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)

22 червня 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавр

зі спеціальності 122 – Комп'ютерних наук,

освітньо-професійної програми «Інформатика»

на тему: «Веб-орієнтований навчальний комплекс «Тренажер-словник з англійської мови»»

здобувача групи ІН-94-1 Баришова Максима Володимировича.

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

Максим БАРИШОВ

(підпис)

Керівник

кандидат фізико-математичних наук,

асистент кафедри комп'ютерних наук Олександр ВЛАСЕНКО

Суми – 2023

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавра

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»
здобувача групи ІН-94-1 Баришова Максима Володимировича

1. Тема роботи: «Веб-орієнтований навчальний комплекс «Тренажер-словник з англійської мови»» затверджую наказом по СумДУ від *«01» червня 2023 р. № 0475-VI*
2. Термін здачі здобувачем кваліфікаційної роботи *до 09 червня 2023 року*
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
1) Аналіз проблеми предметної області, постановка й формування завдань дослідження. 2) Огляд технологій, що використовуються для побудови веб-системи таких типів. 3) Розроблення системи словника та тренажер англійської мови. 4) Аналіз результатів.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
6. Консультанти до проєкту (роботи), із значенням розділів проєкту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «____» _____ 2023 р.

Завдання прийняв до виконання _____ (підпис) Керівник _____ (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 розробки тренажерів*		
3	*Розробка веб-орієнтованого навчального комплексу для вивчення англійської мови»»*		
4	*Аналіз отриманих результатів*		
5	*Оформлення пояснювальної записки до кваліфікаційної роботи*		

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

АНОТАЦІЯ

Записка: 53 стор., 25 рис, 11 використаних джерел.

Обґрунтування актуальності теми роботи – англійська мова є необхідним інструментом спілкування, вона широко використовується по всьому світу, знання її відкриває широкі можливості не тільки для здобувачів освіти, а і для звичайних людей. Гарних і якісних інструментів для вивчення англійської мови не достатньо.

Об'єкт дослідження – процес вивчення англійської мови, зокрема збільшення словникового запасу та його відтворення. Процес відновлення в пам'яті вже вивчених слів. Організація структурованого доступного словника слів.

Мета роботи – розробка веб-додатка, який наддасть можливість створювати колекції слів, знаходити слова, знаходити переклад слів додавати їх до колекцій. Тренувати додані слова за допомогою тренажера слів та відслідковувати прогрес навчання. Можливість систематично відновлювати опрацьовані слова.

Методи дослідження – аналіз потреб користувачів, дослідження ринку та конкурентів, проведення фокус-груп, а також збір та аналіз даних веб-аналітики.

Результати – виконано аналіз предметної області та аналогічних сайтів, сформульовано цілі і вирішено проблеми. Розроблено інформаційну систему з функціональним інтерфейсом, яка забезпечує пошук та додання нових слів до колекцій слів а також тренування та відслідковування прогресу вивчення слів. Система стабільно працює на локальному сервері, розроблені веб-ресурси вже доступні для пробного використання.

ІНФОРМАЦІЙНА СИСТЕМА, СЛОВНИК, ТРЕНАЖЕР, АНГЛІЙСЬКА МОВА, ANGULAR, EXPRESS, NODE JS.

ЗМІСТ

ВСТУП

У сучасному світі англійська мова відіграє важливу роль як міжнародний засіб спілкування, особливо в контексті глобалізації та міжнародної співпраці. Вивчення англійської мови стає все більш актуальним та необхідним завданням для багатьох людей, зокрема для студентів, які прагнуть розширити свої можливості у навчанні, роботі та особистому розвитку. Без сумніву, завдяки наявності інтернету та різноманітних гаджетів процес вивчення мови кардинально змінився. Використання новітніх і працюючих практик є в пріоритеті для переважної більшості тих хто вивчає мову. І на щастя, в наш час є безліч сервісів які впроваджують дані практики. Але тоді постає питання чому до цих пір знаходяться люди які навчаються за цими програмами, курсами, онлайн школами, але не мають особливих досягнень ? Можливо, все залежить від того хто вивчає. Але є нюанс, що різні люди шукають наживу на цьому і не переслідують наміри навчити людей. Можна заплатити велику суму, але у підсумку отримати «нічого». Вивчення мови є складною і комплексною працею, тому вибір інструментів для цього є справді дуже відповідальним етапом який вплине на успіх навчання.

Метою кваліфікаційної роботи було створення універсального інструменту який гарантуватиме результат і вважатиметься практичним веб-сервісом "Тренажер-словник з англійської мови", який надаватиме зручний та ефективний стек можливостей для освоєння англійської мови. Основною метою цього сервісу є надання можливості користувачам вільно вправлятися в перекладі та запам’ятовуванню слів, висловів англійською мовою, розширюючи словарний запас та покращуючи навички.

Щоб реалізувати таку веб-систему необхідно провести кропітку роботу по проєктуванню і створенню дизайну щоб подібна система могла конкурувати з іншими подібними проєктами.

1 ІНФОРМАЦІЙНІЙ ОГЛЯД

1.1 Огляд проблемної області

Комунікація і здатність налагоджувати контакт з один одним є основою людських стосунків та співпраці. Однак, необхідність володіти іноземною мовою, зокрема англійською, часто перешкоджає ефективній комунікації між людьми з різних країн. Багато людей відчувають необхідність вивчати англійську мову, оскільки вона є однією з найпопулярніших (рис. 1.1) і найбажаніших мов у світі.

Знання англійської мови необхідне не тільки для забезпечення ефективного спілкування, але й для розуміння іноземного контенту різних видів, починаючи з розважальних відео і фільмів, і закінчуючи науковими джерелами, які можуть бути важливими для навчання або професійної діяльності.

Навіть у вузькому колі людей, необхідність володіння іноземною мовою може стати життєвою необхідністю для знаходження роботи або підвищення кваліфікації за кордоном. Крім того, неможливо передбачити, коли саме знання англійської мови може стати важливим чинником у житті кожної людини.

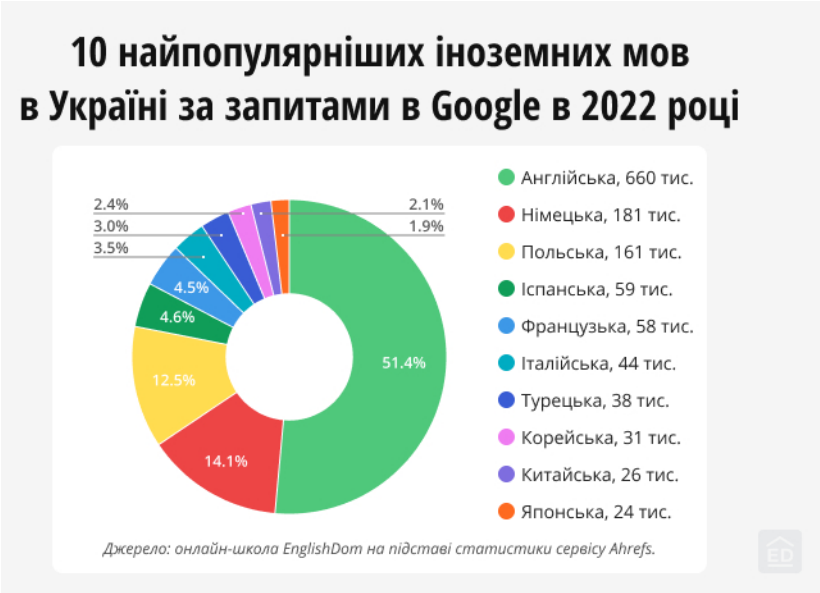

Рисунок 1.1 — Статистика найпопулярніших мов в Україні

Тому англійська мова вважається найбільш доцільною для вивчення як першої іноземної мови. Її популярність у всьому світі відкриває багато можливостей для міжнародного спілкування, навчання, кар'єрного росту та культурному обміні. На часі зараз і те, що Україна як ніколи до цього, стала ближчою до Європи і вектор розвитку показує що в майбутньому, можливо саме англійська мова буде другою державною, а поки маємо таку динаміку запитів до Google (рис. 1.2).

