію від інших користувачів. Даний спосіб розміщення коштує дешевше від окремого серверу, але ще є досить дорогим. Проте таке рішення забезпечує більшу контрольованість та масштабованість віртуального середовища [7].
3. Shared Hosting (Спільний хостинг). У спільному хостингу додаток розміщується на одному сервері разом з іншими додатками. Всі користувачі спільного хостингу поділяють ресурси сервера, такі як процесор, пам'ять та дисковий простір. Перевагами такого рішення є досить низька вартість розміщення та простота у використанні, адже провайдер самостійно займається підтримкою робочого стану фізичного серверу. Однак у спільному хостингу беруть участь й інші користувачі, тому існують обмеження у використанні ресурсів та контролі системи [8].

4. Cloud Hosting (Хмарний хостинг). Хмарний хостинг є одним з популярних способів розміщення додатків та веб-сайтів в мережі Інтернет. Він базується на ідеї використання інфраструктури хмарних сервісів, таких як Azure Cloud, AWS, Google Cloud, та інших. Замість того, щоб спиратися на окремий фізичний сервер, як це відбувається, наприклад, у спеціальному сервері (dedicated server), хмарний хостинг використовує віртуальні ресурси, які можуть миттєво масштабуватися. Хмарний хостинг дозволяє за потребою збільшувати та зменшувати кількість виділених ресурсів та забезпечує високі стандарти безпеки, включаючи шифрування даних та захист мережі від зламів. Попри таку зручність, вартість за використання є відносно невеликою, адже хмарні сервіси передбачають оплату лише за ті ресурси, що використовуються. Це робить хмарний хостинг більш економічно вигідним для бізнесу [9].

Враховуючи всі переваги та недоліки існуючих способів розміщення додатку в мережі інтернет, хмарний хостинг є найбільш привабливим варіантом. Його гнучкість, масштабованість, безпека та вартість дозволяють легко та ефективно розвивати та обслуговувати сервіс онлайн навчання за допомогою карток.

Однією з популярних хмарних платформ є Microsoft Azure [10]. Microsoft Azure - це хмарна платформа, розроблена компанією Microsoft, яка надає широкий спектр хмарних послуг для розробки, тестування, впровадження та керування додатками та послугами через глобальну мережу даних. Azure пропонує велику кількість рішень для обробки даних, штучного інтелекту, аналітики, IoT, а також облаштування хмарної інфраструктури для запуску веб-додатків та мобільних додатків.

Azure Virtual Machines (VM) [11] — це хмарний сервіс від Microsoft Azure, який дозволяє розгортати віртуальні сервери з різноманітними операційними системами та конфігураціями апаратного забезпечення. Це гнучке рішення, яке підходить для різноманітних застосувань, від невеликих

тестових середовищ до великих високонавантажених систем (див. Рисунок 1.1).

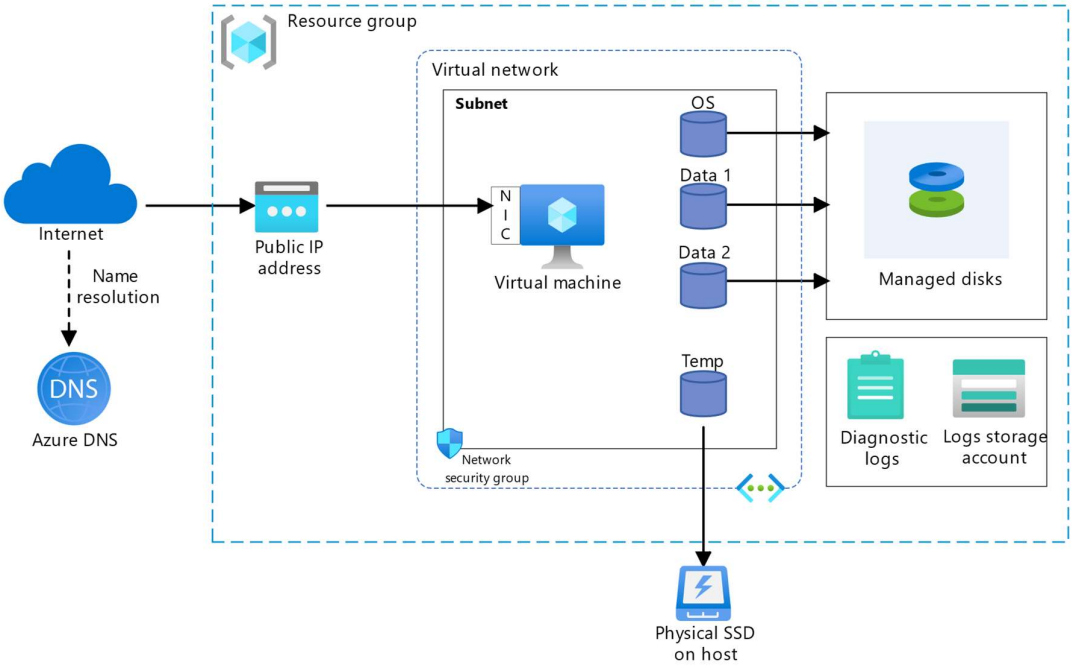

Рисунок 1.1 – архітектура Azure VM [12]

Основні можливості та переваги Azure VM:

1. Вибір операційних систем: Azure VM підтримує різні операційні системи, включаючи Windows Server, Linux (Ubuntu, Red Hat, CentOS та інші).
2. Гнучкість конфігурації: Можна обрати з багатьох типів віртуальних машин, кожен з яких має різні параметри CPU, RAM, дискової пам'яті та мережевих можливостей. Це дозволяє налаштувати VM відповідно до конкретних потреб проекту.
3. Широкий набір інтеграцій: Azure VM інтегрується з іншими сервісами Azure, такими як Azure Blob Storage, Azure Virtual Network, Azure Active Directory та інші. Це забезпечує комплексне хмарне середовище для застосунків.
4. Резервне копіювання та відновлення: Azure Backup дозволяє створювати резервні копії віртуальних машин для захисту від втрати

даних. Azure Site Recovery забезпечує можливості відновлення після аварії та міграції даних.

5. Масштабованість: Azure VM дозволяє легко масштабувати ресурси вгору чи вниз відповідно до потреб проекту. Це забезпечує гнучкість при збільшенні чи зменшенні навантаження на систему.
6. Глобальна доступність: Microsoft має розгалужену мережу дата-центрів по всьому світу, що забезпечує високу доступність і можливість розгортання ресурсів ближче до користувачів системи.
7. Безпека: Azure пропонує надійні засоби безпеки, такі як Azure Security Center, Azure Firewall, і інструменти для управління доступом та шифруванням даних. Це гарантує захист ресурсів додатку від зовнішніх загроз.

Хоча Microsoft Azure є потужною хмарною платформою з багатьма перевагами, вона також має деякі недоліки і обмеження:

1. Складність вивчення: Azure має велику кількість функцій та сервісів, що може зробити його вивчення дещо складним, особливо для новачків. Інколи налаштування та управління окремими аспектами можуть вимагати певного рівня експертизи.
2. Складність налаштування: Відповідно до потреб та конфігурації, налаштування Azure VM може вимагати значного часу та знань. Інтеграція з іншими сервісами також може бути складною для менш досвідчених користувачів.
3. Локалізованість даних: Деякі організації можуть стикається з обмеженнями з приводу того, де зберігаються їхні дані в рамках Azure. У деяких країнах або галузях діяльності існують правила щодо локалізації даних, які можуть ускладнити використання хмарної платформи.
4. Конфігураційні проблеми: При неправильній конфігурації можуть виникати проблеми з безпекою, продуктивністю або масштабованістю

додатків. При використанні платформи слід ретельно перевіряти налаштування для забезпечення оптимальної роботи додатків.

5. Вартість: Хоча Azure пропонує гнучкі цінові плани, використання великої кількості ресурсів або тривале зберігання даних може бути дорогим. Це вимагає ретельного планування та оптимізації ресурсів.

Незважаючи на ці недоліки, Microsoft Azure залишається популярною платформою для розгортання хмарних додатків через свою широку функціональність, надійність та можливості масштабування.

1.3. Постановка задачі

Для виконання поставленої мети необхідно розробити інформаційну систему аудіо-візуальної підтримки процесу навчання осіб з особливими освітніми потребами з використанням карток, яка дозволить студентам ефективніше засвоювати навчальний матеріал й розвивати свої знання та навички.

Розроблений онлайн сервіс повинен включати наступний функціонал:

1. Реєстрація та вхід користувачів:
    - Можливість реєстрації за допомогою адреси електронної пошти.
    - Вхід в систему з обліковими даними користувача.
2. Створення та редагування наборів карт:
    - Користувач може створювати нові набори карток, надаючи їм назву та опис.
    - Можливість додавати нові картки до набору, що мають інформацію вказану на передній та зворотній сторонах.
    - Можливість додавати графічні зображення до карток.
    - Можливість групування наборів карток за тематикою.
3. Перегляд карток:
    - Можливість перегляду власних створених карток.

- Можливість перегляду доступних карток інших користувачів.
- Можливість прослуховування тексту карток.

4. Дизайн та інтерфейс:
    - Дизайн користувацького інтерфейсу додатку повинен бути інтуїтивно зрозумілим та привабливим.
5. Забезпечення безпеки:
    - Додаток повинен використовувати шифрування для зберігання паролів та інших конфіденційних даних.

2. ВИБІР МЕТОДУ ВИРІШЕННЯ ЗАДАЧІ

2.1. Інформаційна модель додатку

При розробці програмного забезпечення інженери стикаються з вибором оптимальної архітектури, яка допомагає досягти високої продуктивності, розширюваності та забезпечити легкість управління додатком. Серед основних типів архітектури можна виділити монолітну, багаторівневу (трирівневу), мікросервісну, та інші [13].

Монолітна архітектура полягає в розробці додатку у вигляді єдиного, монолітного модуля, де всі компоненти об'єднані в одному додатку. Ця архітектура має певні переваги, такі як простота в розробці та розгортанні, але при зростанні розміру додатку може зазнати проблем з масштабованістю та підтримкою.

Мікросервісна архітектура передбачає розділення додатку на невеликі, автономні сервіси, які взаємодіють через мережу. Це дозволяє гнучко масштабувати та розвивати окремі компоненти, але приносить незручності у керуванні та розгортанні всього додатку, а також не підходить для невеликих додатків через свою складність.

Трирівнева архітектура (багаторівнева архітектура) включає розділення додатку на три логічні рівні: клієнтський, серверний та шар бази даних. Ця архітектура надає зручну структуру для розробки, де кожен рівень має свою відповідальність та функції. Клієнтський рівень забезпечує інтерфейс для користувачів, серверний рівень обробляє бізнес-логіку та запити користувачів, а шар бази даних зберігає інформацію [14] (див Рисунок 2.1).

Трирівнева архітектура є однією з найбільш поширених та ефективних архітектур для розробки веб-додатків. Вона дозволяє розділити функціональність додатку на логічні групи та забезпечує модульність, що полегшує підтримку та масштабування проекту, а тому чудово підходить для розробки онлайн сервісу для навчання за допомогою карток.

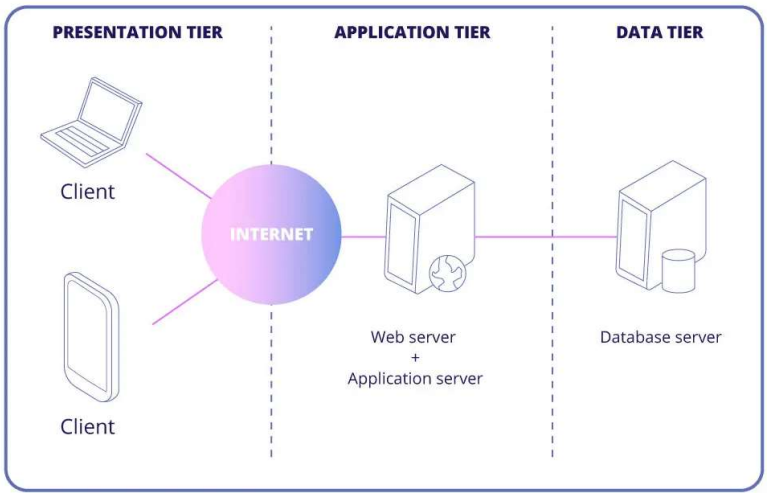

Рисунок 2.1 – діаграма трирівневої архітектури [15]

2.2. Вибір платформи для розробки серверного шару

У даній інформаційній системі серверний шар відповідає за обробку запитів від клієнтського шару та виконання бізнес-логіки, також цей шар звертається до бази даних. Для таких цілей доцільно використовувати архітектурний підхід REST API.

REST API [16] (Representational State Transfer Application Programming Interface) - це стиль архітектури програмного інтерфейсу, який визначає набір правил для створення веб-сервісів. Його ключові принципи включають використання HTTP для передачі даних та робота з ресурсами через визначені методи. Для створення REST API необхідно дотримуватися наступних принципів проектування:

- Розділення клієнтського і серверного шарів, дозволяючи кожному розвиватися незалежно. Клієнтський шар запитує ресурси, а серверний їх надає.
- Всі запити повинні дотримуватися стандартного набору методів HTTP, таких як GET, POST, PUT, DELETE, PATCH.
- Кожен запит від клієнта до сервера повинен містити всю необхідну інформацію, оскільки сервер не зберігає контекст між запитами.
- Відповіді від сервера можуть кешуватися для зменшення навантаження на сервер і підвищення продуктивності.

- REST дозволяє розділити компоненти на шари, наприклад, балансування навантаження, кешування, або безпеку.

Для написання серверного шару із застосуванням архітектурного підходу REST API використовуються багато різних мов програмування, фреймворків та платформ. Розглянемо найпопулярніші з них детальніше:

- JavaScript/Node.js є популярним вибором, особливо для розробки високопродуктивних асинхронних серверів. Тут зазвичай використовується фреймворк Express.js, який надає простий спосіб створення REST API.
- Python також є потужним інструментом для створення серверного коду, з такими фреймворками як Flask та Django. Його гнучкість і багата екосистема роблять його популярним серед розробників, особливо для малих і середніх проектів.
- Java є традиційним вибором для серверного програмування, пропонуючи високу продуктивність та надійність. Популярні фреймворки, такі як Spring та JAX-RS, надають інструменти для розробки REST API.
- PHP широко використовують для веб-розробки. Laravel та Symfony – це фреймворки, що полегшують створення REST API.
- C# разом з платформою .NET пропонує фреймворк ASP.NET Core для створення серверних застосунків, забезпечуючи високу продуктивність та гнучкість.

Для розробки серверного шару даної інформаційної системи було вирішено використовувати мову програмування C# із застосуванням фреймворку ASP.NET Core.

C# [17] – це об'єктно-орієнтована мова програмування, створена Microsoft у 2000 році як частина платформи .NET. Вона була розроблена під впливом мов, таких як C++ та Java, і націлена на простоту, безпеку та продуктивність. C# поєднує в собі багато корисних особливостей, включаючи підтримку класів, наслідування, інтерфейсів та поліморфізму.

Однією з переваг цієї мови є її статична типізація, яка дозволяє виявляти багато помилок на етапі компіляції, забезпечуючи додаткову безпеку.

C# інтегрована з платформою .NET, що надає доступ до великого набору бібліотек для роботи з різними технологіями, від баз даних до веб-розробки та мережевої взаємодії. Лямбда-вирази та LINQ дозволяють писати код у функціональному стилі, що робить його більш ефективним у деяких випадках. Крім того, C# має вбудовану підтримку асинхронного програмування, що особливо корисно при розробці веб-додатків, де потрібна висока продуктивність та масштабованість.