Усі ці фактори підкреслюють значення вивчення англійської мови та визначають проблемну область, пов'язану з недостатнім рівнем знання цієї мови у багатьох людей. Вивчення англійської мови вимагає зусиль та ефективних інструментів, які допомагатимуть досягти поставлених цілей.

Рисунок 1.2 — Динаміка популярності мов в Україні

Незважаючи на широкий вибір ресурсів, таких як онлайн школи, книги та курси, деякі люди можуть зіткнутися з труднощами у вивченні мови. Це може бути пов'язано з вибором несвідомих сервісів або відсутністю підтримки та корекції з боку вчителя. Для досягнення успіху вивчення англійської мови важливо знайти підходящий сервіс або вчителя, який зможе надати необхідну підтримку та навчити ефективним стратегіям навчання. А краще коли і одне і друге працюють разом, що гарантуватиме ще більший результат.

Також існує освітня проблема, коли людям, які закінчують школу або університет, часто не вистачає знань, щоб успішно скласти іспити з англійської мови. Міністерство освіти також визнає цю проблему та підтверджує необхідність додаткових курсів та програм для забезпечення успішного вивчення мови та складанню іспитів. Важливість наявності додаткового навчання може становити значну частку (до 90%) у випадку успішного опанування англійської мови.

У зв'язку з цим, існує потреба у додаткових інструментах та сервісах для ефективного вивчення мови. Вибір неправильного сервісу може призвести до невдачі у вивченні англійської мови. Вирішення цих проблем вимагає

розробки ефективних методів і підходів до навчання англійської мови та розширення доступу до якісних навчальних інструментів.

1.2 Огляд Web-ресурсів

У сучасному світі існує безліч різноманітних сервісів, веб-ресурсів та інструментів, які пропонують можливості для вивчення англійської мови. Проте, при пошуку ідеального веб-ресурсу, користувачі часто зіштовхуються з великою кількістю комерційних, безкоштовних та інших пропозицій. Як зробити правильний вибір серед усіх цих варіантів і знайти веб-ресурс, який задовольнить потреби кожного користувача?

Один з підходів до вирішення цієї проблеми полягає у знаходженні спільних рис та характеристик, які об'єднують ці веб-ресурси. Нижче наведений огляд найпоширеніших типів веб-ресурсів, які вдалося розглянути.

Інтерактивні веб-сайти та платформи: Було виявлено широкий вибір онлайн-сайтів та платформ які пропонують інтерактивні курси англійської мови. Ці ресурси надають відео навчальні матеріали, аудіозаписи, вправи на граматику та лексику, а також інтерактивні завдання і тести. Деякі платформи навіть пропонують можливість спілкування з носіями мови або викладачами, що дозволяє отримати практичний досвід у реальних ситуаціях, прикладом є AntiSCHOOL (рис. 1.3). За часту саме отримання такого досвіду є найкориснішим способом удосконалити себе. Та і взагалі, будь-яка сфера діяльності для опанування потребує практики в першу чергу, а правило вісімдесят на двадцять говорить про те, що 80 процентів практики і 20 процентів теорії дає максимум результату.

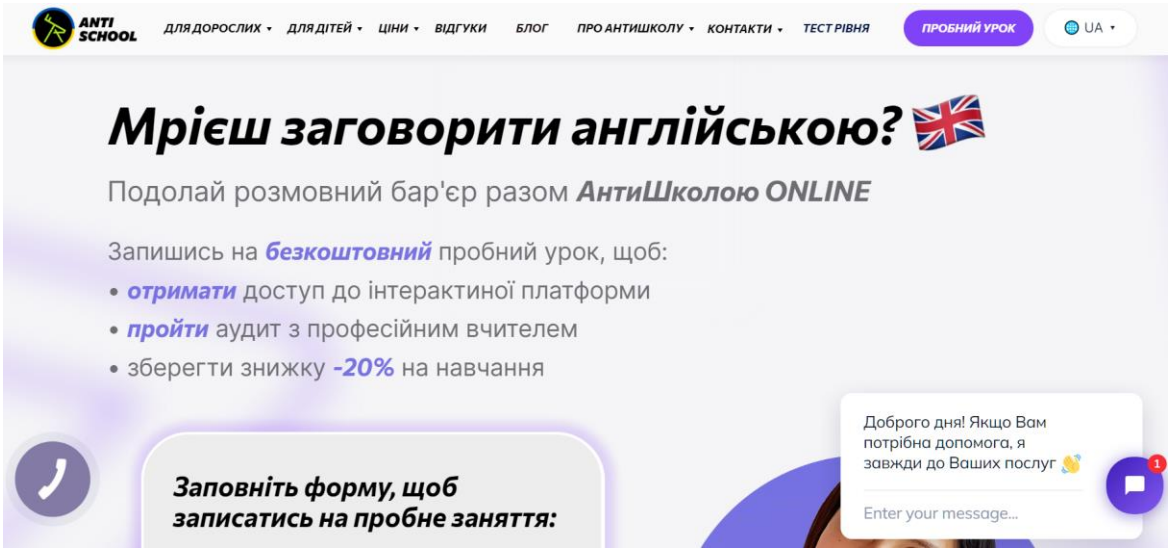

Рисунок 1.3 — Сторінка сайту antischool.online

Онлайн-спільноти та форуми: В інтернеті існує також багато онлайн-спільнот та форумів, присвячених вивченню англійської мови. У цих спільнотах студенти, школярі і всі здобувачі подібних знань можуть обговорювати теми, задавати питання, обмінюватися корисною інформацією та навіть знаходити партнерів для спілкування та виконання вправ. Не слід забувати і про існування соціальних мереж. Для сучасних людей вже стало нормою дізнаватися інформацію з різних медіа ресурсів. Новини з телебачення уже давно стали не актуальними так як, якщо людина має доступ в інтернет і підписана на канал який її цікавить, новина доходить набагато швидше до розуму і через мобільні або комп’ютерні девайси за допомогою інтернету зараз можна добувати інформацію в рази швидше. І тематика англійської мови також не обходить стороною дану сферу, існує безліч Facebook, Viber, Instagram (рис. 1.4) сторінок в соціальних мережах які так чи інакше пов’язані з темами вивчення англійської мови.

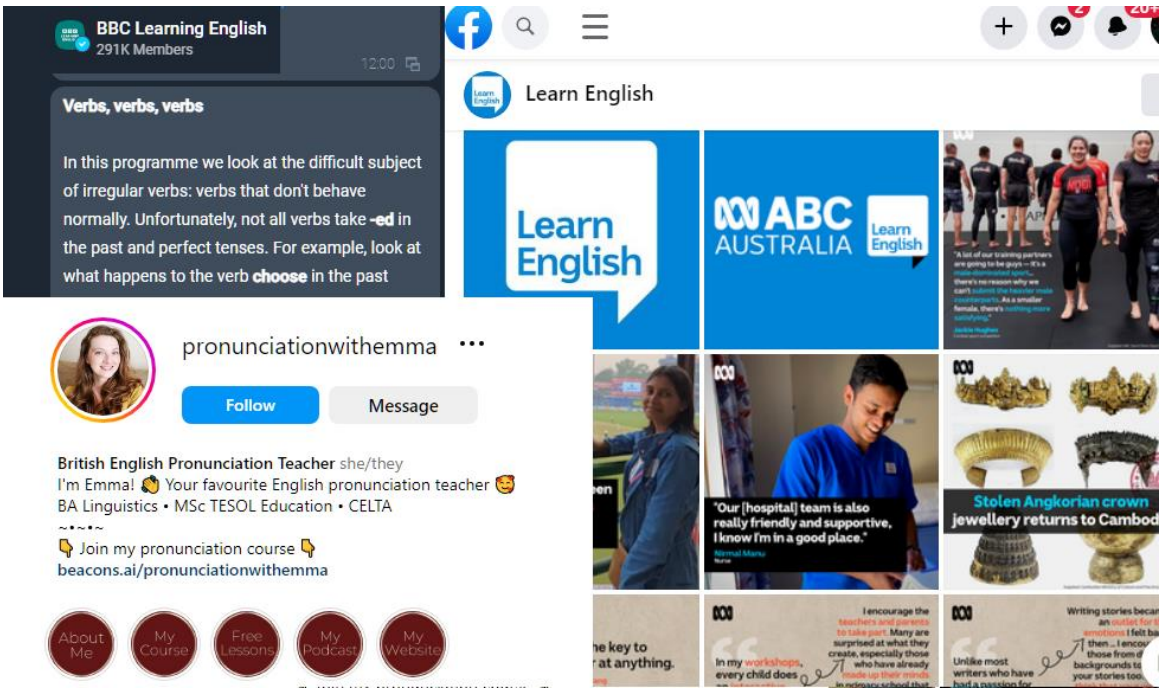

Рисунок 1.4 — Освітні соціальні сторінки з англійської мови

Навчальні відео матеріали та канали на YouTube: YouTube є джерелом безлічі відеоуроків англійської мови, які можуть бути корисним додатковим інструментом для самостійного навчання. На каналах з вивчення мови можна знайти відео з граматики, вимови, лексики, а також автентичні матеріали, такі як відео з реальними розмовами або фільми з субтитрами. Не дарма кажуть, щоб вивчити мову необхідно оточити себе мовою з усіх сторін. На подібних відео-сервісах можна знайти неймовірну багату кількість уроків для вашого рівня знання чи не знання англійської мови. Саме чудове, що окрім відео-уроків останнім часом стали популярними TikTok, Shorts, Reels все це однотипні короткі відео які дуже легко дивитись і особливість є в тому, що вони працюють за алгоритмами які прораховують інтереси користувача, формуючи стрічку з подібних цікавих відео. Це досить гарна практика, адже занурюючись в контекст того, чого ти вивчаєш стає простіше і швидше здобувати нові знання та навички.

Мобільні додатки: Розвиток технологій сприяє появі різноманітних мобільних додатків для вивчення англійської мови. Ці додатки пропонують вправи, ігри, відслідковують прогрес користувача та надають можливість навчатися в будь-який зручний час і в будь-якому місці. Прикладом такого

додатку є Duolingo (рис. 1.5). І дійсно, використання такого потужного інструменту повинно давати результат, адже коли є ігровий контекст завжди зацікавленість людини підвищується. Особливо такий підхід буде пасувати дітям, яким дуже важко терпіти заняття та бути уважними на них. Але і дорослі люди в середині залишаються дітьми, тому на них це також працюватиме. Мобільні телефони завжди поруч і часто виникають моменти в житті, де потрібно чекати щось або якось скоротити час тому телефон з встановленим на нього освітнім додатком може стати в пригоді і таким чином посприяти опануванню мови.

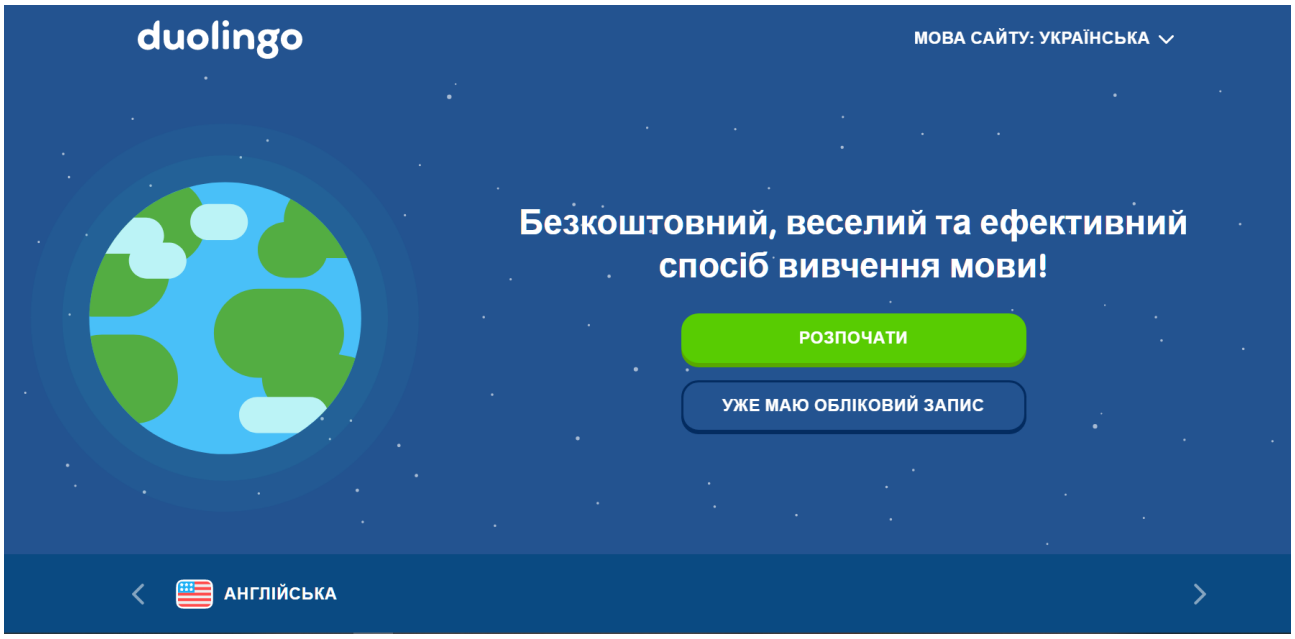

Рисунок 1.5 — Сторінка сайту duolingo.com

Онлайн-курси та вебінари: Деякі відомі освітні установи та мовні школи надають онлайн-курси та вебінари з англійської мови. Ці ресурси можуть бути оптимальним вибором для учнів, які бажають отримати більш структуроване навчання під керівництвом викладача та зможуть отримати додаткову підтримку та зворотний зв'язок. Таке навчання є дуже ефективним адже за нього беруться інші люди, які беруть відповідальність за те, що ви повинні мати результат по закінченню цих курсів. Також вони мають досвід у підготовці подібних учнів, тому якщо і вибирати подібний підхід спочатку

необхідно упевнитись у доброчесності тієї чи іншої компанії і перевірити відгуки.

1.3 Актуальність розробки

Вивчення англійської мови має велике значення в сучасному світі, де комунікація та знання іноземної мови стали невід'ємною частиною в різних сферах життя. Англійська мова є однією з найпопулярніших та найважливіших мов, яка використовується в бізнесі, туризмі, міжнародних відносинах, наукових дослідженнях, освіті та багатьох інших сферах.

Звичайно, не можна забувати, що Україна веде активну політику європейського інтегрування та визнає важливість знання англійської мови для розвитку країни. Прийняття сучасного європейського вектору розвитку має безпосередній вплив на потреби українців у вивченні англійської мови.

Наприклад, військовим особам, які знають англійську мову, відкриваються нові можливості для навчання та співпраці з іншими країнами. Вони можуть брати участь у міжнародних навчальних програмах, тренуваннях та спільних військових відрядженнях, сприяючи підвищенню рівня військової підготовки переймаючи досвід і співпрацювати з іншими державами. Це допомагає покращувати обороноспроможність та відстоювати інтереси країни і на даний момент захищатись від окупанта. Крім того, знання англійської мови дозволяє українським громадянам активніше впливати на політичні та економічні процеси, розвивати міжнародні зв'язки та залучати іноземні інвестиції. Володіння англійською мовою відкриває двері до міжнародних ринків праці та можливостей для кар'єрного зростання.