Платформа .NET включає в себе різні компоненти та інструменти для розробки програмного забезпечення. Спочатку вона була орієнтована лише на Windows-платформи і мала назву .NET Framework. Однак згодом з'явилася крос-платформна версія під назвою .NET Core. Вона дозволяє розробляти та запускати програми на операційних системах Windows, macOS та Linux, а також є більш модульною, що полегшує контейнеризацію та масштабування додатків.

ASP.NET Core [18] – це фреймворк для створення веб-додатків та сервісів, який входить до складу .NET Core. Крос-платформність і модульність роблять його популярним серед розробників, оскільки він дозволяє створювати сучасні веб-додатки, які можна запускати на різних платформах. Інтеграція з C# надає ASP.NET Core додаткову зручність, оскільки дозволяє використовувати об'єктно-орієнтовані конструкції та переваги асинхронного програмування.

ASP.NET має вбудовану підтримку REST, що дозволяє легко створювати API, які можна використовувати з різними клієнтами, такими як веб-браузери, мобільні додатки чи інші серверні сервіси. Розробка RESTful API в ASP.NET зазвичай включає використання контролерів для обробки HTTP-запитів, а також моделей, що представляють дані.

Ще одним важливим компонентом в екосистемі ASP.NET є Entity Framework [19]. Це Object Relational Mapper (ORM), який спрощує роботу з

базами даних у рамках ASP.NET. Замість написання складних SQL-запитів, можна використовувати C# для роботи з даними, представляючи їх у вигляді об'єктів. Entity Framework автоматично перетворює операції над об'єктами на відповідні SQL-запити, що значно спрощує роботу з базами даних. Це робить Entity Framework дуже корисним для розробки RESTful API, оскільки можна зосередитися на бізнес-логіці, а не на деталях взаємодії з базою даних.

При розробці RESTful API з використанням ASP.NET та Entity Framework, важливу роль відіграє правильне проектування та організація коду. Для цього зазвичай створюються окремі контролери для різних ресурсів, що дозволяє організувати код за принципом розділення відповідальності. Entity Framework також підтримує міграції, що дозволяє легко змінювати структуру бази даних у процесі розробки, а також автоматично синхронізувати ці зміни з програмним забезпеченням.

Таким чином ASP.NET та Entity Framework надають широкий набір інструментів для розробки серверного шару інформаційної системи з використанням архітектурного підходу REST API.

2.3. Вибір платформи для розробки клієнтського шару

У цій інформаційній системі клієнтський шар відповідає за візуальне представлення, взаємодію з користувачем та відправлення запитів на серверний шар для обробки.

Основною мовою програмування для розробки клієнтського шару є мова програмування JavaScript. JavaScript [20] – це високорівнева, інтерпретована мова програмування, що широко використовується для розробки веб-застосунків. За останні роки, JavaScript стала домінуючою мовою веб-розробки, завдяки своїй універсальності, гнучкості та широкій спільноті розробників.

JavaScript базується на ECMAScript стандарті, який регулює синтаксис та поведінку мови. Вона підтримує об'єктно-орієнтований та функціональний підходи до програмування. Синтаксис JavaScript простий та зрозумілий, що

робить його доступним для новачків, але має потужні можливості для досвідчених розробників.

JavaScript є основною мовою для динамічного контенту на веб-сторінках. Вона використовується для взаємодії з користувачем через обробку подій, асинхронне завантаження даних та маніпулювання DOM структурою. Проте її основним недоліком є використання динамічної типізації. Це означає, що основна частина перевірок типів змінних виконується під час виконання програми, а не під час компіляції. Щоб виправити цей недолік існує мова програмування TypeScript.

TypeScript [21] – це мова програмування, розроблена компанією Microsoft, яка розширює можливості JavaScript, додаючи статичну типізацію, об'єктно-орієнтовані концепції та інші сучасні можливості. TypeScript був розроблений для того, щоб вирішити проблеми, які виникають при масштабній розробці на JavaScript, і став популярним серед розробників, які прагнуть підвищити надійність та передбачуваність свого коду.

TypeScript – це надбудова для JavaScript, що означає, що будь-який коректний код на JavaScript також є коректним в TypeScript. Основна відмінність полягає у використанні типізації. Завдяки статичній типізації TypeScript дозволяє виявляти помилки на етапі компіляції, що допомагає запобігти багатьом поширеним помилкам у час виконання.

TypeScript також підтримує багато сучасних концепцій, таких як класи, інтерфейси, абстрактні класи та багато іншого. Ці особливості роблять TypeScript ідеальним для об'єктно-орієнтованого програмування та більш структурованого підходу до розробки.

Разом з мовами програмування JavaScript та TypeScript використовуються мови розміток та стилів HTML та CSS. HTML [22] (HyperText Markup Language) і CSS [23] (Cascading Style Sheets) є основними мовами розмітки, що використовуються для створення веб-сторінок і веб-застосунків. HTML визначає структуру та вміст веб-сторінок, тоді як CSS відповідає за їхній візуальний стиль і макет.

Проте використання лише цих технологій робить розробку клієнтського шару досить складною та громіздкою. Тому для полегшення роботи використовують додаткові фреймворки та бібліотеки:

- React: Одна з найпопулярніших бібліотек для створення компонентів, яка надає можливість побудови інтерактивних інтерфейсів. Використовується для розробки одно сторінкових додатків (SPA).
- Angular: Фреймворк розроблений компанією Google, який використовує TypeScript та пропонує структурований підхід до розробки. Він часто обирається для великих корпоративних проектів.
- Vue.js: Легкий фреймворк для створення компонентів, який поєднує простоту та потужність. Він підходить для невеликих і середніх проєктів.
- Svelte: Відносно новий фреймворк, який компілюється в простий JavaScript, дозволяючи отримати легкі та швидкі програми.

Для розробки клієнтського шару системи аудіо-візуальної підтримки процесу навчання осіб з особливими освітніми потребами з використанням карток було вирішено використовувати бібліотеку React з використанням мови програмування TypeScript.

React [24] – це популярна JavaScript-бібліотека, яку розробила компанія Facebook для створення інтерактивних користувацьких інтерфейсів веб-додатків. Основною особливістю React є компонентний підхід: інтерфейс користувача розбивається на невеликі компоненти, кожен з яких має свій стан і логіку. Така структура дозволяє легко підтримувати та масштабувати проекти.

React використовує віртуальний Document Object Model (DOM), що оптимізує процес оновлення реального DOM. Замість повного рендерингу React виконує зміни лише там, де це необхідно. Це значно підвищує продуктивність і швидкість роботи додатків.

Однією з переваг React є реактивний підхід. Можна зосередитися на логіці додатків, не турбуючись про оновлення інтерфейсу — React робить це

автоматично, коли змінюється стан компонентів. Це сприяє швидкій розробці та легкості масштабування. Бібліотека також може використовуватися на стороні клієнта та сервера, що дозволяє створювати універсальні додатки, сприяючи кращій продуктивності та пошуковій оптимізації.

React має широку екосистему, що включає бібліотеки та інструменти, такі як Redux, React Router та інші, які допомагають створювати складніші та потужніші застосунки. Також існують бібліотеки готових стилізованих компонентів для React, такі як Material-UI.

Material-UI (MUI) [25] – це бібліотека компонентів інтерфейсу користувача для React, яка реалізує принципи дизайну Material Design, розроблені компанією Google. Вона надає широкий набір компонентів, стилів та тем, що допомагають створювати сучасні та привабливі інтерфейси для веб-додатків.

Таким чином використання мови програмування TypeScript та бібліотек React і Material-UI дозволяє розробити привабливий інтерактивний користувацький інтерфейс для клієнтського шару інформаційної системи.

2.4. Структура бази даних

Для даної інформаційної системи шар бази даних використовуватиме PostgreSQL [26] для збереження та управління даними. PostgreSQL є потужною реляційною системою управління базами даних, яка забезпечує надійне збереження даних та ефективні можливості для роботи з ними.

Реалізація онлайн додатку для навчання за допомогою карток передбачає детальне проектування та організацію бази даних. Структура цієї бази даних визначається потребами додатку, зберіганням користувальницьких даних, навчального контенту та іншою інформацією, що впливає на ефективність процесу навчання.

У розробленому онлайн додатку, кожен зареєстрований користувач отримає можливість створювати свої власні набори карток. У цих наборах

кожна картка міститиме запитання або термін з одного боку і відповідь чи пояснення до нього з іншого боку.

Ці набори карток будуть мати гнучку організацію. Вони можуть існувати окремо один від одного, дозволяючи користувачам концентруватися на конкретних темах чи завданнях. Або ж їх можна об'єднати у класи за спільною темою чи предметом, що сприяє більш системному навчанню та легшому доступу до необхідних матеріалів.

Важливою частиною інформаційної системи додатку є таблиці бази даних, де будуть зберігатися дані користувачів. Для цього буде записано логіни, хеші паролів для забезпечення безпеки та електронні адреси. Це дозволить забезпечити авторизацію користувачів та безпечний доступ до їхніх особистих навчальних матеріалів.

Ці процеси можна відобразити на DF-діаграмах (див. Рисунок 2.2 і Рисунок 2.3)

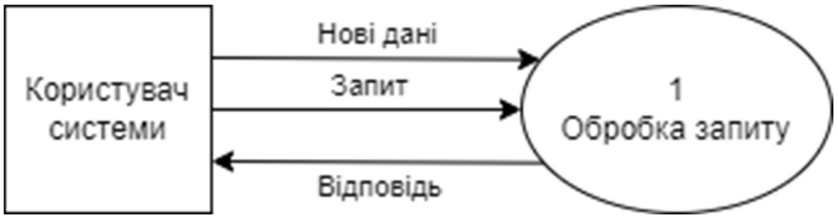

Рисунок 2.2 – DFD 0-го рівня для ІС навчання за допомогою карток

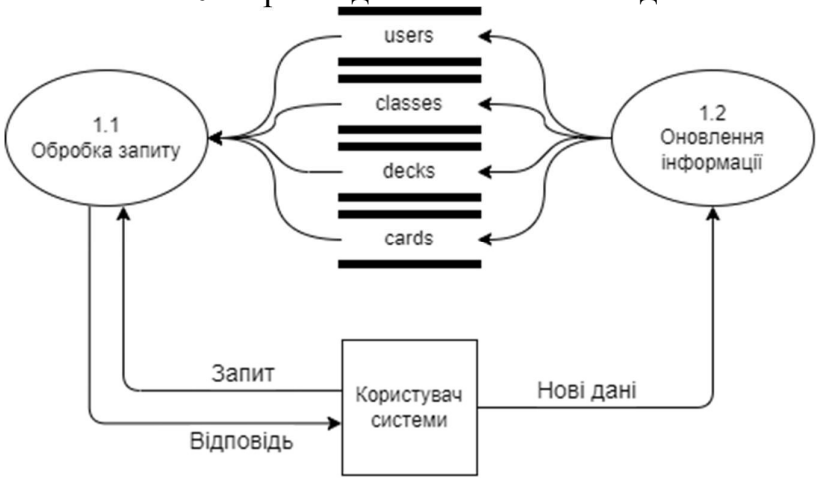

Рисунок 2.3 – DFD 1-го рівня для ІС навчання за допомогою карток

Запити які надходять від користувача поділяються на процеси «Обробка запиту», що надає необхідну запитувану інформацію і «Оновлення

інформації», який вносить зміни в сховища даних. Для обробки процесів для моделі було створено наступні таблиці: «Users», «Classes», «Decks» та «Cards».

Таблиця «Users» зберігатиме інформацію про користувачів, таку як унікальний ідентифікатор для кожного користувача, який є первинним ключем, ім’я, хеш пароля та електронну адрес. Таблиця «Classes» міститиме інформацію про класи з наборами карток, а саме ідентифікатор для кожного класу, що є первинним ключем, зовнішній ключ ідентифікатор користувача якому належить група, назву, опис, покажчик приватності класу, час створення та оновлення класу. Таблиця «Decks» необхідна для збереження інформації про набори карток та має такі поля як ідентифікатор для кожного набору, що є первинним ключем, зовнішній ключ ідентифікатор користувача якому належить набір, зовнішній ключ ідентифікатор класу до якого відноситься набір, назву, опис, покажчик приватності набору, час створення та оновлення набору. Таблиця «Cards» міститиме інформацію про картки, а саме унікальний ідентифікатор для кожної картки, зовнішній ключ ідентифікатор набору до якого відноситься картка, вміст передньої та зворотної сторін карти, посилання на зображення картки, а також час створення та оновлення картки. Зобразимо отримані таблиці та їхні відношення на ER-діаграмі (див. Рисунок 2.4).

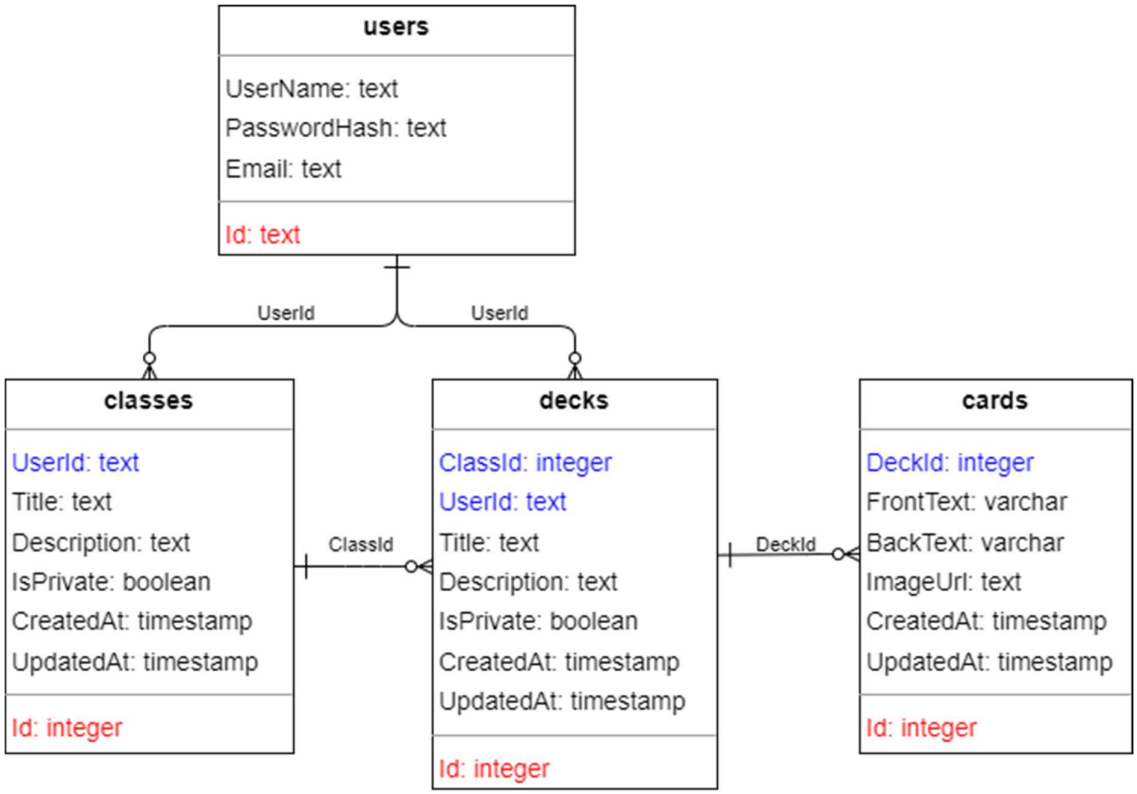

Рисунок 2.4 – ERD для ІС навчання за допомогою карток

Дана інформаційна система вже приведена до третьої нормальної форми, так як відповідає наступним вимогам [27]:

1. Перша нормальна форма (1NF): У кожній таблиці є первинний ключ, тобто стовпець з унікальними ідентифікаторами (ID). Крім того, всі атрибути кожної таблиці є атомарними, тобто не можуть бути розбиті на більш малі компоненти.
2. Друга нормальна форма (2NF): У таблицях атрибути пов'язані з первинними ключами таблиці, що забезпечує унікальність кожного запису в таблицях.
3. Третя нормальна форма (3NF): У всіх таблицях немає залежностей транзитивної функціональної залежності.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

3.1. Опис програмної реалізації серверного шару

Перед початком розробки серверного шару інформаційної системи, що буде побудований з застосуванням архітектурного підходу REST API та використанням фреймворку ASP.NET Core, було обрано середовище розробки, створено проект та додано потрібні бібліотеки та надбудови.

Для розробки даної інформаційної системи було використано редактор коду Visual Studio Code [28]. Visual Studio Code – це потужний редактор коду, який включає підтримку налагодження коду, підсвічування синтаксису, автоматичне інтелектуальне завершення коду та підтримку розширень, які додають функціональність для різних мов програмування та платформ.

Для створення нового проекту ASP.NET Core Web API необхідно створити директорію, перейти до неї та виконати команду *dotnet new webapi --use-controllers* в терміналі. Після виконання команди платформа dotnet створила набір директорій та файлів проекту, а також було додано деякі додаткові директорії для зручності.

Не менш важливим етапом створення проекту є додавання сторонніх бібліотек. Для додавання бібліотек було використано пакетний менеджер Nuget [29]. Nuget – це система керування пакетами, що призначена для розповсюдження бібліотек та інших інструментів написаних з застосуванням платформи .NET. Отже для реалізації серверного шару потрібно встановити наступні пакети:

- AutoMapper;
- AutoMapper.Extensions.Microsoft.DependencyInjection;
- Microsoft.AspNetCore.Authentication.JwtBearer;
- Microsoft.AspNetCore.Identity.EntityFrameworkCore;
- Microsoft.EntityFrameworkCore.Design;
- Microsoft.EntityFrameworkCore.Sqlite;

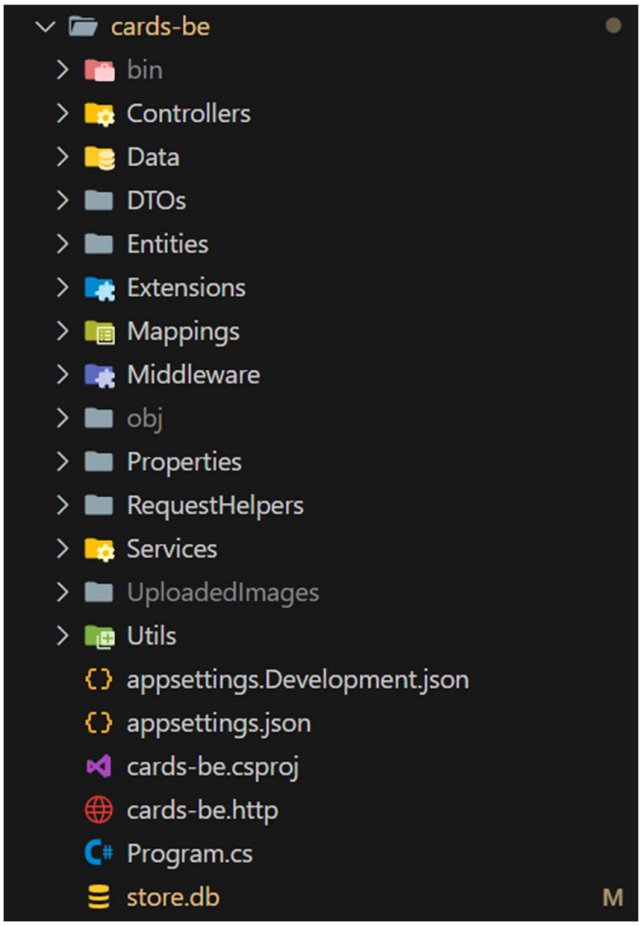

Рисунок 3.1 – структура директорій проекту серверного шару інформаційної системи

У результаті створення проекту та додавання перерахованих бібліотек маємо наступну структуру директорій (див. Рисунок 3.1):

- Controllers – директорія, що містить класи контролерів.
- Data – директорія, що містить класи контексту бази даних, класи ініціалізації бази даних та класи міграції.
- Entities – директорія з класами-сутностями.
- DTOs – директорія з класами DTO (Data transfer object)
- Extensions – директорія з класами розширеннями
- Mappings – директорія з допоміжними класами для відображення сутностей до DTO.
- Middleware – директорія з middleware.
- RequestHelpers – директорія з допоміжними класами для створення запитів.

- Services – директорія з сервісними класами.
- UploadedImages – директорія для завантажених зображень.
- Utils – директорія з іншими допоміжними класами.
- Program.cs – головний клас, який потрібен для ініціалізації та запуску серверу.
- Інші директорії та файли.

Як згадувалось раніше, для створення серверного шару даної інформаційної системи використовується фреймворк Entity Framework, що дозволяє не писати SQL запити, а представляти базу даних як набір об’єктів. Такими об’єктами є класи-сутності в директорії Entities. Всього дана інформаційна система використовує чотири класи: Card, Deck, Class та User. Кожен клас представляє рядок даних для відповідних таблиць.

Для створення бази даних заснованої на сутностях було створено класи DataContext та DbInitializer. Клас DataContext містить властивості для створення структури таблиць та зв’язків бази даних. Клас DbInitializer містить код для початкового заповнення бази даних тестовими даними.

Сутності містять багато властивостей, що відповідають за зв’язки між таблицями баз даних або є допоміжними властивостями. Такі дані зазвичай не потрібно повертати у відповідях на запити до сервера, а також відповідь з такими сутностями часто викликають помилки та циклічне виконання коду. Тому дані, які повертаються на запити повинні бути додатково сформованими у більш зручні об’єкти. Саме для таких цілей існують класи в директорії DTOs. У даній системі існує одинадцять класів DTO (Data transfer object): CardContentDto, CardDto, ClassDeckContentDto, ClassDto, ClassWithDeckDto, ContentDto, DeckDto, DeckWithCardsDto, LoginDto, RegisterDto та UserDto. Також для зручнішого формування DTOs застосовується клас MappingProfiles у директорії Mappings.

Для обробки запитів до серверу використовуються контролери. Контролери – це класи, що містять логіку для обробки запитів до серверу.

Кожен метод такого класу це endpoint, до якого є визначені URL та вхідні дані.

Контролер UserController потрібен для реєстрації, авторизації та отримання поточного авторизованого користувача (див. Таблиця 3.1).

Таблиця 3.1 – endpoint-и контролеру UserController

Маршрут	**Метод**	**Авторизований користувач**	**Вхідні дані**	**Відповідь**
1	**2**	**3**	**4**	**5**
/api/user/login	POST	Ні	Username – ім’я користувача, Password – пароль	Username – ім’я користувача, Email – електронна адреса, Token – токен
/api/user/register	POST	Ні	Username – ім’я користувача, Password – пароль, Email – електронна адреса	Username – ім’я користувача, Email – електронна адреса, Token – токен
/api/user/currentuser	GET	Так	-	Username – ім’я користувача, Email – електронна адреса, Token – токен

Контролер DecksController використовується для отримання, створення, редагування та видалення наборів карток (див. Таблиця 3.2).

Таблиця 3.2 – endpoint-и контролеру DecksController

Маршрут	**Метод**	**Авторизований користувач**	**Вхідні дані**	**Відповідь**
1	**2**	**3**	**4**	**5**
/api/decks	GET	Ні	Дані для пагінації	Масив об’єктів DeckDto
/api/decks	POST	Так	Title – заголовок набору, Description – опис набору, IsPrivate – позначення приватності набору	Об’єкт DeckDto
/api/decks/{id}	GET	Ні	Id – ідентифікатор набору карток	Об’єкт DeckWithCardsDto

Продовження табл. 3.2

1	2	3	4	5
/api/decks/{id}	PUT	Так	Id – ідентифікатор набору карток, Об’єкт ContentDto	Об’єкт DeckDto
/api/decks/{id}	DELETE	Так	Id – ідентифікатор набору карток	-
/api/decks/current-user	GET	Так	-	Масив з об’єктів DeckDto
/api/decks/assign-to-class	PUT	Так	deckId – ідентифікатор набору карток, classId – ідентифікатор класу	-
/api/decks/remove-from-class	PUT	Так	deckId – ідентифікатор набору карток, classId – ідентифікатор класу	-

Контролер CardsController потрібен для створення, редагування та видалення карток, а також для отримання зображення картки (див. Таблиця 3.3).

Таблиця 3.3 – endpoint-и контролеру CardsController

Маршрут	Метод	Авторизований користувач	Вхідні дані	Відповідь
1	2	3	4	5
/api/cards/{id}	POST	Так	Id – ідентифікатор картки, Об’єкт CardContentDto	-
/api/cards/{id}	PUT	Так	Id – ідентифікатор картки, Об’єкт CardContentDto	-
/api/cards/{id}	DELETE	Так	Id – ідентифікатор картки	-
/api/cards/image/{filename}	GET	Ні	fileName – назва файлу з зображенням	Файл зображення

Контролер ClassesController використовується для отримання, створення, редагування та видалення класів з наборами (див. Таблиця 3.4).

Таблиця 3.4 – endpoint-и контролеру ClassesController

Маршрут	**Метод**	**Авторизований користувач**	**Вхідні дані**	**Відповідь**
1	**2**	**3**	**4**	**5**
/api/classes	GET	Ні	Дані для пагінації	Масив об’єктів ClassDto
/api/classes	POST	Так	Об’єкт ContentDto	Об’єкт ClassDto
/api/classes/{id}	GET	Ні	Id – ідентифікатор класу	Об’єкт ClassWithDeckDto
/api/classes/{id}	PUT	Так	Id – ідентифікатор класу	Об’єкт ClassDto
/api/classes/{id}	DELETE	Так	Id – ідентифікатор класу	-
/api/classes/current-user	GET	Так	-	Масив об’єктів ClassDto

Контролер ProxyController потрібен для створення запитів до сторонніх ресурсів, а саме до сервісу translate.google.com, який генерує голосовий запис із поданого тексту (див. Таблиця 3.5).

Таблиця 3.5 – endpoint-и контролеру ProxyController

Маршрут	**Метод**	**Авторизований користувач**	**Вхідні дані**	**Відповідь**
1	**2**	**3**	**4**	**5**
/api/proxy/text-to-speech	GET	Ні	Text – текст для перетворення на звук, Lang – мова тексту	Файл із голосовим записом

Для забезпечення безпеки даних користувачів було використано бібліотеку Microsoft.AspNetCore.Identity, що дозволяє надійно шифрувати паролі. А також було створено клас TokenService, який побудований з використанням бібліотеки Microsoft.AspNetCore.Authentication.JwtBearer та потрібен для генерації JWT (JSON Web Tokens) [30]. Такі токени надаються

користувачам після авторизації та використовуються для підтвердження авторизації користувача в інших запитах. Основна програмна реалізація наведена в додатку до роботи (А.1 Серверний шар інформаційної системи).

3.2. Опис програмної реалізації клієнтського шару

Перед початком розробки клієнтського шару інформаційної системи, що побудований на бібліотеці React з застосуванням мови програмування TypeScript, було створено проект та додано необхідні бібліотеки.

Під час створення проекту та додавання сторонніх бібліотек було використано пакетний менеджер npm [31]. Npm (Node Package Manager) – це пакетний менеджер для мови програмування JavaScript, що дозволяє завантажувати сторонні бібліотеки та інші інструменти.

Для створення нового проекту React було використано інструмент побудови проектів Vite [32], а саме виконано команду *npm create vite@latest*. Після виконання команди інструмент Vite створив набір директорій та файлів проекту, а також було додатково додано деякі директорії для зручності.

Після отримання початкової структури проекту необхідно було встановити наступні пакети:

- @mui/material, @emotion/react, @emotion/styled – пакети бібліотеки стилізованих компонентів Material UI для React.
- @mui/lab – додатковий пакет бібліотеки Material UI, що містить компоненти в розробці.
- @mui/icons-material – додатковий пакет бібліотеки Material UI, який містить знаки та зображення.
- react-router-dom – бібліотека, яка дозволяє зручно організовувати перехід між сторінками в SPA (Single Page Application).
- Axios – бібліотека, що дозволяє зручно створювати HTTP запити, а також додавати проміжні обробки запитів.
- react-redux; @reduxjs/toolkit – бібліотека, що надає можливість управляти глобальним станом додатку.

- react-hook-form – допоміжна бібліотека для створення та простого управління формами.

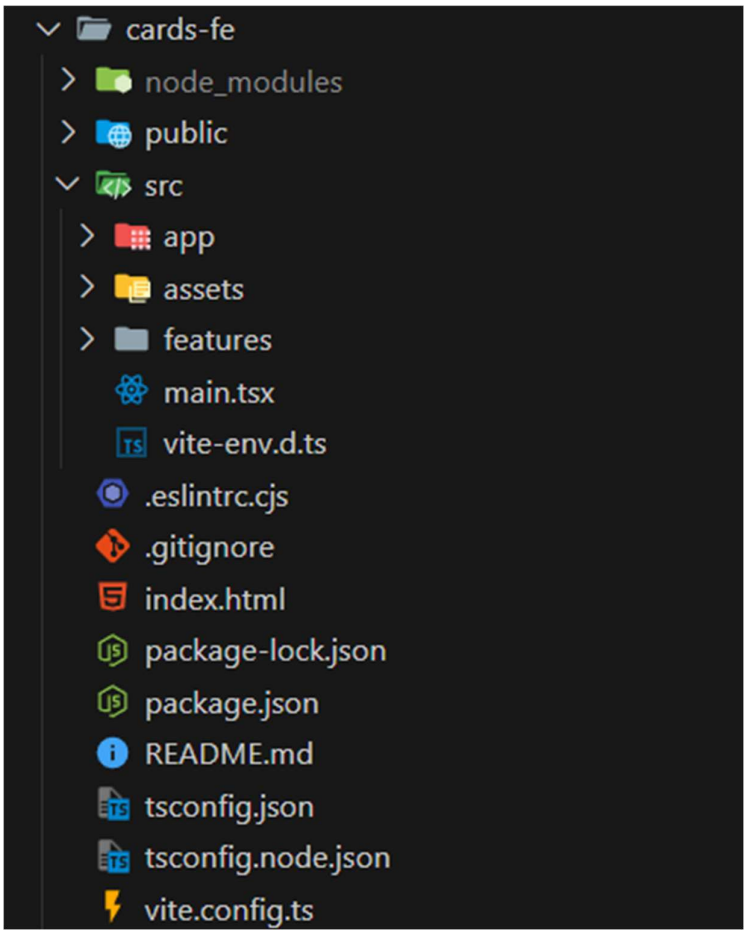

Рисунок 3.2 – структура директорій проекту клієнтського шару інформаційної системи

У результаті створення проекту та додавання перерахованих бібліотек маємо наступну структуру директорій (див Рисунок 3.2).

- node_modules – директорія зі встановленими пакетами менеджером npm.
- Public – директорія, що містить статичні файли.
- Src – директорія з кодом додатку.
    - App – директорія з основними класами, інтерфейсами та компонентами, що широко застосовуються іншими компонентами.