Таким чином, вивчення англійської мови стає необхідним для українців не лише з освітньою та комунікативною метою, але й для відповіді на виклики, що ставляться перед нашою державою у контексті сучасного світу.

Однак, багато людей зіштовхуються з труднощами та випробуваннями у процесі вивчення англійської мови. Це може бути пов'язано з різними факторами, такими як недостатня мотивація, неефективні методи навчання,

відсутність доступу до якісних навчальних ресурсів та відсутність підтримки вчителя або наставника.

Саме тому актуальність розробки додаткових інструментів та сервісів для ефективного вивчення англійської мови стає все більш важливою. Люди потребують доступу до якісних та інноваційних ресурсів, які можуть підтримати їх у навчанні та допомогти досягти поставлених цілей.

Також слід зазначити, що сучасні вимоги до володіння англійською мовою зростають. У багатьох сферах професійної діяльності, вакансіях та кар'єрному розвитку часто вимагається високий рівень володіння англійською мовою. Існує попит на ефективні методи та інструменти навчання, які допоможуть студентам підвищити свої навички та конкурентоспроможність на ринку праці.

Отже, актуальність розробки веб-ресурсів та інструментів для вивчення англійської мови полягає у задоволенні потреб сучасних учнів, студентів, громадян, користувачів які шукають ефективні та зручні способи навчання. Ці розробки можуть надати доступ до високоякісних інструментів, навчальних матеріалів та сприяти самостійному навчанню, підвищити мотивацію та досягнення вивчення англійської мови, а також допомогти у підготовці до іспитів та особистому і професійному розвитку.

1.4 Постановка задачі

Задача проєкту полягає у створенні ефективного прототипу веб-сайту – “Тренажер-словник з англійської мови”, який допоможе користувачам поліпшити свої навички вивчення англійської мови. Для досягнення цієї мети, проєкт має наступні вимоги:

Рисунок 1.6 — Вимоги до проєкту

Також у проєкті повинні бути реалізовані наступні компоненти:

Лендінг: Це перша сторінка, яку зустрічає користувач, і вона має зацікавити та залучити його до використання словника-тренажера. Лендінг може містити коротку інформацію про переваги використання словник-тренажера, його функціонал та можливості.

Словник-тренажер: Це основний функціонал проєкту, де користувачі матимуть можливість знаходити слова, додавати їх до колекцій, виконувати завдання на запам’ятовування цих слів і потім на їх засвоєння та повторення. Можливості ділитися колекціями з іншими користувачам, викладати їх у відкритий доступ. Він може включати такі можливості, як пошук по колекціям, пошук колекцій за тематикою, додавання нових колекцій до своїх з можливістю тренувати їх, та багато чого іншого.

Вправи та тести: Словник-тренажер також має містити вправи та тести, які допоможуть користувачам вивчати додані до колекцій слова і закріплювати їх. Це можуть бути різні типи завдань, включаючи вибір правильного значення, заповнення пропусків, переклад та інші.

Розширений функціонал: Додатковий функціонал може включати можливість перегляду статистики вивчених слів, можливо ігровий режим з

отриманням призів за кожне виконане завдання, відстеження прогресу, можливість роздрукувати колекцію слів. Система нагадувань, щоб користувач міг розуміти коли йому слід займатись і що саме потрібно робити.

Адміністрування: Для зручного керування словником-тренажером може бути створена адміністративна панель, де адміністратор зможе додавати та видаляти слова, керувати користувачами та виконувати інші адміністративні функції.

Цей веб-інструмент “Словник-тренажер з англійської мови” має на меті надати користувачам зручні та ефективні інструменти для вивчення та покращення їх навичок англійської мови.

2 ВИБІР МЕТОДУ РІШЕННЯ

2.1 Вибір програмної реалізації

З кожним роком програмне забезпечення для розробки веб-додатків стрімко змінюється. Буває навіть таке, що за трендами розробки дуже важко встигати, адже на освоєння того чи іншого нового матеріалу потрібен час і на практику також. Тому щоб розробити сучасну веб-систему треба розуміти структуру сучасних веб-додатків.

Розвиток веб-додатків розпочався у 1990-х роках. З тих пір відбулося значне зростання та розвиток веб-технологій. Спочатку веб складався з простих HTML-документів та сторінок. Кожний початківець програміст знає про існування мови гіпертекстової розмітки - HTML. Вона і до тепер використовується в програмуванні, завдяки її існуванню можна створювати каркас веб-сторінок, і використовуючи html-теги формувати DOM(Document object model)-дерево. Але повернемося в ті часи, і спочатку HTML-документи використовувались вченими та іншими зацікавленими особами для обміну інформацією. Все це відбувалося за допомогою протоколу HTTP. Також варто зазначити, що сама HTML-сторінка не "літає у повітрі". Після того, як користувач вводить адресу додатка або сайту в браузері, відбувається HTTP-запит до сервера. Браузер отримує відповідь, а вже потім відображає готову сторінку, з якою можна взаємодіяти.

Кожна доступна веб-сторінка має свою адресу, а зв'язок між цими сторінками досягається за допомогою гіперпосилань, що дозволяють переходити з однієї сторінки на іншу. Початково всесвітня павутина була про обмін документами та сторінками, які пов'язані між собою гіперпосиланнями. Ця ідея та концепція досить проста рис (2.1).

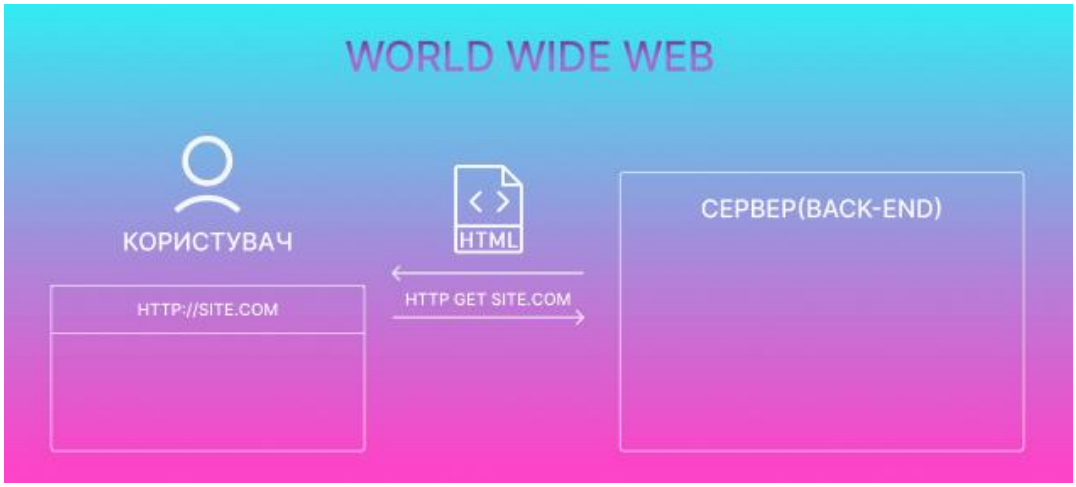

Рисунок 2.1 — Концепція World Wide Web

Очевидно, що з часом технології стали складнішими, і простого HTML вже не вистачало. З'явилася потреба додавати візуальну складову на сторінку, змінювати кольори та шрифти, задавати розміри та позиціонування. Тоді з'явилися каскадні таблиці стилів CSS, за допомогою яких можна було змінювати вигляд HTML-сторінок.

Проте зовнішній вигляд сторінки - це лише один аспект. Очевидно, що з часом з потужним інструментом обміну даними для розробки знадобиться не тільки відображення інформації, але й її збір. Було бажання отримувати різні дані від користувачів, які знаходяться на відстані тисяч кілометрів від нас. Але вже звичного надсилання даних недостатньо - цю інформацію потрібно десь зберігати. Тут розвиток наблизився до серверних технологій, до роботи з базами даних.