    - Assets – директорія з зображеннями та іншими статичними файлами.
    - Features – директорія з компонентами, що описують окремі сторінки додатку.
    - Main.tsx – головний компонент, що є коренем компонентної структури React додатку.
    - Інші файли.
- Інші файли.

Директорія app має наступну структуру:

- Api – директорія, що містить файл agent.ts, який використовує бібліотеку axios та потрібен для зручної організації запитів до серверного шару.
- Components – директорія, що містить часто використовувані компоненти, такі як AppPagination та SearchBar.
- Errors – директорія, що містить компоненти, які описують сторінки помилок, такі як NotFound та Unauthorized.
- Layout – директорія, що містить основні компоненти для кожної сторінки, такі як Header, Footer, App, LoadingComponent та SignedInMenu.
- Models – директорія з інтерфейсами, що описують об’єкти, які повертаються від серверного шару. Всього існує п’ять файлів з інтерфейсами: card, deck, studyClass, user та pagination.
- Router – директорія для управління переходу між сторінками додатку. Компонент RequireAuth потрібен для перевірки авторизації користувача, а компонент Routes потрібен для налаштування переходів між сторінками. Цей компонент використовує бібліотеку react-router-dom.
- Store – директорія, що містить файл configureStore для налаштування глобального стану додатку, а саме зберігання даних авторизованого користувача. Використання бібліотеки

Redux дозволяє зберігати та використовувати дані користувача з будь-якого компоненту додатку.

- Styles – директорія для файлів зі стилями.
- Utils – директорія з допоміжними файлами, а саме з файлом TextToSpeechHelper, який потрібен для забезпечення відтворення запису звуку з тексту.

Директорія features має наступну структуру:

- Home – директорія з компонентом HomePage.
- Search – директорія з компонентами Search, SearchPage, SearchResults, ShortLists, DeckList та ClassList.
- Account – директорія з файлом accountSlice та компонентами Profile, Register, Login та Logout.
- Dashboard – директорія з компонентом Dashboard.
- Deck – директорія з компонентами CardView, EditCardView, EditCardsPage, DeckCard, DeckCardEdit та DeckDetails.
- Class – директорія з компонентами ClassCard, ClassCardEdit, ClassDetails.

Як було описано раніше, за перехід між сторінками відповідає компонент Routes. Всього налаштовано тринадцять шляхів до сторінок.

Якщо неавторизований користувач переходить до веб-сайту даної інформаційної системи без вказання певного шляху, то він опиняється на домашній сторінці за яку відповідає компонент HomePage. На цій сторінці представлені хедер з полем пошуку наборів карток та кнопками авторизації й реєстрації, футер та власне домашня сторінка з привітанням користувача (див. Рисунок 3.3).

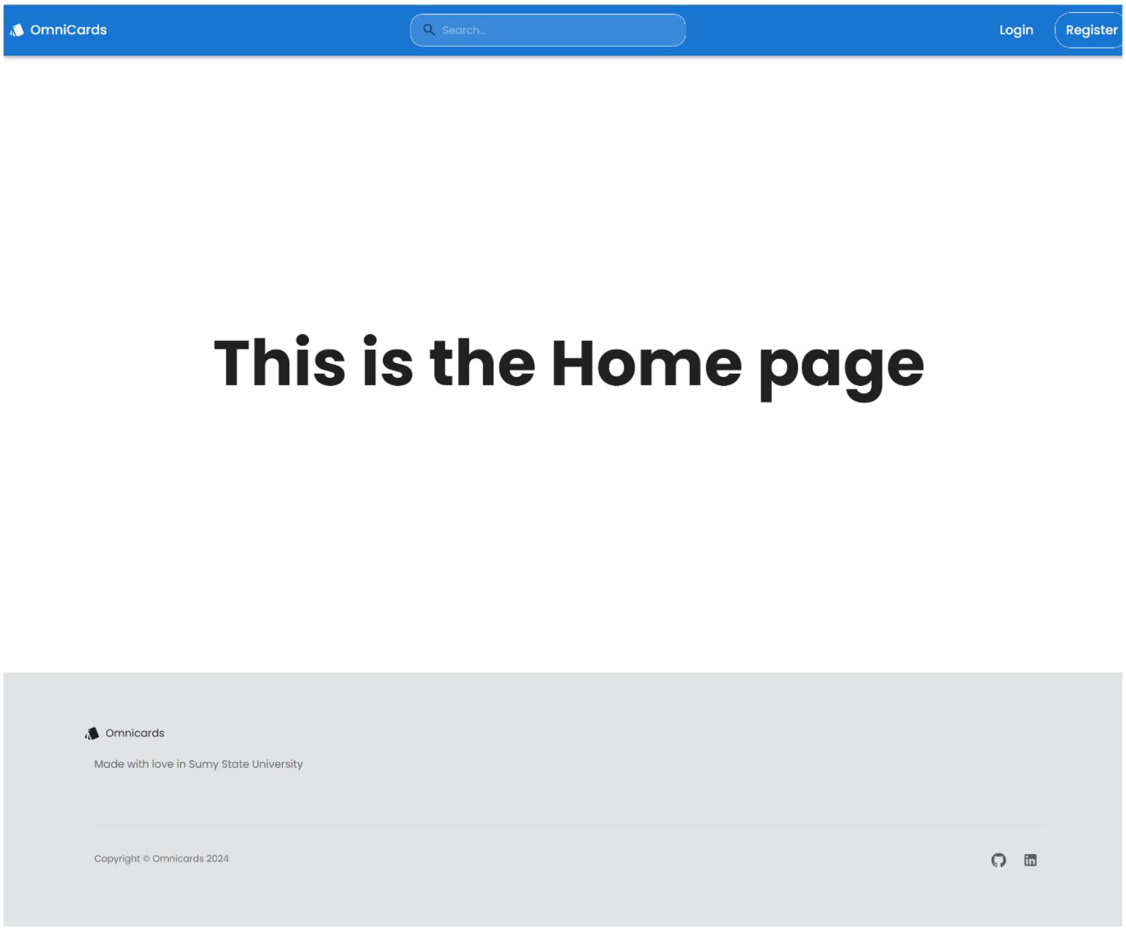

Рисунок 3.3 – вигляд домашньої сторінки

Якщо користувач введе потрібне значення в поле пошуку та натисне на відповідну кнопку, то він опиниться на сторінці пошуку за шляхом */search?q={текст для пошуку}&type={тип вкладки}*. На цій сторінці є запис про значення тексту для пошуку та три вкладки, які відповідають за різні типи результатів: всі результати пошуку, тільки результати наборів карток та тільки результати класів з наборами карток. За замовчуванням під час пошуку відкривається вкладка зі всіма результатами, де можна переглянути обмежену кількість як наборів карток так і класів з наборами (див. Рисунок 3.4). Щоб побачити всі набори карток можна перейти на вкладку з наборами карток (див. Рисунок 3.5) і відповідно, щоб побачити всі класи можна перейти на вкладку з класами (див. Рисунок 3.6). Також на вкладках з наборами карток та класами реалізовано механізм пагінації.

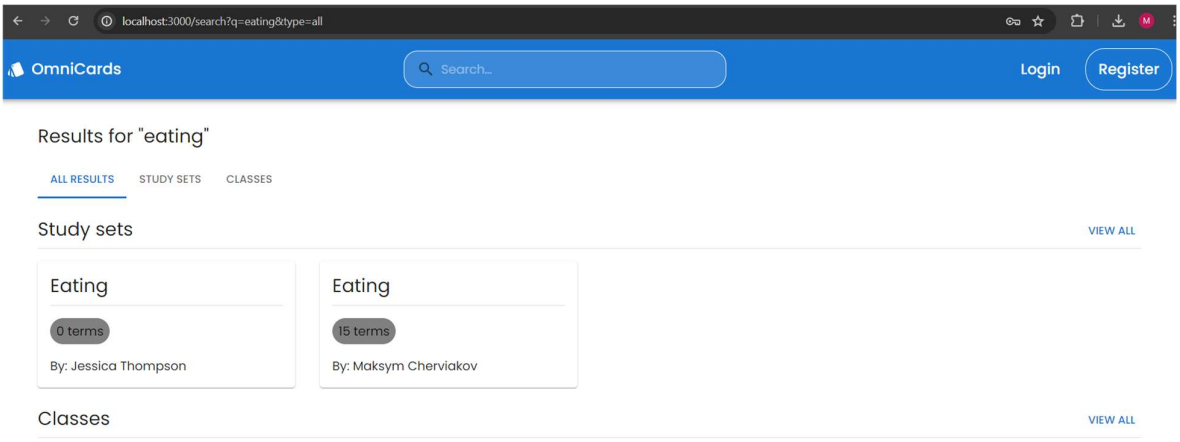

Рисунок 3.4 – вигляд сторінки пошуку з вкладкою зі всіма результатами

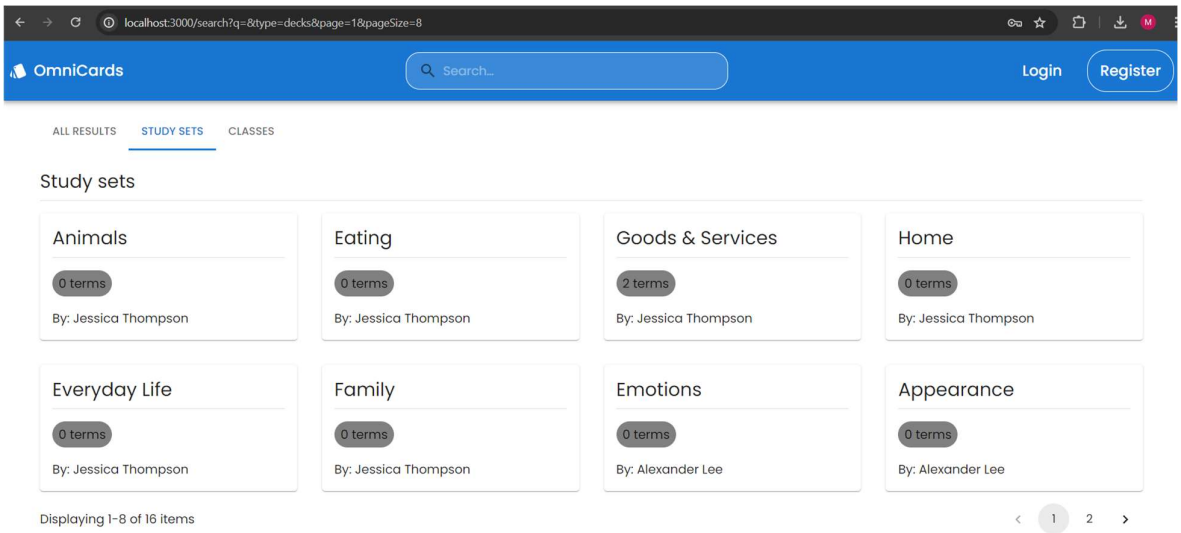

Рисунок 3.5 – вигляд сторінки пошуку з вкладкою з наборами карток

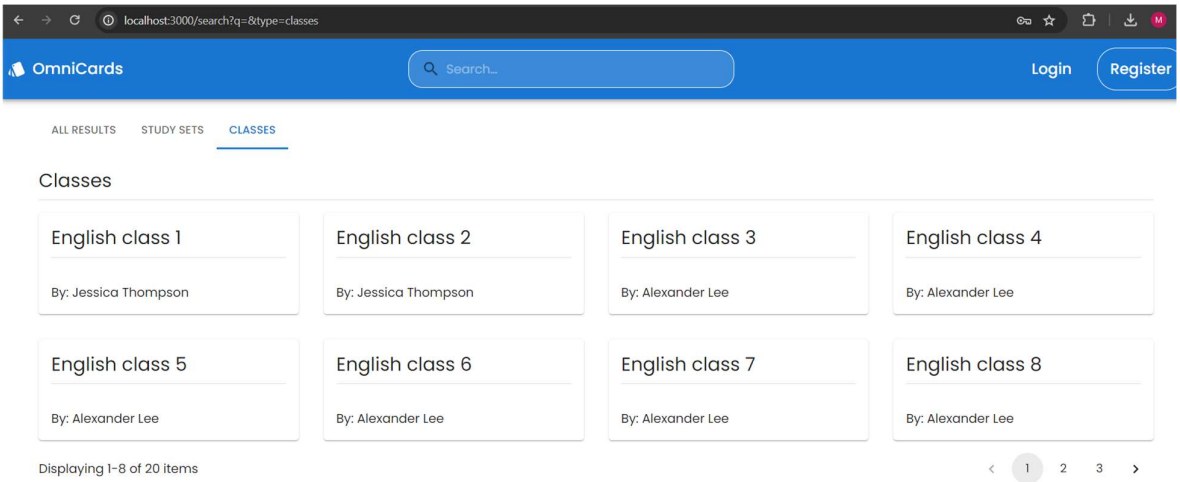

Рисунок 3.6 – вигляд сторінки пошуку з вкладкою з класами

Також, якщо користувач перейде за посиланням */search* без явного вказання запиту пошуку, то він перейде на окрему сторінку пошуку (див. Рисунок 3.7).

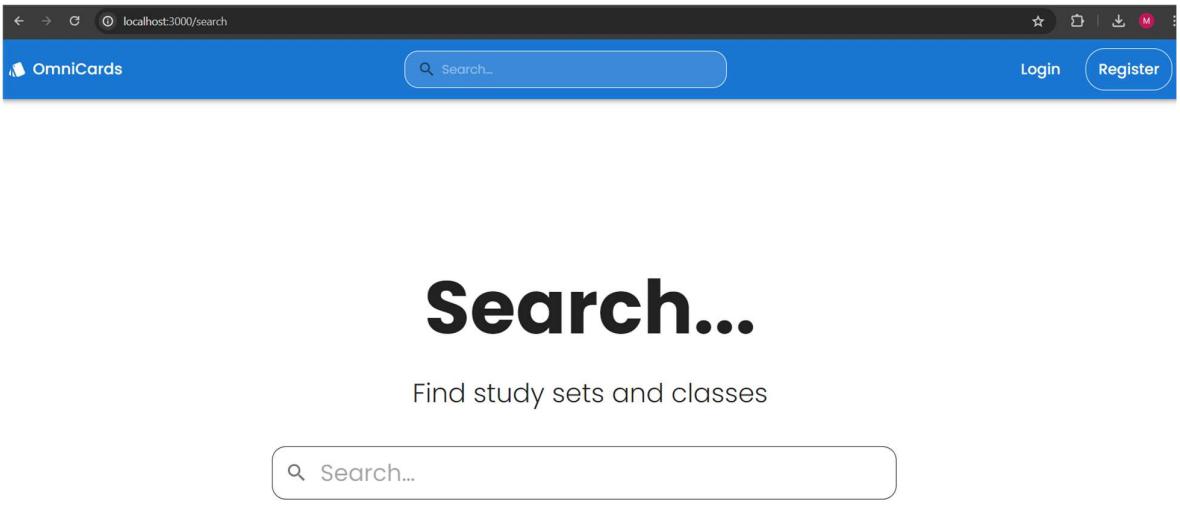

Рисунок 3.7 – вигляд окремої сторінки пошуку

Як було показано на попередніх зображеннях результати пошуку складаються з карток наборів та карток класів. Кожна картка набору складається із заголовку набору, кількості карток в наборі та запису про автора набору (див. Рисунок 3.8). Також кожна картка класу складається з заголовку набору та запису про автора (див. Рисунок 3.9).

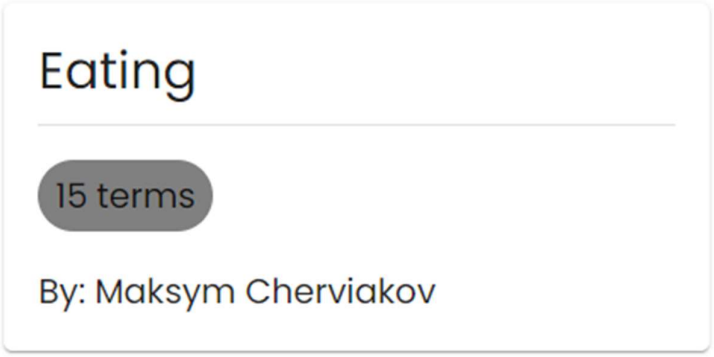

Рисунок 3.8 – картка набору

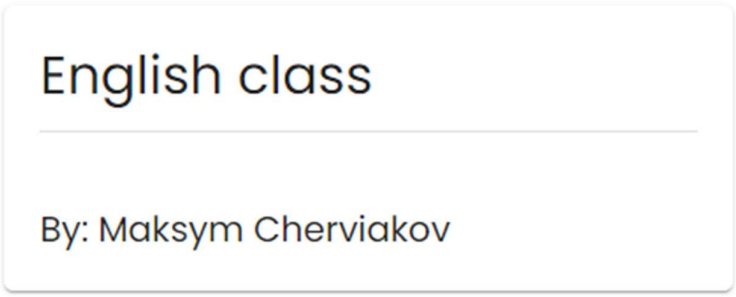

Рисунок 3.9 – картка класу

Якщо користувач натисне на набір з картками, то він опиниться на сторінці з детальним вмістом набору, де містяться заголовок набору, панель з перегортанням карток, де можна прочитати текст з передньої та задньої сторін картки, переглянути зображення картки та прослухати текст сторін, далі розміщується панель з кнопками, які мають перенаправляти користувача до сторінок з тестами, нижче розміщені записи про автора набору та короткий опис і останнім йде повний перелік карток набору з текстом сторін та зображеннями (див. Рисунок 3.10). Щоб переглянути вміст зворотної сторони картки необхідно натиснути на неї (див. Рисунок 3.11).

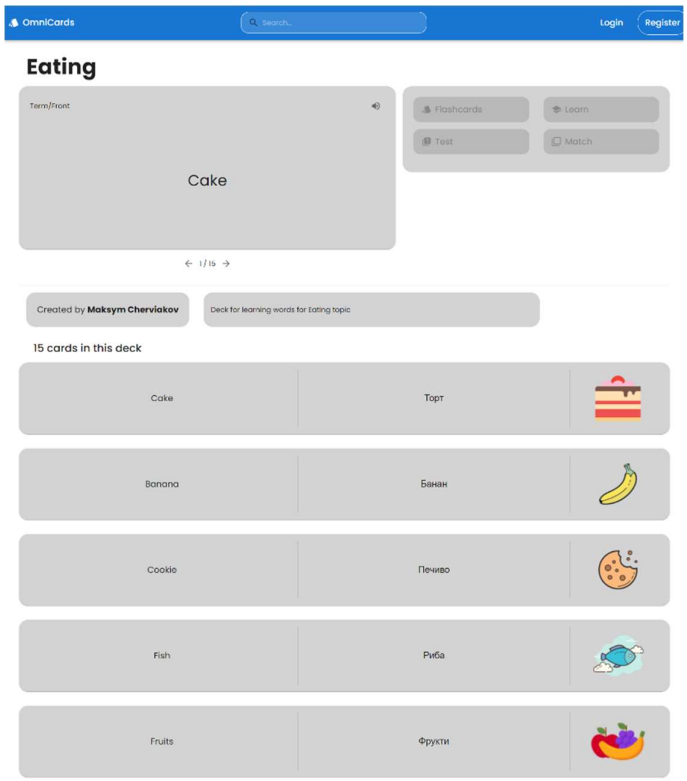

Рисунок 3.10 – вигляд сторінки детального вмісту набору карток

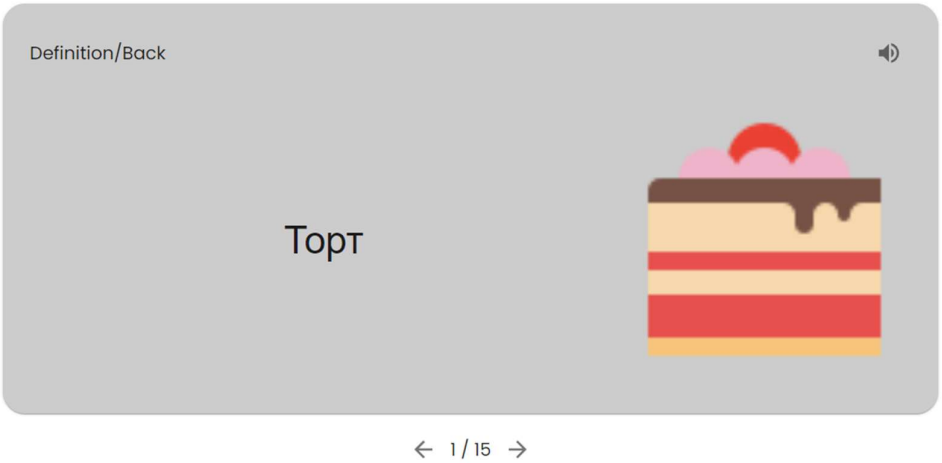

Рисунок 3.11 – вигляд зворотної сторони картки

Якщо користувач натисне на клас з наборами, то він опиниться на сторінці з детальним вмістом класу, де містяться заголовок класу, записи про автора класу, короткий опис та список наборів карток (див. Рисунок 3.12).

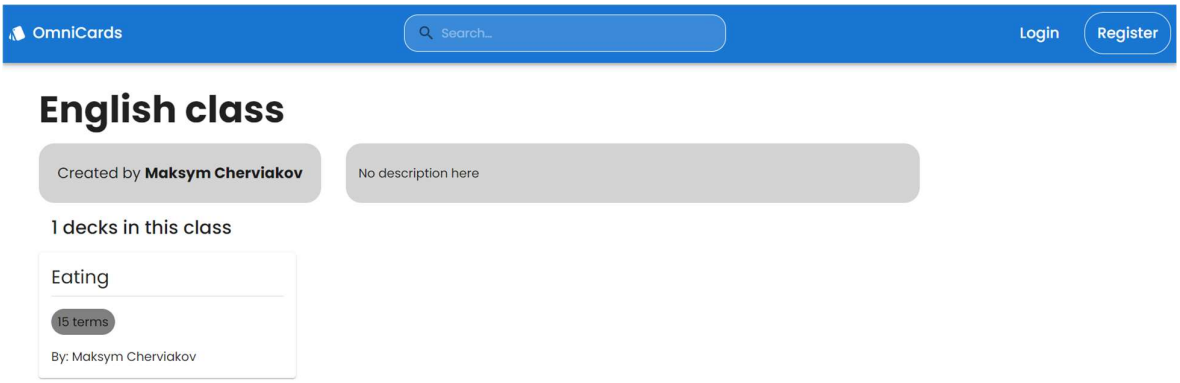

Рисунок 3.12 – вигляд сторінки детального вмісту класу з наборами карток

Неавторизовані користувачі мають змогу користуватися лише сторінками, що вказані раніше. Щоб мати можливість створювати редагувати та видаляти власні набори карток та класи, користувач може зареєструватися в інформаційній системі. Для цього потрібно перейти на сторінки реєстрації або авторизації за які відповідають компоненти Register та Login відповідно. Для реєстрації користувач має ввести ім’я, електронну адресу та пароль (див. Рисунок 3.13), а для авторизації ім’я та пароль (див. Рисунок 3.14).

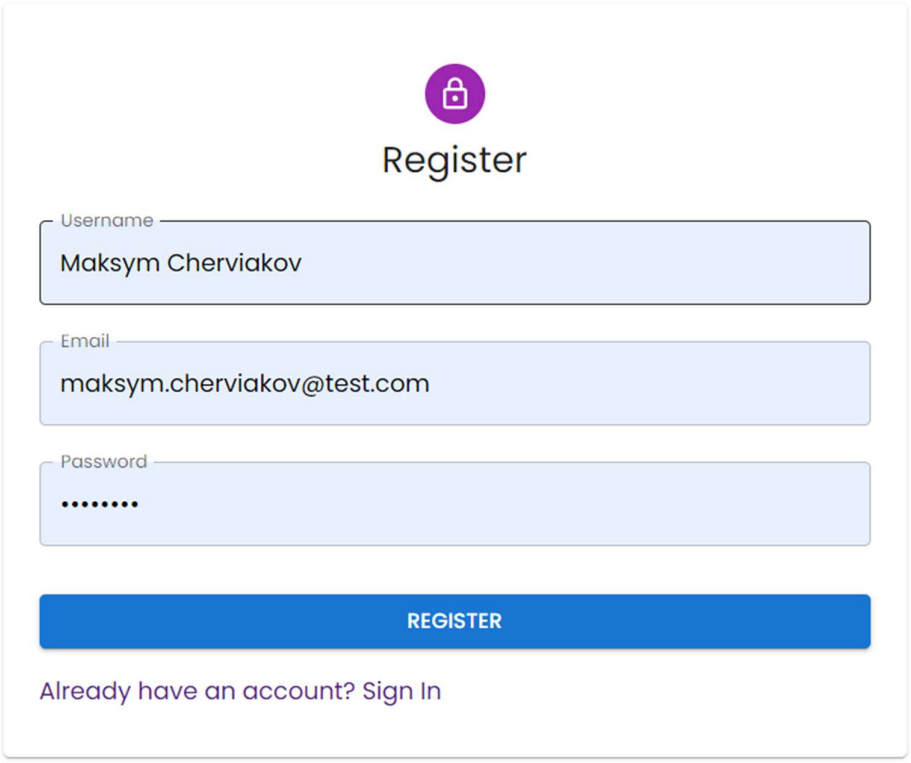

Рисунок 3.13 – вигляд сторінки реєстрації

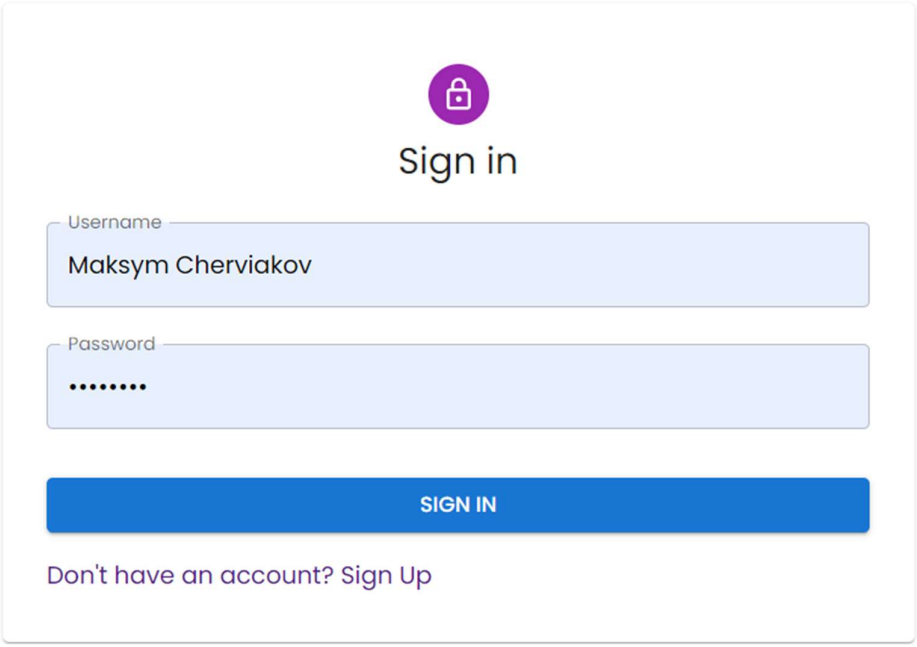

Рисунок 3.14 – вигляд сторінки авторизації

Після авторизації користувачу стає доступною сторінка з власними наборами карток та класами, де містяться список наборів карток, список класів з наборами та відповідні кнопки для додавання нових наборів та класів

(див. Рисунок 3.15). Також кожна картка з набором карток та класом містить кнопки для редагування (див. Рисунок 3.16) та видалення (див. Рисунок 3.17).

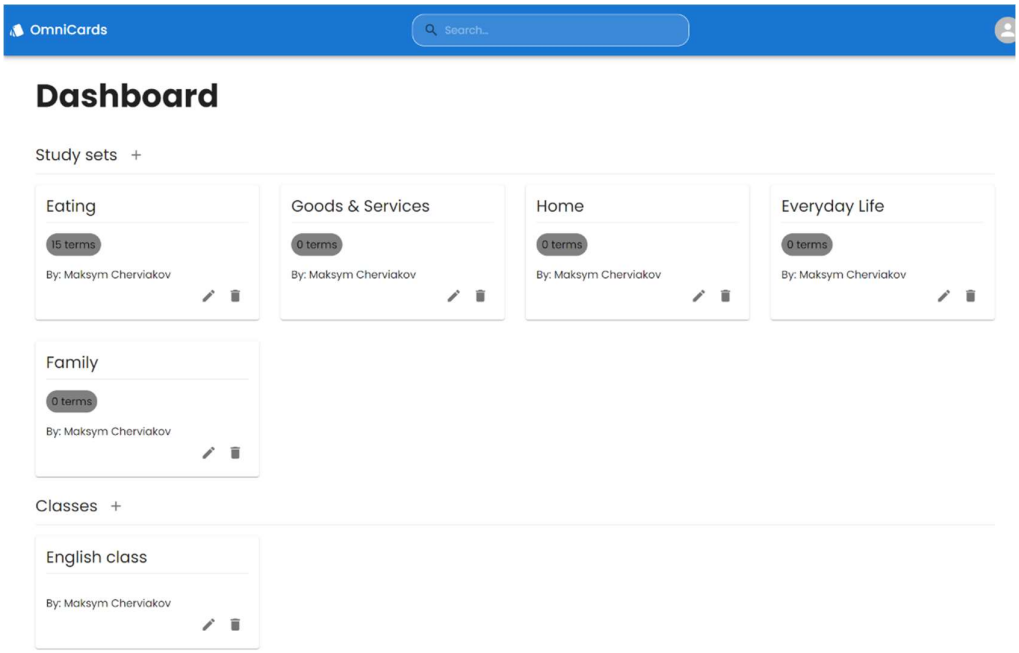

Рисунок 3.15 – вигляд сторінки з власними наборами карток та класами

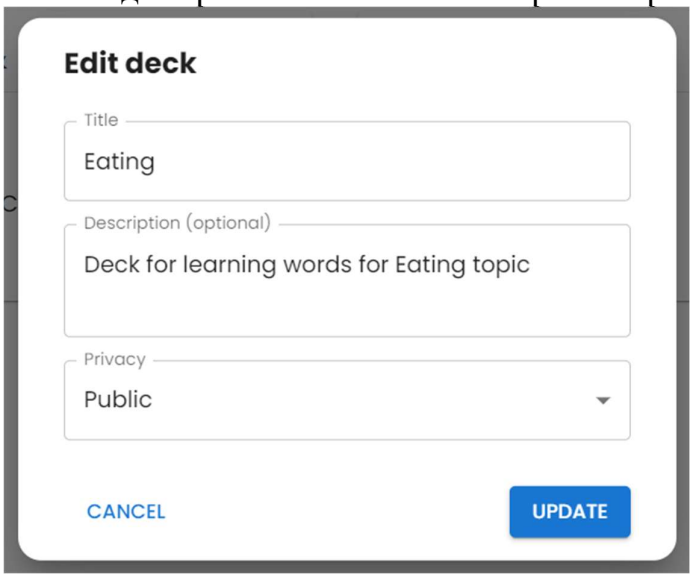

Рисунок 3.16 – модальне вікно редагування набору карток

Рисунок 3.17 – модальне вікно підтвердження видалення набору карток

Якщо авторизований користувач перейде до власного набору карток, то йому будуть доступні кнопки редагування набору та редагування карток. Кнопка редагування набору відкриває таке ж саме модальне вікно для редагування як і кнопка редагування на картці набору. А якщо користувач натисне на кнопку редагування карток, то він опиниться на сторінці редагування карток, де містяться панель для створення нової картки з полями для тексту передньої та задньої сторін картки та поле для завантаження зображення картки (див. Рисунок 3.18). Нижче знаходиться список зі всіма картками, де навпроти кожної є кнопки для редагування (див. Рисунок 3.19) та видалення карток (див. Рисунок 3.20).