Для того, щоб віддавати статичні сторінки без динамічних даних, достатньо мати простий веб-сервер, який може віддавати статичні файли. Він слухає запити і ідентифікує файли, які потрібно віддати, такі як HTML-файли, файл головної сторінки “index.html”, файл "about_us.html" та інші. Сервер може зберігати дані в базу даних і читати їх з неї. На той момент для роботи з документами ми використовувались HTML-файли та CSS-файли для опису стилів. Тут відбувається певне розподілення - відокремлення серверної і клієнтську частини, яку користувач бачить у браузері. Крім HTML та CSS, ще з'являється JavaScript.

JavaScript дозволяє додавати динаміку на сторінку, створювати сценарії, відкривати модальні вікна, робити складні анімації, обробляти форми, проводити їх валідацію та багато іншого. Якщо на деяких простих сторінках можна було обійтися практично без JavaScript, то з ускладненням цих сторінок вже неможливо обійтися без нього, оскільки саме він відповідає за логіку та динаміку на сторінці. Крім того, JavaScript дозволяє додати таку важливу концепцію, як асинхронні запити.

Отже, як виглядала концепція без асинхронних запитів. Користувач надсилає запит на сервер, наприклад, дані з форми. У цей момент він перебуває у режимі очікування, оскільки дані по мережі передаються не миттєво, і як уже було визначено раніше, сервер генерує користувачу нову HTML сторінку цілком нову і повертає її повністю. Після цього користувач повинен повністю завантажити цю сторінку по мережі. Такий варіант не дуже хороший, оскільки всі сторінки у веб-додатку, як правило, містять деякі загальні частини, навігаційну панель, меню, футер, блок де розміщені контакти. І саме тут з’являється нова концепція асинхронних запитів. За допомогою JavaScript можна надіслати асинхронний запит, і в цей момент браузер чекатиме відповіді від сервера, але сторінка при цьому не оновлюватиметься. Користувач просто перебуватиме у режимі очікування, але оновлення сторінки не відбуватиметься. Це дуже важливий момент, і сервер вже повертає на клієнт не новий згенерований HTML-документ. Він повертає або JSON, або XML, або будь-який інший тип даних. Важливо зрозуміти, що він не повертатиме готову HTML-сторінку. Він повертає на клієнт лише дані, які користувач у нього запитав. JSON є одним з найпоширеніших типів даних для взаємодії клієнта та сервера, і для того, щоб користувач побачив нові дані на своїй сторінці, браузеру не потрібно повністю завантажувати HTML-сторінку. Клієнт може взяти дані з сервера, які він нам надіслав, і вставити їх у потрібні місця на сторінці. І це все можна реалізовувати виключно за допомогою JavaScript, починаючи з запиту і закінчуючи підстановкою цих даних у потрібні місця. І якщо раніше у додатку було багато HTML-сторінок, то зараз

вже стає дуже-дуже багато JavaScript, де розписана логіка взаємодії з сервером та обробка даних які приходять знаходиться саме у JavaScript.

Також до моменту появи асинхронних запитів веб-додаток не містив розділення на front-end та back-end це було приблизно одне й те саме. Але після асинхронних запитів і обробки всіх цих запитів у JavaScript, у додатку відбувається явне розрізнення між клієнтом, тобто фронтендом, і сервером, тобто бекендом. Тепер це два окремі застосунки (рис 2.2), і розробкою займаються дві окремі команди - бекендери та фронтендери. З використанням асинхронних запитів стає зрозумілим те, що у великій кількості HTML немає сенсу, оскільки весь вміст на сторінці можна замінити за допомогою JavaScript та даних, що надсилаються клієнту сервером, наприклад, у форматі JSON. Подібні додатки, які складаються лише з одної HTML-сторінки, називаються SPA (Single Page Application) - одно сторінкові додатки.

Рисунок 2.2 — Розділення веб-застосунків

Але тоді виникає питання: звідки з’являється початкова HTML-сторінка, як користувач отримує її, якщо фронтенд і бекенд тепер це дві окремі додатки. Бекенд - це додаток, який взаємодіє з базою даних і повертає певні дані, а клієнт - це те, з чим ми взаємодіємо у браузері. Істина є в тому, що користувач вводить URL у браузер. У цей момент клієнтський HTTP-запит відправляється на сервер і цей сервер вміючи надавати статичні файли, надсилає HTML, CSS,

JavaScript для користувача у браузер. В результаті отримуємо, що фронтенд - це повноцінний окремий додаток, а за розподіл статики відповідає веб-сервер, наприклад, Nginx. Натомість реальний бекендовий додаток-сервер, в якому знаходиться ядро бізнес-логіки та взаємодія з базою даних, є окремим додатком, через такий підхід, бекенд можуть використовувати багато клієнтів. Це може бути мобільний додаток, веб-додаток, комп'ютерна програма або просто партнери, які взаємодіють з серверами через API. Тобто кількість споживачів може бути будь-якою. І користувачі працюють з цим сервером у форматі запит-відповідь, тобто клієнт вказує йому певну команду, наприклад, дайте мені дані про користувача. І сервер повертає ці дані у форматі JSON, XML або будь-якому іншому форматі. На даному етапі розвитку системи, фронтенд і бекенд є складними окремими додатками. На фронтенді, як вже було з'ясовано, є багато CSS, JavaScript файлів, можуть бути й інші файли різних розширень, такі як PNG, GIF, шрифти і так далі. Ручне керування всім цим стає незручним, тому на допомогу приходять збирачі, такі як Webpack, Rollup, Parcel і так далі. Перед тим, як сказати Nginx, які файли потрібно надавати, програма пропускає код-застосунку через цей збирач.

Збирач готує все це в зручному форматі для використання у продакшені. Він стискає файли. За допомогою Babel, він може підготувати код додатку для роботи у старих браузерах і виконує багато інших вторинних завдань, пропускаючи JavaScript-файли та інші файли через збирач. На виході отримуємо так званий "бандл" (пакет). Це весь код додатка, і цей бандл вже розповсюджується веб-сервером. Взаємодія з бекендом відбувається у форматі запит-відповідь безпосередньо на фронтенді (рис. 2.3).

Рисунок 2.3 – архітектура веб-додатку

Також інфраструктура бекенду має деякі зміни. Програма має ядро, де описано додаток. Там знаходиться бізнес-логіка. Є база даних, з якою це ядро взаємодіє. Зазвичай це реляційна база даних, наприклад, PostgreSQL, яка складається з набору пов'язаних таблиць, де дані строго структуровані.

Для певного типу даних це підходить, але у деякий момент розробник може захотіти зберігати хаотичні, слабо структуровані дані. Одним з видів NoSQL баз даних є документ орієнтовані бази даних, і найбільш поширеною з них є MongoDB.

Розробка проєкту "Словник-тренажер з англійської" передбачає використання сучасних технологій веб-програмування. На основі проведеного аналізу еволюції веб-додатків було прийнято рішення щодо вибору певних інструментів для реалізації клієнтської та серверної частин проєкту.

У розробці клієнтської частини обрано фреймворк Angular. Цей вибір зумовлений його чіткою структурою, яка сприяє організації проєкту та зручному управлінню компонентами. Angular також надає широкі можливості для підключення різних бібліотек та збирачів, що спрощує розробку і поліпшує продуктивність. Окрім того, наявність Angular CLI дозволяє легко створювати компоненти, встановлювати додаткові бібліотеки, а також запускати та збирати проєкт.

Для серверної частини обрано Node.js, який давно зарекомендував себе як чудова технологія для створення серверної складової додатку. Часто він використовується разом з легковажним фреймворком Express, який спрощує процес створення сервера і дозволяє налагодити гарну архітектуру додатку.