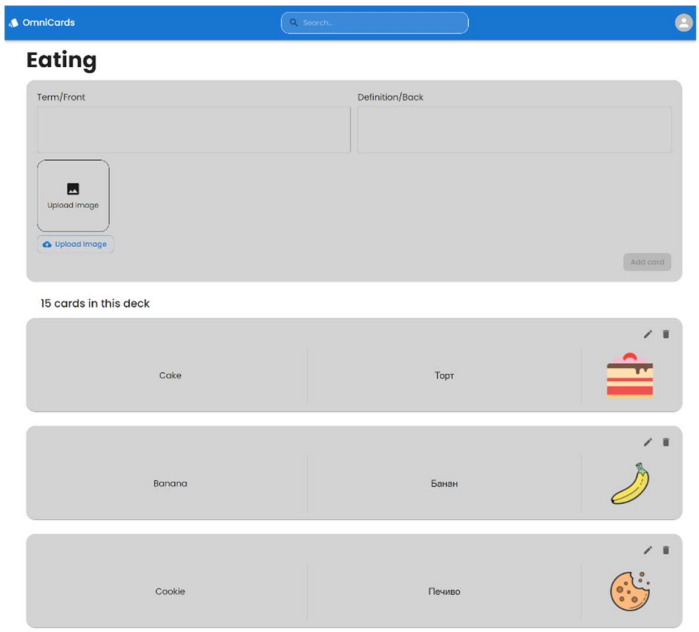

Рисунок 3.18 – вигляд сторінки редагування карток набору

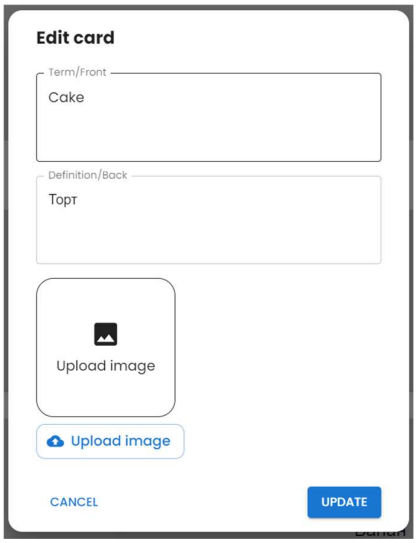

Рисунок 3.19 – модальне вікно редагування картки

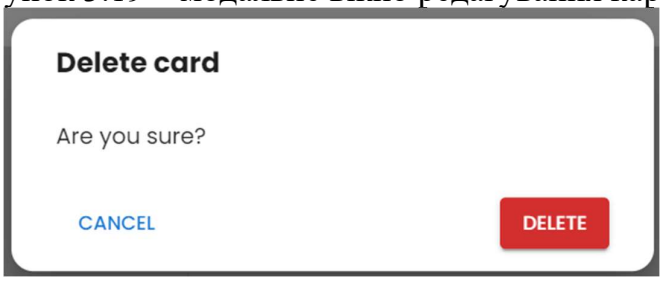

Рисунок 3.20 - модальне вікно підтвердження видалення картки

Такий набір сторінок та реалізований функціонал формують інформаційну систему, що відповідає поставленим завданням даної роботи та надає можливості аудіо-візуальної підтримки для осіб з особливими освітніми потребами. Основна програмна реалізація наведена в додатку до роботи (А.2 Клієнтський шар інформаційної системи).

3.3. Тестування

Для тестування додатку на придатність до використання особами з особливими освітніми потребами було використано розширення браузера axe DevTools [33]. Axe DevTools — це набір інструментів для тестування доступності веб-сайтів і веб-застосунків, розроблений компанією Deque Systems. Вони допомагають знаходити та виправляти проблеми з доступністю, щоб забезпечити використання веб-контенту для всіх користувачів, включаючи людей з особливими потребами.

Сторінки додатку даної інформаційної системи були протестовані розширенням браузеру axe DevTools та показали відсутність проблем з доступністю (див. Рисунок 3.21).

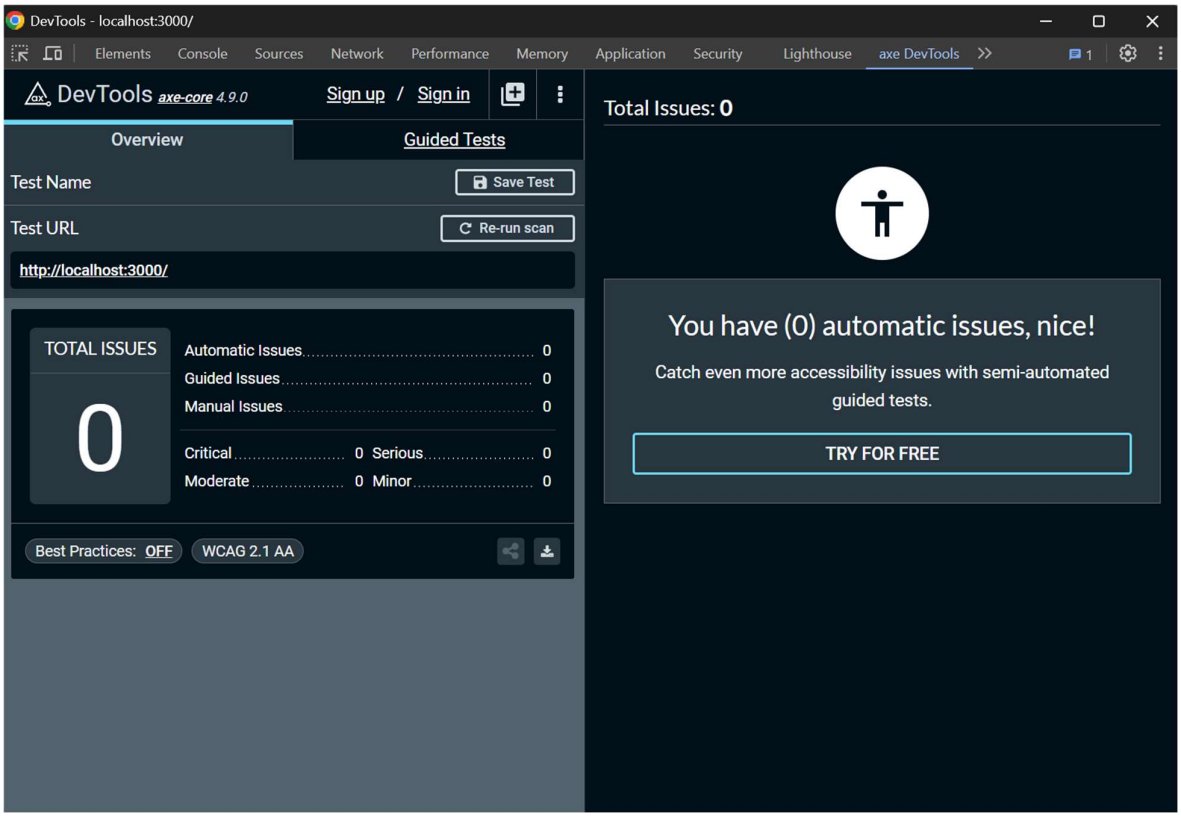

Рисунок 3.21 – вікно тестування додатку на доступність для осіб з особливими потребами

ВИСНОВКИ

У даній роботі було проведено детальний аналіз сучасних онлайн сервісів для навчання з використанням карток та визначено конкретні платформи для розробки інформаційної системи. В рамках постановки задачі була визначена мета створення інформаційної системи аудіо-візуальної підтримки процесу навчання осіб з особливими освітніми потребами з використанням карток, що спрямована на полегшення та покращення процесу навчання через зручний та інтерактивний спосіб вивчення матеріалу.

Для вирішення поставленої задачі було обрано та описано метод використання трирівневої архітектури додатку. Ця модель передбачає наявність окремих рівнів для презентаційної логіки, бізнес-логіки та доступу до даних. Такий підхід дозволяє ефективно розділити функціональні складові системи та забезпечити їх незалежний розвиток та підтримку.

В роботі була визначена необхідність зберігання та організації інформації, пов'язаної з картками, користувачами та навчальними матеріалами. Для цього була спроектована відповідна схема бази даних, що відображає зв'язки між сутностями та забезпечує зручний доступ до необхідної інформації.

Було розроблено зручний та зрозумілий інтерфейс користувача та програмно реалізовано серверну та клієнтську частини інформаційної системи.

Загалом було виконано розробку інформаційної системи, яка може ефективно підтримувати навчальний процес за допомогою карток, забезпечуючи зручний інтерфейс для користувачів та можливість вивчення матеріалу в інтерактивному форматі. Реалізація обраної архітектурної моделі та бази даних може послужити стійкою основою для подальшого розвитку системи та покращення якості навчання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Improving Students' Learning With Effective Learning Techniques / J. Dunlosky та ін. Psychological Science in the Public Interest. 2013. Т. 14, № 1. С. 4–58. URL: https://doi.org/10.1177/1529100612453266 (дата звернення: 30.04.2024).
2. Quizlet. Quizlet. URL: https://quizlet.com (дата звернення: 17.07.2023).
3. Anki - powerful, intelligent flashcards. Anki - powerful, intelligent flashcards. URL: https://apps.ankiweb.net/ (дата звернення: 30.04.2024).
4. The ultimate flashcard tool | OmniSets.com. The ultimate flashcard tool | OmniSets.com. URL: https://www.omnisets.com/ (дата звернення: 30.04.2024).
5. Brainscape: The Best Flashcards App | Make Flashcards Online. Brainscape: The Best Flashcards App | Make Flashcards Online. URL: https://www.brainscape.com (дата звернення: 30.04.2024).
6. What is a Dedicated Server? - Dedicated Server Hosting Explained - AWS. Amazon Web Services, Inc. URL: https://aws.amazon.com/what-is/dedicated-server/ (дата звернення: 30.04.2024).
7. What is VPS? - VPS Hosting and Server Explained - AWS. Amazon Web Services, Inc. URL: https://aws.amazon.com/what-is/vps/ (дата звернення: 30.04.2024).
8. What is Shared Hosting and how does it work?. Namecheap. URL: https://www.namecheap.com/hosting/what-is-shared-hosting-definition/ (дата звернення: 30.04.2024).
9. What is cloud hosting. IBM - Deutschland | IBM. URL: https://www.ibm.com/cloud/learn/what-is-cloud-hosting (дата звернення: 30.04.2024).
10. Cloud Computing Services | Microsoft Azure. Microsoft Azure. URL: https://azure.microsoft.com (дата звернення: 30.04.2024).

11. Virtual machines in Azure - Azure Virtual Machines. Microsoft Learn. URL: https://learn.microsoft.com/en-us/azure/virtual-machines/ (дата звернення: 30.04.2024).
12. [Azure VM architecture diagram]. Microsoft Learn. URL: https://learn.microsoft.com/en-us/azure/architecture/reference-architectures/n-tier/images/single-vm-diagram.svg (дата звернення: 30.04.2024).
13. Web Application Architecture. MobiDev. URL: https://mobidev.biz/blog/web-application-architecture-types (дата звернення: 30.04.2024).
14. What is Three-Tier Architecture | IBM. IBM - Deutschland | IBM. URL: https://www.ibm.com/topics/three-tier-architecture (дата звернення: 30.04.2024).
15. [Діаграма трирівневої архітектури]. Mobidev. URL: https://mobidev.biz/wp-content/uploads/2021/07/3-tier-web-architecture.jpg (дата звернення: 30.04.2024).
16. What is a REST API? | IBM. IBM in Deutschland, Österreich und der Schweiz. URL: https://www.ibm.com/topics/rest-apis (дата звернення: 30.04.2024).
17. C# Guide - .NET managed language. Microsoft Learn. URL: https://learn.microsoft.com/en-us/dotnet/csharp/ (дата звернення: 30.04.2024).
18. ASP.NET | Open-source web framework for .NET. Microsoft. URL: https://dotnet.microsoft.com/en-us/apps/aspnet (дата звернення: 30.04.2024).
19. Entity Framework documentation hub. Microsoft Learn. URL: https://learn.microsoft.com/en-us/ef/ (дата звернення: 30.04.2024).
20. JavaScript | MDN. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (дата звернення: 30.04.2024).
21. JavaScript With Syntax For Types. TypeScript: JavaScript With Syntax For Types. URL: https://www.typescriptlang.org/ (дата звернення: 30.04.2024).
22. HTML: HyperText Markup Language | MDN. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/HTML (дата звернення: 30.04.2024).

23. CSS: Cascading Style Sheets | MDN. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/CSS (дата звернення: 30.04.2024).
24. React. React. URL: https://react.dev/ (дата звернення: 30.04.2024).
25. MUI: The React component library you always wanted. MUI: The React component library you always wanted. URL: https://mui.com/ (дата звернення: 30.04.2024).
26. PostgreSQL. PostgreSQL. URL: https://www.postgresql.org/ (дата звернення: 30.04.2024).
27. Normal Forms in DBMS - GeeksforGeeks. GeeksforGeeks. URL: https://www.geeksforgeeks.org/normal-forms-in-dbms/ (дата звернення: 30.04.2024).
28. Visual Studio Code - Code Editing. Redefined. *Visual Studio Code - Code Editing. Redefined*. URL: https://code.visualstudio.com/ (дата звернення: 30.04.2024).
29. NuGet Gallery | Home. NuGet Gallery | Home. URL: https://www.nuget.org/ (дата звернення: 30.04.2024).
30. JWT.IO - JSON Web Tokens Introduction. JSON Web Tokens - jwt.io. URL: https://jwt.io/introduction (дата звернення: 30.04.2024).
31. npm | Home. npm | Home. URL: https://www.npmjs.com/ (дата звернення: 30.04.2024).
32. Vite. Vite | Next Generation Frontend Tooling. URL: https://vitejs.dev/ (дата звернення: 30.04.2024).
33. axe DevTools - Web Accessibility Testing. Chrome Web Store. URL: https://chromewebstore.google.com/detail/axe-devtools-web-accessib/lhdoppojpmngadmnindnejefpokejbdd (дата звернення: 30.04.2024).