Окремою складовою проєкту є база даних, і для цілей створюваного проєкту обрано MongoDB. Вона є популярною документ орієнтованою базою даних, яка забезпечує гнучкість у роботі з даними, їх збереженням та масштабованістю. MongoDB дозволяє зберігати дані у форматі документів, що підходить для проєкту, де можлива зміна структури даних. Крім того, MongoDB має зручний інтерфейс та швидкодію, що позитивно вплине на продуктивність веб-системи.

Підсумовуючи все вище сказане, був проведений аналіз розвитку веб-додатків і на етапі, що відповідає вимогам сучасності підібрано стек технологій для успішної реалізації проєкту "Словник-тренажер з англійської мови".

2.2 Визначення середовища розробки

Інтегроване середовище розробки (IDE) є важливим інструментом для програмістів, оскільки воно надає зручне і ефективне середовище для написання, відладки і тестування програмного коду. Використання IDE допомагає збільшити продуктивність розробника, забезпечуючи різноманітні функціональні можливості та інтеграцію з іншими інструментами.

Одним з популярних IDE, яке використовують програмісти, є Visual Studio Code (VS Code). VS Code є безкоштовним, відкритим і легким у використанні ІДЕ, розробленим компанією Microsoft. Це має декілька переваг, які роблять його чудовим вибором для реалізації проєкту "Словник-тренажер з англійської".

Одна з переваг VS Code полягає в його розширюваності. VS Code надає широкий спектр розширень (рис. 2.4), які дозволяють розширити його функціональність для відповідності конкретним потребам розробника.

Сучасні розробники ПО, також орієнтуються на популярність цієї ІДЄ і розробляють під неї програмне забезпечення. Це дозволяє використовувати різні мови програмування, фреймворки та інші інструменти без необхідності переходити до іншого середовища.

На додаток, VS Code має неперевантажений, приємний інтерфейс користувача, що робить його зручним у використанні. Він має чистий та ергономічний дизайн, який сприяє зручному написанню коду, навігації по проєкту та роботі з різними файлами і папками.

Рисунок 2.4 – Розширення в IDE VS Code

Крім того, VS Code підтримує інтеграцію з системами керування версіями, такими як Git, що є не аби якою знахідкою, адже в будь-який проміжок часу, поки триває війна дані перебувають під загрозою і якщо щось станеться, є можливість виконувати роботу з віддаленим, незалежним сховищем коду та здійснювати контроль версій стане в нагоді.

Під час виконання кваліфікаційної роботи використовувались різні допоміжні програми та сервіси для ефективної розробки проєкту. Одним з основних інструментів був Postman, який дозволив тестувати запити API. Використовуючи Postman, можно створювати, відправляти та аналізувати

HTTP запити, перевіряти їх відповіді та відлагоджувати код, пов'язаний з серверною частиною проєкту.

Для роботи з базою даних MongoDB, користуючись веб-інтерфейсом MongoDB, надається зручний спосіб керування даними. З його допомогою можна легко переглядати, редагувати та виконувати операції з базою даних.

Для взаємодії з базою даних MongoDB з консолі, використовувався інструмент Mongosh. Він дозволяв виконувати запити, відлагоджувати код та керувати базою даних зручним способом, працюючи безпосередньо з командного рядка.

Окрім цього, впровадження інструменту Chat GPT є сучасним засобом для автоматизації та пришвидшення розробки. Завдяки його можливостям штучного інтелекту, можна отримати швидкі результати та автоматизувати деякі завдання.

Також, використання інтегрованого середовища розробки, яке доступне в браузері. Воно надаає зручний інтерфейс для редагування коду, відлагодження, а також дозволяє переглядати в реальному часі відповіді з сервера, редагувати стилі та взаємодіяти з dom-деревом.

Результатом використання цих допоміжних програм та сервісів, розробка проєкту повинна бути більш ефективною та продуктивною, дозволяючи швидко тестувати, відлагоджувати та керувати різними аспектами проєкту.

Загалом, обрання Visual Studio Code як ІДЕ та інших перерахованих інструментів для реалізації проєкту "Тренажер-словник з англійської мови" дозволяє забезпечити робочий процес на високому рівні.

3 ІНФОРМАЦІЙНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

3.1 Програмна реалізація

3.1.1 Початок розробки

Під час початкової реалізації проєкту важливо правильно організувати структуру каталогів. Це є ключовим аспектом, оскільки без належної архітектури стає складним орієнтуватися в проєкті, пізніше, коли кількість файлів стане більшою, пошук конкретного файлу або усунення помилок буде забирати багато часу. Згідно з вище наведеними інструкціями, веб-додаток складається з клієнтської та серверної частин, які є окремими програмами. Отже, основними каталогами у файловій структурі проєкту є "client" та "server" (рис. 3.1).

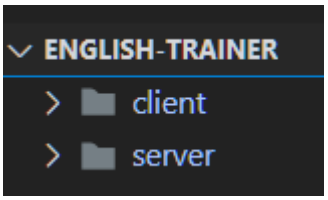

Рисунок 3.1 — Структура початкових директорій проєкту

Розпочнемо з "client", де необхідно розгорнути проєкт Angular. Це можна зробити за допомогою команди в терміналі: "ng new project_name". Angular допомагає знаходити початкову архітектуру клієнтського додатку, заснованого на компонентах та модулях. Компоненти є основними будівельними блоками, з яких складається весь проєкт, включаючи сторінки веб-додатку. Щоб ефективно працювати з компонентами, необхідно керуватися модульною архітектурою створення додатку. Цей підхід дозволяє досягти правильної взаємодії модулів один з одним, зменшуючи зчепленність і збільшуючи згуртованість, в свою чергу це забезпечує їх високу самостійність (рис. 3.2). Такий підхід дозволяє перевикористовувати модулі, що зменшує час, покращує ефективність і якість розробки. Вибір конкретної

архітектури дозволяє забезпечити масштабованість та подальший розвиток проєкту разом із підтримкою.