ДОДАТОК А

А.1 Серверний шар інформаційної системи

User.cs

```
namespace API.Entities;

public class User : IdentityUser
{
    // navigation properties
    public List<Class> Classes { get; set; } = [];
    public List<Deck> Decks { get; set; } = [];
}
```

Card.cs

```
namespace API.Entities;

public class Card
{
    public int Id { get; set; }
    public string FrontText { get; set; }
    public string BackText { get; set; }
    public string? ImageUrl { get; set; }
    public DateTime CreatedAt { get; set; }
    public DateTime UpdatedAt { get; set; }

    // navigation properties
    public int DeckId { get; set; }
    public Deck Deck { get; set; }
}
```

Deck.cs

```
namespace API.Entities;

public class Deck
{
    public int Id { get; set; }
    public string Title { get; set; }
    public string? Description { get; set; }
    public bool IsPrivate { get; set; }
    public DateTime CreatedAt { get; set; }
    public DateTime UpdatedAt { get; set; }

    // navigation properties
    public string UserId { get; set; }
```

```
    public User User { get; set; }
    public int? ClassId { get; set; }
    public Class? Class { get; set; }

    public List<Card> Cards { get; set; } = [];
}
```

Class.cs

```
namespace API.Entities;

public class Class
{
    public int Id { get; set; }
    public string Title { get; set; }
    public string? Description { get; set; }
    public bool IsPrivate { get; set; }
    public DateTime CreatedAt { get; set; }
    public DateTime UpdatedAt { get; set; }

    // navigation properties
    public string UserId { get; set; }
    public User User { get; set; }
    public List<Deck> Decks { get; set; } = [];
}
```

DataContext.cs

```
namespace API.Data;

public class DataContext : IdentityDbContext<User>
{
    public DataContext(DbContextOptions options) : base(options)
    {
    }

    public DbSet<Card> Cards { get; set; }
    public DbSet<Deck> Decks { get; set; }
    public DbSet<Class> Classes { get; set; }

    protected override void OnModelCreating(ModelBuilder builder)
    {
        base.OnModelCreating(builder);
        builder.Entity<IdentityRole>()
            .HasData(
                new IdentityRole { Name = "Member", NormalizedName = "MEMBER" },
                new IdentityRole { Name = "Admin", NormalizedName = "ADMIN" }
            );
    }
}
```

TokenService.cs

```
namespace API.Services;

public class TokenService
{
    private readonly UserManager<User> _userManager;
    private readonly IConfiguration _config;

    public TokenService(UserManager<User> userManager, IConfiguration config)
    {
        _userManager = userManager;
        _config = config;
    }

    public async Task<string> GenerateToken(User user)
    {
        var claims = new List<Claim>
        {
            new Claim(ClaimTypes.Email, user.Email),
            new Claim(ClaimTypes.Name, user.UserName),
        };

        var roles = await _userManager.GetRolesAsync(user);
        foreach (var role in roles)
        {
            claims.Add(new Claim(ClaimTypes.Role, role));
        }

        var key = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(_config["JWTSettings:TokenKey"]));
        var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha512);

        var tokenOptions = new JwtSecurityToken(
            issuer: null,
            audience: null,
            claims: claims,
            expires: DateTime.Now.AddDays(7),
            signingCredentials: creds
        );

        return new JwtSecurityTokenHandler().WriteToken(tokenOptions);
    }
}
```

BaseApiController.cs

```
namespace API.Controllers;

[ApiController]
[Route("api/[controller]")]
public class BaseApiController : ControllerBase { }
```

UserController.cs

```
namespace API.Controllers;

public class UserController : BaseApiController
{
    private readonly UserManager<User> _userManager;
    private readonly TokenService _tokenService;

    public UserController(UserManager<User> userManager, TokenService
tokenService)
    {
        _userManager = userManager;
        _tokenService = tokenService;
    }

    [HttpPost("login")]
    public async Task<ActionResult<UserDto>> Login(LoginDto loginDto)
    {
        var user = await _userManager.FindByNameAsync(loginDto.Username);

        if (user == null || !await _userManager.CheckPasswordAsync(user,
loginDto.Password))
            return Unauthorized();

        return new UserDto
        {
            UserName = user.UserName,
            Email = user.Email,
            Token = await _tokenService.GenerateToken(user)
        };
    }

    [HttpPost("register")]
    public async Task<ActionResult<UserDto>> Register(RegisterDto registerDto)
    {
        var user = new User { UserName = registerDto.Username, Email =
registerDto.Email };

        var result = await _userManager.CreateAsync(user, registerDto.Password);

        if (!result.Succeeded)
        {
            foreach (var error in result.Errors)
            {
                ModelState.AddModelError(error.Code, error.Description);
            }

            return ValidationProblem();
        }
```

```
            await _userManager.AddToRoleAsync(user, "Member");

            return new UserDto
            {
                UserName = user.UserName,
                Email = user.Email,
                Token = await _tokenService.GenerateToken(user)
            };
        }

        [Authorize]
        [HttpGet("currentUser")]
        public async Task<ActionResult<UserDto>> GetCurrentUser()
        {
            var user = await _userManager.FindByNameAsync(User.Identity.Name);

            return new UserDto
            {
                UserName = user.UserName,
                Email = user.Email,
                Token = await _tokenService.GenerateToken(user)
            };
        }
    }
```

CardsController.cs

```
namespace API.Controllers;

public class CardsController : BaseApiController
{
    private readonly DataContext _context;
    private readonly UserManager<User> _userManager;

    public CardsController(DataContext context, UserManager<User> userManager)
    {
        _context = context;
        _userManager = userManager;
    }

    [Authorize]
    [HttpPost("{id}")]
    public async Task<ActionResult> CreateCard(int id, CardContentDto cardDto)
    {
        // get user
        var user = await _userManager.FindByNameAsync(User.Identity?.Name);
        if (user == null) return Unauthorized();
        // get deck
        if (await _context.Decks.FindAsync(id) == null) return NotFound();

        var deck = await _context.Decks
```

```
            .Include(item => item.User)
            .Include(item => item.Cards)
            .FirstAsync(item => item.Id == id);

        if (deck == null) return NotFound();
        // check user
        if (deck.User.UserName != User.Identity?.Name) return Unauthorized();
        // create card
        Card card = null;
        try
        {
            card = new Card
            {
                FrontText = cardDto.FrontText,
                BackText = cardDto.BackText,
                ImageUrl = await UploadHelper.UploadImage(cardDto.Image),
                CreatedAt = DateTime.Now,
                UpdatedAt = DateTime.Now
            };
        }
        catch (Exception ex)
        {
            return BadRequest(new ProblemDetails { Title = ex.Message });
        }
        // add card
        deck.Cards.Add(card);
        // save
        var result = await _context.SaveChangesAsync() > 0;

        if (result) return Ok();

        return BadRequest(new ProblemDetails { Title = "Problem with saving card"
});
    }

    [Authorize]
    [HttpPut("{id}")]
    public async Task<ActionResult> UpdateCard(int id, CardContentDto cardDto)
    {
        // get user
        var user = await _userManager.FindByNameAsync(User.Identity?.Name);
        if (user == null) return Unauthorized();
        // get card
        if (await _context.Cards.FindAsync(id) == null) return NotFound();

        var card = await _context.Cards
            .Include(item => item.Deck)
            .Include(item => item.Deck.User)
            .FirstAsync(item => item.Id == id);

        if (card == null) return NotFound();
```

```
        // check user
        if (card.Deck.User.UserName != User.Identity?.Name) return
Unauthorized();
        // update card
        var oldImageUrl = card.ImageUrl;
        try
        {
            if (!cardDto.FrontText.IsNullOrEmpty())
                card.FrontText = cardDto.FrontText;
            if (!cardDto.BackText.IsNullOrEmpty())
                card.BackText = cardDto.BackText;
            card.UpdatedAt = DateTime.Now;

            var newImageUrl = await UploadHelper.UploadImage(cardDto.Image);
            if (newImageUrl != null)
            {
                card.ImageUrl = newImageUrl;
                UploadHelper.DeleteFile(oldImageUrl);
            }

        }
        catch (Exception ex)
        {
            return BadRequest(new ProblemDetails { Title = ex.Message });
        }
        // save
        var result = await _context.SaveChangesAsync() > 0;

        if (result) return Ok();

        return BadRequest(new ProblemDetails { Title = "Problem with saving card"
});
    }

    [Authorize]
    [HttpDelete("{id}")]
    public async Task<ActionResult> DeleteCard(int id)
    {
        // get user
        var user = await _userManager.FindByNameAsync(User.Identity?.Name);
        if (user == null) return Unauthorized();
        // get card
        if (await _context.Cards.FindAsync(id) == null) return NotFound();

        var card = await _context.Cards
            .Include(item => item.Deck)
            .Include(item => item.Deck.User)
            .FirstAsync(item => item.Id == id);

        if (card == null) return NotFound();
        var deck = card.Deck;
```

```
        // check user
        if (card.Deck.User.UserName != User.Identity?.Name) return
Unauthorized();
        // delete card
        deck.Cards.Remove(card);
        // save
        var result = await _context.SaveChangesAsync() > 0;

        if (result) return Ok();

        return BadRequest(new ProblemDetails { Title = "Problem with deleting
card" });
    }

    [HttpGet("image/{fileName}")]
    public ActionResult GetCardImage(string fileName)
    {
        if (!string.IsNullOrEmpty(fileName))
        {
            try
            {
                FileStream fileStream = UploadHelper.GetFileStream(fileName);
                return File(fileStream, "image/*");
            }
            catch (Exception ex)
            {
                return BadRequest(new ProblemDetails { Title = ex.Message });
            }
        }

        return BadRequest(new ProblemDetails { Title = "Url is empty" });
    }
}
```

DecksController.cs

```
namespace API.Controllers;

public class DecksController : BaseApiController
{
    private readonly DataContext _context;
    private readonly IMapper _mapper;
    private readonly UserManager<User> _userManager;

    public DecksController(DataContext context, IMapper mapper, UserManager<User>
userManager)
    {
        _context = context;
        _mapper = mapper;
        _userManager = userManager;
    }
```

```
    [HttpGet]
    public async Task<ActionResult<PagedList<DeckDto>>> GetDecks([FromQuery]
QueryParams queryParams)
    {
        var username = User.Identity?.Name;
        var user = username != null ? await
_userManager.FindByNameAsync(username) : null;
        var userId = user?.Id;
        var query = _context.Decks
            .Search(queryParams.SearchTerm)
            .Include(item => item.User)
            .Include(item => item.Cards)
            .Where(items => items.IsPrivate == false || items.UserId == userId)
            .OrderBy(item => item.Id)
            .Select(items => _mapper.Map<DeckDto>(items))
            .AsQueryable();

        var decks = await PagedList<DeckDto>.ToPagedList(query,
queryParams.PageNumber, queryParams.PageSize);

        Response.AddPaginationHeader(decks.MetaData);

        return decks;
    }

    [HttpGet("{id}", Name = "GetDeck")]
    public async Task<ActionResult<DeckWithCardsDto>> GetDeck(int id)
    {
        if (await _context.Decks.FindAsync(id) == null) return NotFound();

        var deck = await _context.Decks
            .Include(item => item.User)
            .Include(item => item.Cards)
            .FirstAsync(item => item.Id == id);

        if (deck == null) return NotFound();

        if (deck.IsPrivate && deck.User.UserName != User.Identity?.Name) return
Unauthorized();

        return _mapper.Map<DeckWithCardsDto>(deck);
    }

    [Authorize]
    [HttpGet("current-user")]
    public async Task<ActionResult<List<DeckDto>>> GetDecksForCurrentUser()
    {
        // get user
        var user = await _userManager.FindByNameAsync(User.Identity.Name);
        if (user == null) return Unauthorized();
```

```
        // get decks for user
        var decks = await _context.Decks
            .Include(item => item.User)
            .Include(item => item.Cards)
            .Where(item => item.UserId == user.Id)
            .Select(items => _mapper.Map<DeckDto>(items))
            .ToListAsync();

        return decks;
    }

    [Authorize]
    [HttpPost]
    public async Task<ActionResult<DeckDto>> CreateDeck(ContentDto deckDto)
    {
        // get user
        var user = await _userManager.FindByNameAsync(User.Identity.Name);
        if (user == null) return Unauthorized();
        // create deck
        var deck = new Deck
        {
            Title = deckDto.Title,
            Description = deckDto.Description,
            IsPrivate = deckDto.IsPrivate,
            CreatedAt = DateTime.Now,
            UpdatedAt = DateTime.Now
        };
        // add deck
        user.Decks.Add(deck);
        // save
        var result = await _context.SaveChangesAsync() > 0;

        if (result) return CreatedAtRoute("GetDeck", new { id = deck.Id },
_mapper.Map<DeckDto>(deck));

        return BadRequest(new ProblemDetails { Title = "Problem with saving deck"
});
    }

    [Authorize]
    [HttpPut("{id}")]
    public async Task<ActionResult<DeckDto>> UpdateDeck(int id, ContentDto
deckDto)
    {
        // get user
        var user = await _userManager.FindByNameAsync(User.Identity?.Name);
        if (user == null) return Unauthorized();
        // get deck
        if (await _context.Decks.FindAsync(id) == null) return NotFound();

        var deck = await _context.Decks
```

```
            .Include(item => item.User)
            .FirstAsync(item => item.Id == id);
        if (deck == null) return NotFound();
        if (deck.User?.UserName != User.Identity?.Name) return Unauthorized();
        // update deck
        deck.Title = deckDto.Title;
        deck.Description = deckDto.Description;
        deck.IsPrivate = deckDto.IsPrivate;
        deck.UpdatedAt = DateTime.Now;
        // save
        var result = await _context.SaveChangesAsync() > 0;
        if (result) return _mapper.Map<DeckDto>(deck);

        return BadRequest(new ProblemDetails { Title = "Problem with updating
deck" });
    }

    [Authorize]
    [HttpPut("assign-to-class")]
    public async Task<ActionResult> AssignClass(ClassDeckContentDto
classDeckContentDto)
    {
        // get user
        var user = await _userManager.FindByNameAsync(User.Identity?.Name);
        if (user == null) return Unauthorized();
        // get deck
        if (await _context.Decks.FindAsync(classDeckContentDto.deckId) == null)
return NotFound();

        var deck = await _context.Decks
            .Include(item => item.User)
            .FirstAsync(item => item.Id == classDeckContentDto.deckId);
        if (deck == null) return NotFound();
        if (deck.User?.UserName != User.Identity?.Name) return Unauthorized();
        // get class
        if (await _context.Classes.FindAsync(classDeckContentDto.classId) ==
null) return NotFound();

        var studyClass = await _context.Classes
            .Include(item => item.User)
            .FirstAsync(item => item.Id == classDeckContentDto.classId);
        if (studyClass == null) return NotFound();
        if (studyClass.User?.UserName != User.Identity?.Name) return
Unauthorized();
        // assign class to deck
        deck.Class = studyClass;
        // save
        var result = await _context.SaveChangesAsync() > 0;
        if (result) return Ok("Class has been assigned to deck");
```

```
        return BadRequest(new ProblemDetails { Title = "Problem with assigning
class to deck" });
    }

    [Authorize]
    [HttpPut("remove-from-class")]
    public async Task<ActionResult> RemoveClass(ClassDeckContentDto
classDeckContentDto)
    {
        // get user
        var user = await _userManager.FindByNameAsync(User.Identity?.Name);
        if (user == null) return Unauthorized();
        // get deck
        if (await _context.Decks.FindAsync(classDeckContentDto.deckId) == null)
return NotFound();

        var deck = await _context.Decks
            .Include(item => item.User)
            .FirstAsync(item => item.Id == classDeckContentDto.deckId);
        if (deck == null) return NotFound();
        if (deck.User?.UserName != User.Identity?.Name) return Unauthorized();
        // get class
        if (await _context.Classes.FindAsync(classDeckContentDto.classId) ==
null) return NotFound();

        var studyClass = await _context.Classes
            .Include(item => item.User)
            .FirstAsync(item => item.Id == classDeckContentDto.classId);
        if (studyClass == null) return NotFound();
        if (studyClass.User?.UserName != User.Identity?.Name) return
Unauthorized();
        // remove deck from class
        deck.Class = null;
        // save
        var result = await _context.SaveChangesAsync() > 0;
        if (result) return Ok("Class has been removed from deck");

        return BadRequest(new ProblemDetails { Title = "Problem with assigning
class to deck" });
    }

    [Authorize]
    [HttpDelete("{id}")]
    public async Task<ActionResult> DeleteDeck(int id)
    {
        // get user
        var user = await _userManager.FindByNameAsync(User.Identity?.Name);
        if (user == null) return Unauthorized();
        // get deck
        if (await _context.Decks.FindAsync(id) == null) return NotFound();
```

```
        var deck = await _context.Decks
            .Include(item => item.User)
            .FirstAsync(item => item.Id == id);
        if (deck == null) return NotFound();
        if (deck.User?.UserName != User.Identity?.Name) return Unauthorized();
        // remove deck
        _context.Decks.Remove(deck);
        // save
        var result = await _context.SaveChangesAsync() > 0;
        if (result) return Ok("Deck has been removed");

        return BadRequest(new ProblemDetails { Title = "Problem with removing
deck" });
    }
}
```

ClassesController.cs

```
namespace API.Controllers;

public class ClassesController : BaseApiController
{
    private readonly DataContext _context;
    private readonly IMapper _mapper;
    private readonly UserManager<User> _userManager;

    public ClassesController(DataContext context, IMapper mapper,
UserManager<User> userManager)
    {
        _context = context;
        _mapper = mapper;
        _userManager = userManager;
    }

    [HttpGet]
    public async Task<ActionResult<PagedList<ClassDto>>> GetClasses([FromQuery]
QueryParams queryParams)
    {
        var query = _context.Classes
            .Search(queryParams.SearchTerm)
            .Include(item => item.User)
            .Select(items => _mapper.Map<ClassDto>(items))
            .AsQueryable();

        var classes = await PagedList<ClassDto>.ToPagedList(query,
queryParams.PageNumber, queryParams.PageSize);

        Response.AddPaginationHeader(classes.MetaData);

        return classes;
```

```
    }

    [HttpGet("{id}", Name = "GetClass")]
    public async Task<ActionResult<ClassWithDeckDto>> GetClass(int id)
    {
        if (await _context.Classes.FindAsync(id) == null) return NotFound();

        var studyClass = await _context.Classes
            .Include(item => item.User)
            .Include(item => item.Decks)
            .ThenInclude(item => item.Cards)
            .FirstAsync(item => item.Id == id);

        if (studyClass == null) return NotFound();

        if (studyClass.IsPrivate && studyClass.User.UserName !=
User.Identity?.Name) return Unauthorized();

        return _mapper.Map<ClassWithDeckDto>(studyClass);
    }

    [Authorize]
    [HttpGet("current-user")]
    public async Task<ActionResult<List<ClassDto>>> GetClassesForCurrentUser()
    {
        // get user
        var user = await _userManager.FindByNameAsync(User.Identity.Name);
        if (user == null) return Unauthorized();
        // get decks for user
        var studyClasses = await _context.Classes
            .Include(item => item.User)
            .Include(item => item.Decks)
            .Where(item => item.UserId == user.Id)
            .Select(items => _mapper.Map<ClassDto>(items))
            .ToListAsync();

        return studyClasses;
    }

    [Authorize]
    [HttpPost]
    public async Task<ActionResult<ClassDto>> CreateClass(ContentDto classDto)
    {
        // get user
        var user = await _userManager.FindByNameAsync(User.Identity.Name);
        if (user == null) return Unauthorized();
        // create class
        var studyClass = new Class
        {
            Title = classDto.Title,
            Description = classDto.Description,
```

```
            IsPrivate = classDto.IsPrivate,
            CreatedAt = DateTime.Now,
            UpdatedAt = DateTime.Now
        };
        // add class
        user.Classes.Add(studyClass);
        // save
        var result = await _context.SaveChangesAsync() > 0;

        if (result) return CreatedAtRoute("GetClass", new { id = studyClass.Id },
_mapper.Map<ClassDto>(studyClass));

        return BadRequest(new ProblemDetails { Title = "Problem with saving
class" });
    }

    [Authorize]
    [HttpPut("{id}")]
    public async Task<ActionResult<ClassDto>> UpdateClass(int id, ContentDto
classDto)
    {
        // get user
        var user = await _userManager.FindByNameAsync(User.Identity?.Name);
        if (user == null) return Unauthorized();
        // get class
        if (await _context.Classes.FindAsync(id) == null) return NotFound();

        var studyClass = await _context.Classes
            .Include(item => item.User)
            .FirstAsync(item => item.Id == id);
        if (studyClass == null) return NotFound();
        if (studyClass.User?.UserName != User.Identity?.Name) return
Unauthorized();
        // update deck
        studyClass.Title = classDto.Title;
        studyClass.Description = classDto.Description;
        studyClass.IsPrivate = classDto.IsPrivate;
        studyClass.UpdatedAt = DateTime.Now;
        // save
        var result = await _context.SaveChangesAsync() > 0;
        if (result) return _mapper.Map<ClassDto>(studyClass);

        return BadRequest(new ProblemDetails { Title = "Problem with updating
class" });
    }

    [Authorize]
    [HttpDelete("{id}")]
    public async Task<ActionResult> DeleteClass(int id)
    {
        // get user
```

```
        var user = await _userManager.FindByNameAsync(User.Identity?.Name);
        if (user == null) return Unauthorized();
        // get class
        if (await _context.Classes.FindAsync(id) == null) return NotFound();

        var studyClass = await _context.Classes
            .Include(item => item.User)
            .Include(item => item.Decks)
            .FirstAsync(item => item.Id == id);
        if (studyClass == null) return NotFound();
        if (studyClass.User?.UserName != User.Identity?.Name) return
Unauthorized();
        // remove class
        studyClass.Decks.RemoveRange(0, studyClass.Decks.Count);
        _context.Classes.Remove(studyClass);
        // save
        var result = await _context.SaveChangesAsync() > 0;
        if (result) return Ok("Class has been removed");

        return BadRequest(new ProblemDetails { Title = "Problem with removing
class" });
    }
}
```

ProxyController.cs

```
namespace API.Controllers;

public class ProxyController : BaseApiController
{
    private readonly IHttpClientFactory _clientFactory;

    public ProxyController(IHttpClientFactory clientFactory)
    {
        _clientFactory = clientFactory;
    }

    [HttpGet("text-to-speech")]
    public async Task<ActionResult> GetSpeachFromText(string text, string lang)
    {
        var httpClient = _clientFactory.CreateClient();
        var request = new HttpRequestMessage(HttpMethod.Get,
$"https://translate.google.com.vn/translate_tts?ie=UTF-
8&q={text}&tl={lang}&client=tw-ob");

        var response = await httpClient.SendAsync(request);

        if (response.IsSuccessStatusCode)
        {
            var stream = await response.Content.ReadAsStreamAsync();
            return File(stream, "audio/mpeg");
```

```
        }
        else
        {
            return StatusCode((int)response.StatusCode);
        }
    }
}
```

Program.cs

```
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllers();
// Swagger configuration
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen(conf =>
{
    var jwtSecurityScheme = new OpenApiSecurityScheme
    {
        BearerFormat = "JWT",
        Name = "Authorization",
        In = ParameterLocation.Header,
        Type = SecuritySchemeType.ApiKey,
        Scheme = JwtBearerDefaults.AuthenticationScheme,
        Description = "Put Bearer + your token in the box below",
        Reference = new OpenApiReference
        {
            Id = JwtBearerDefaults.AuthenticationScheme,
            Type = ReferenceType.SecurityScheme
        }
    };

    conf.AddSecurityDefinition(jwtSecurityScheme.Reference.Id,
jwtSecurityScheme);

    conf.AddSecurityRequirement(new OpenApiSecurityRequirement
    {
        {
            jwtSecurityScheme, Array.Empty<string>()
        }
    });
});
builder.Services.AddDbContext<DataContext>(opt =>
{

opt.UseSqlite(builder.Configuration.GetConnectionString("DefaultConnection"));
});
builder.Services.AddCors();
builder.Services.AddAutoMapper(typeof(Program));
builder.Services.AddIdentityCore<User>(opt =>
```

```
{
    opt.User.RequireUniqueEmail = true;
    opt.User.AllowedUserNameCharacters =
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789-._@+/ ";
})
    .AddRoles<IdentityRole>()
    .AddEntityFrameworkStores<DataContext>();

builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
    .AddJwtBearer(opt =>
    {
        opt.TokenValidationParameters = new TokenValidationParameters
        {
            ValidateIssuer = false,
            ValidateAudience = false,
            ValidateLifetime = true,
            ValidateIssuerSigningKey = true,
            IssuerSigningKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(builder.Configuration["JWTSettings:To
kenKey"]))
        };
    });
builder.Services.AddAuthorization();
builder.Services.AddScoped<TokenService>();
builder.Services.AddHttpClient();

var app = builder.Build();

// Configure the HTTP request pipeline.
app.UseMiddleware<ExceptionMiddleware>();

if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI(conf =>
    {
        conf.ConfigObject.AdditionalItems.Add("persistAuthorization", "true");
    });
}

app.UseCors(opt =>
{

opt.AllowAnyHeader().AllowAnyMethod().AllowCredentials().WithOrigins("http://loca
lhost:3000");
});
app.UseAuthentication();
app.UseAuthorization();
app.MapControllers();

var scope = app.Services.CreateScope();
```

```
var context = scope.ServiceProvider.GetRequiredService<DataContext>();
var userManager = scope.ServiceProvider.GetRequiredService<UserManager<User>>();
var logger = scope.ServiceProvider.GetRequiredService<ILogger<Program>>();

app.Run();
```

А.2 Клієнтський шар інформаційної системи

Agent.ts

```
axios.defaults.baseURL = "http://localhost:5000/api/";
axios.defaults.withCredentials = true;

const sleep = () => new Promise((resolve) => setTimeout(resolve, 500));
const responseBody = (response: AxiosResponse) => response.data;

axios.interceptors.request.use((config) => {
  const token = store.getState().account.user?.token;
  if (token) config.headers.Authorization = `Bearer ${token}`;
  return config;
});

axios.interceptors.response.use(
  async (response) => {
    await sleep();

    const pagination = response.headers["pagination"];
    if (pagination) {
      response.data = new PaginatedResponse(
        response.data,
        JSON.parse(pagination)
      );
    }

    return response;
  },
  (error: AxiosError) => {
    const { status } = error.response as AxiosResponse;

    switch (status) {
      case 401:
        router.navigate("/unauthorized");
        break;
      case 404:
        router.navigate("/not-found");
        break;
    }

    return Promise.reject(error.response);
  }
);
```

```
const requests = {
  get: (url: string, params?: URLSearchParams) =>
    axios.get(url, { params }).then(responseBody),
  getRawData: (url: string) =>
    axios.get(url, { responseType: "arraybuffer" }).then(responseBody),
  getRawDataWithParams: (url: string, params?: URLSearchParams) =>
    axios
      .get(url, { params, responseType: "arraybuffer" })
      .then(responseBody),
  post: (url: string, body: object) => axios.post(url, body).then(responseBody),
  postWithParams: (url: string, body: object, params: URLSearchParams) =>
    axios.post(url, body, { params }).then(responseBody),
  put: (url: string, body: object) => axios.put(url, body).then(responseBody),
  putWithParams: (url: string, body: object, params: URLSearchParams) =>
    axios.put(url, body, { params }).then(responseBody),
  delete: (url: string) => axios.delete(url).then(responseBody),
};

const Class = {
  list: (params?: URLSearchParams) => requests.get(`classes`, params),
  listForCurrentUser: () => requests.get(`classes/current-user`),
  classById: (id: number) => requests.get(`classes/${id}`),
  createClass: (values: any) => requests.post(`classes`, values),
  updateClass: (id: number, values: any) => requests.put(`classes/${id}`,
values),
  deleteClass: (id: number) => requests.delete(`classes/${id}`),
};

const Deck = {
  list: (params?: URLSearchParams) => requests.get(`decks`, params),
  listForCurrentUser: () => requests.get(`decks/current-user`),
  deckById: (id: number) => requests.get(`decks/${id}`),
  createDeck: (values: any) => requests.post(`decks`, values),
  updateDeck: (id: number, values: any) => requests.put(`decks/${id}`, values),
  deleteDeck: (id: number) => requests.delete(`decks/${id}`),
  assignToClass: (values: any) => requests.put(`decks/assign-to-class`, values),
  removeFromClass: (values: any) => requests.put(`decks/remove-from-class`,
values),
};

const Card = {
  createCard: (id: number, values: any, params: URLSearchParams) =>
    requests.postWithParams(`cards/${id}`, values, params),
  getImage: (fileName: string) =>
    requests.getRawData(`cards/image/${fileName}`),
  updateCard: (id: number, values: any, params: URLSearchParams) =>
    requests.putWithParams(`cards/${id}`, values, params),
  deleteCard: (id: number) => requests.delete(`cards/${id}`),
};
```

```
const User = {
  login: (values: any) => requests.post(`user/login`, values),
  register: (values: any) => requests.post(`user/register`, values),
  currentUser: () => requests.get(`user/currentUser`),
};

const Proxy = {
  textToSpeech: (params?: URLSearchParams) =>
    requests.getRawDataWithParams(`proxy/text-to-speech`, params),
};

const agent = {
  Class,
  Deck,
  Card,
  User,
  Proxy,
};

export default agent;
```

App.tsx

```
export default function App() {
  const dispatch = useAppDispatch();
  const [loading, setLoading] = useState(true);

  const initApp = useCallback(async () => {
    try {
      await dispatch(fetchCurrentUser());
    } catch (error: any) {
      console.log(error);
    }
  }, [dispatch]);

  useEffect(() => {
    initApp().then(() => setLoading(false));
  }, [initApp]);

  const theme = createTheme({
    typography: {
      fontFamily: "poppins, sans-serif",
    },
  });

  if (loading) return <LoadingComponent />;

  return (
    <ThemeProvider theme={theme}>
      <CssBaseline />
      <Header />
```

```
      <Box minHeight="100vh">
        <Outlet />
      </Box>
      <Footer />
    </ThemeProvider>
  );
}
```

Routes.tsx

```
export const router = createBrowserRouter([
  {
    path: "/",
    element: <App />,
    children: [
      {
        element: <RequireAuth />,
        children: [
          { path: "dashboard", element: <Dashboard /> },
          { path: "profile", element: <Profile /> },
          { path: "edit/deck/:id", element: <EditCardsPage /> },
        ],
      },
      { path: "", element: <HomePage /> },
      { path: "search", element: <Search /> },
      { path: "deck/:id", element: <DeckDetails /> },
      { path: "class/:id", element: <ClassDetails /> },

      { path: "login", element: <Login /> },
      { path: "register", element: <Register /> },
      { path: "logout", element: <Logout /> },

      { path: "not-found", element: <NotFound /> },
      { path: "unauthorized", element: <Unauthorized /> },
      { path: "*", element: <Navigate replace to="/not-found" /> },
    ],
  },
]);
```

configureStore.ts

```
export const store = configureStore({
  reducer: {
    account: accountSlice.reducer,
  },
});
export type RootState = ReturnType<typeof store.getState>;
export type AppDispatch = typeof store.dispatch;
export const useAppDispatch = () => useDispatch<AppDispatch>();
export const useAppSelector: TypedUseSelectorHook<RootState> = useSelector;
```

main.tsx

```
ReactDOM.createRoot(document.getElementById("root")!).render(
  <Provider store={store}>
    <RouterProvider router={router} />
  </Provider>
);
```