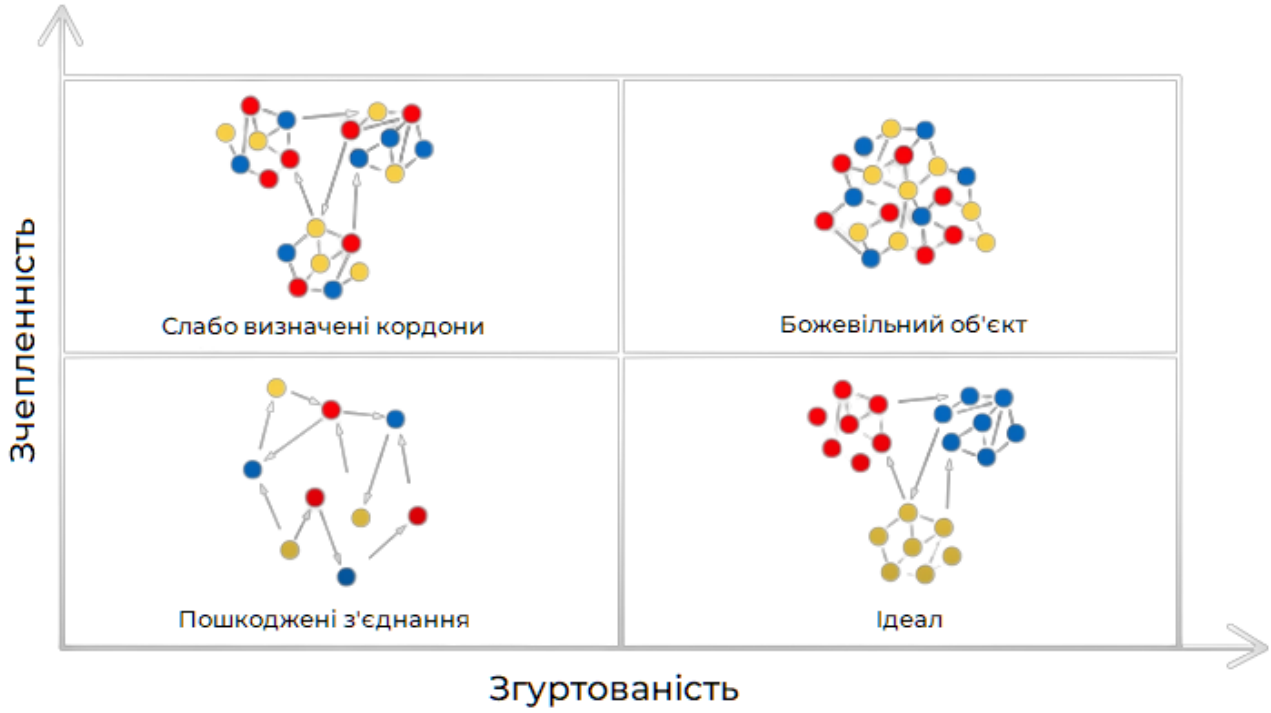

Рисунок 3.2 — Модуль архітектура додатку

Таким чином, відповідно до рисунку 3.2 та визначеному підходу під назвою “Ідеал” для створення архітектури було створено фізичну структуру клієнтської частини веб-додатку (рис. 1.3).

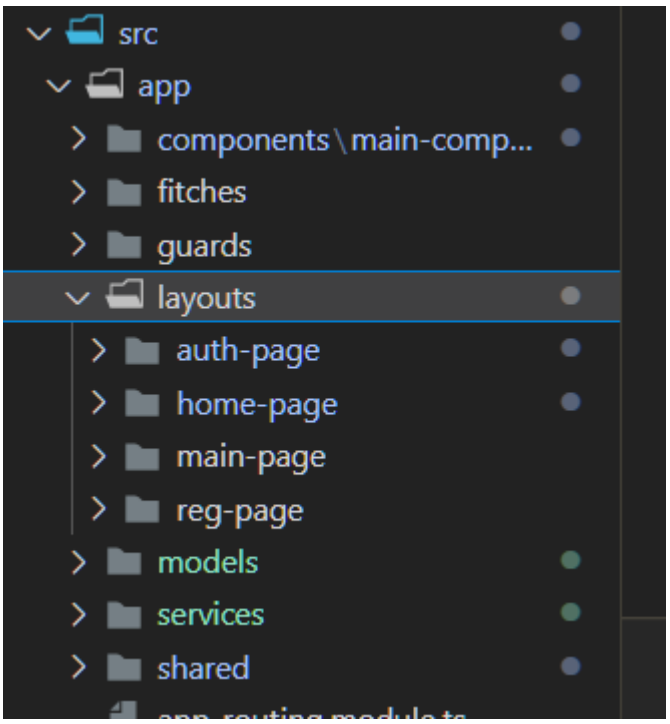

Рисунок 3.3 — Фізична структура клієнтської частини

У серверній частині також необхідно дотримуватися якісних підходів до розробки. Для створення серверних веб-додатків рекомендується використовувати структуру, яка базується на ієрархії функціональності. Основною задачею backend додатку є створення маршрутів, по яких можна звертатися до сервера та отримувати дані - створення API.

Перший та найвищий рівень ієрархії - це головний файл додатку, де потрібно реєструвати маршрути. Другий рівень - це самі маршрути та їх визначення. Третій рівень - контролери, в яких можна прописувати логіку роботи з маршрутами, обробляти дані та працювати зі статусами відповідей та запитами до сервера. Четвертий рівень - сервіси, які використовуються для роботи з базою даних.

Також існує проміжний етап між передачею даних запиту в контролер та його обробкою - middleware. Це універсальний компонент, який може мати різні типи. Він використовується для певних операцій з даними перед їх обробкою.

Підсумовуючи, фізична структура серверної частини додатку буде виглядати наступним чином (рис. 3.4).

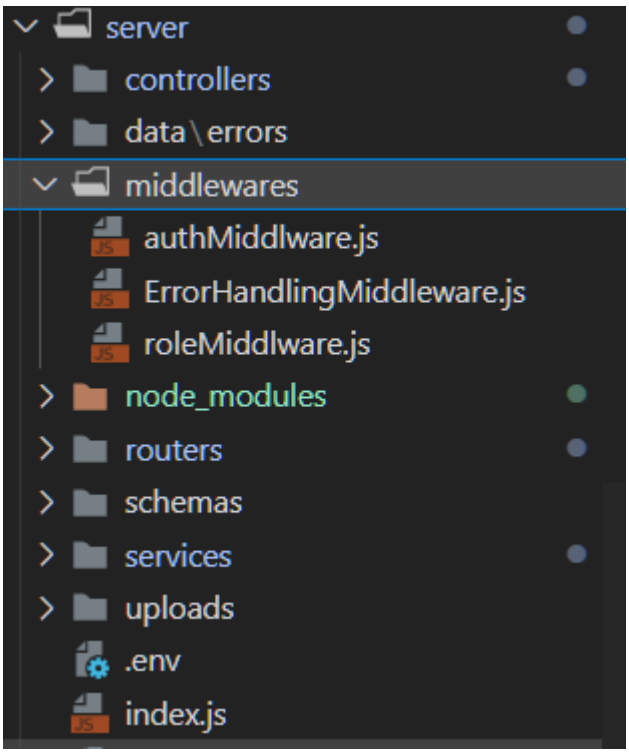

Рисунок 3.4 — Фізична структура серверної частини

3.1.2 Авторизація

Розробка системи авторизації на клієнтській та серверній сторонах є важливим етапом в багатьох проєктах, а в даному проєкті вона є не від'ємною частиною. Авторизація дозволяє ідентифікувати користувачів, контролювати доступ до ресурсів та забезпечувати безпеку даних.

Реалізація авторизації на клієнтській та серверній сторонах дозволяє забезпечити безпеку доступу до ресурсів додатку, захистити користувачів від несанкціонованого доступу та зберегти конфіденційність їх даних. Це особливо важливо для проєктів, які зберігають чутливу інформацію або вимагають обміну даними між користувачами.

Крім того, розробка системи авторизації допомагає створити персоналізований досвід користувача, де кожен користувач може мати свій обліковий запис з власними налаштуваннями та прогресом. Це може підвищити залученість користувачів та забезпечити зручну навігацію та використання додатку.

У випадку нашого проєкту, система авторизації може дозволити користувачам створювати власні облікові записи, зберігати свої улюблені слова та прогрес тренувань. Вона також може забезпечувати контроль доступу до приватних ресурсів, наприклад, статистики тренувань або розширених функцій.

У даному проєкті була реалізована авторизація за допомогою Angular на клієнтській стороні та Node.js на серверній стороні. Angular забезпечує потужний механізм для створення клієнтської частини додатку, зокрема форм для введення даних, обробників подій та маршрутизації. Він має вбудовану підтримку для роботи з HTTP-запитами, що дозволяє взаємодіяти з сервером для авторизації та отримання даних.

Окремий компонент registration.component (Додаток А – registration.components.html) було створено для реєстрації користувача (рис. 3.5). Нижче наведено опис кожного поля в структурі цього компонента:

- Поле "Ім'я": Призначене для зчитування імені користувача.
- Поле "Прізвище": Призначене для зчитування прізвища користувача.
- Поле "Email": Призначене для зчитування електронної пошти користувача.
- Поле "Пароль": Призначене для зчитування пароля користувача. Має можливість переключати відображення паролю між текстом і прихованим значенням за допомогою кнопки "visibility_off" і "visibility".
- Кнопка "Зареєструватися": Призначена для відправки форми реєстрації.
- Кнопка "google-btn": Призначена для реєстрації користувача за допомогою облікового запису Google, реалізована окремим компонентом.

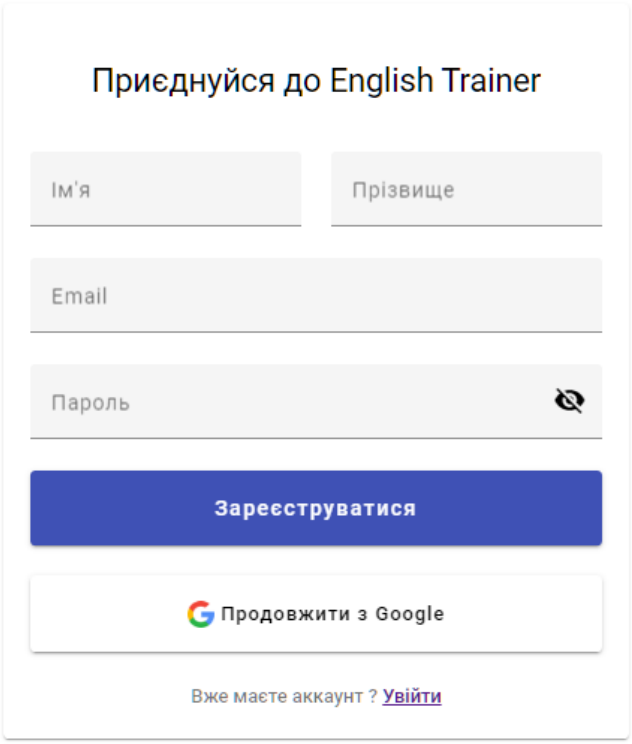

Рисунок 3.5 — Компонент реєстрації користувача

Перед відправкою даних на сервер повинна відбуватись певна перевірка для них, для цього створений користувацький валідатор (Додаток Б – validator.ts), він має функції перевірки, які використовуються для валідації введених даних у формі реєстрації:

- validateUserNameField(): Функція для валідації поля "Ім'я" та "Прізвище". Вона перевіряє правильність введеного значення і повертає помилку, якщо дані не задовольняють вимоги.
- validateEmailField(): Функція для валідації поля "Email". Вона перевіряє правильність формату введеної електронної пошти і повертає помилку, якщо формат некоректний.
- validatePasswordField(): Функція для валідації поля "Пароль". Вона перевіряє правильність введеного пароля, наприклад, його довжину або наявність певних символів, і повертає помилку, якщо пароль не задовольняє вимоги.

Ці функції дозволяють здійснювати перевірку введених даних і виводити відповідні повідомлення про помилки користувачеві (рис. 3.6).

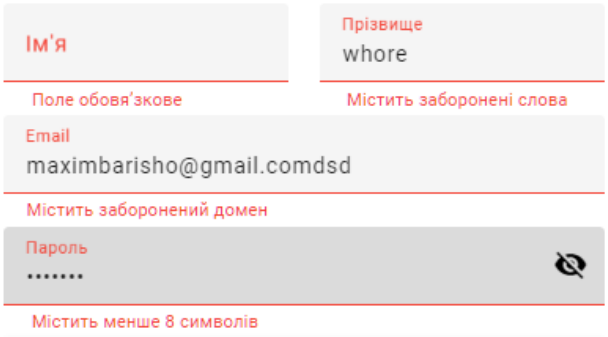

Рисунок 3.5 — Приклад роботи валідатора

Також важливою перевагою створення такого користувацького валідатора є можливість використовувати його для перевірки будь-яких інших вхідних даних, які вводить користувач. Наприклад, його повторне використання знадобилось для перевірки полів форми авторизації, що дозволило перевіряти введені дані в цій формі з використанням тих самих правил валідації, що і для форми реєстрації.

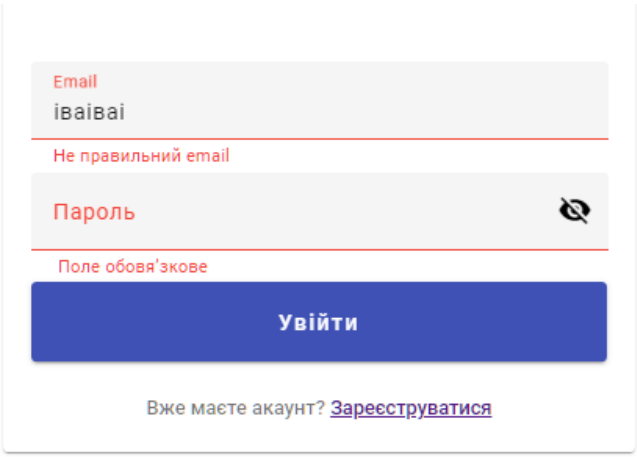

Рисунок 3.5 — Приклад роботи валідатора

Це спрощує процес перевірки даних і забезпечує одноманітність правил валідації на різних формах.

Також було впроваджено authorization service (Додаток В – authorization.service.ts) для взаємодії з сервером. В цьому сервісі реалізовані наступні функції:

- register: Функція, яка виконує реєстрацію користувача.

- login: Функція, яка виконує процес авторизації користувача, отримання токену з серверу на клієнт, та подальше його збереження у локальному сховищі, та оновлення глобальної змінної loggedIn.
- logout: Функція, яка виконує вихід з облікового запису користувача, за рахунок зміни loggedIn та видалення токену з локального сховища.
- verifyToken: Функція, яка виконує перевірку токену на валідність.

Ці функції дозволяють користувачам зареєструватись, увійти до системи та вийти з неї відповідно.

Структура серверної частини включає наступні елементи:

RoleSchema: Це схема ролей, яка входить у схему користувача (UserSchema). Поля RoleSchema описують роль користувача і включають, назву ролі, дозволи та іншу інформацію, що відноситься до ролі (Додаток Г – Role.js).

UserSchema: Це схема користувача, яка містить поля для зберігання даних про користувача (Додаток Г – User.js). Деякі з цих полів включають:

- userName: рядок, який представляє ім'я користувача.
- userSurname: рядок, який представляє прізвище користувача.
- email: рядок, який представляє електронну пошту користувача. Це поле також має унікальний обмеження (unique: true), що означає, що кожен користувач повинен мати унікальну електронну пошту.
- password: рядок, який містить пароль користувача.
- roles: масив ролей, до яких належить користувач. Це поле використовує посилання (ref: 'Role') для зв'язку зі схемою RoleSchema.

AuthRouter: Це маршрутизатор аутентифікації, який містить різні маршрути для обробки реєстрації, входу та перевірки авторизації користувача. Наприклад, AuthRouter.post('/registration', AuthController.registration) відповідає за обробку запиту на реєстрацію користувача (Додаток Г – AuthRouter.js).

AuthController: Це контролер аутентифікації, який містить функції обробки запитів, пов'язаних з аутентифікацією. Наприклад, AuthController.registration відповідає за реєстрацію користувача (Додаток Г – AuthController.js).

AuthMiddleware: Це проміжний обробник аутентифікації, який перевіряє, чи користувач авторизований перед продовженням обробки запиту (Додаток Г – AuthMiddleware.js). Він перевіряє наявність токена авторизації в заголовках запиту, розкодовує його та передає декодовані дані запиту для подальшого використання. Якщо авторизація не пройшла успішно, він генерує помилку (рис. 3.6).

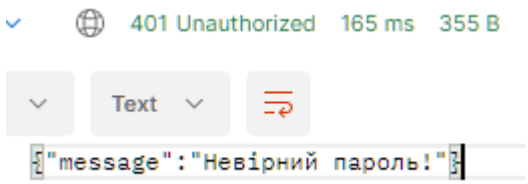

Рисунок 3.6 — Помилка яка надійшла після невдалої авторизації

3.2 Основний функціонал

Основний функціонал проєкту складається з декількох частина. Перша це можливість створювати колекції слів. Особливістю стало те що користувач може створювати колекції в колекціях, тобто по суті реалізовано файлова система. Це дозволить користувачу ділити слова по колекціях і гнучко управлятися з ними (рис. 3.6).

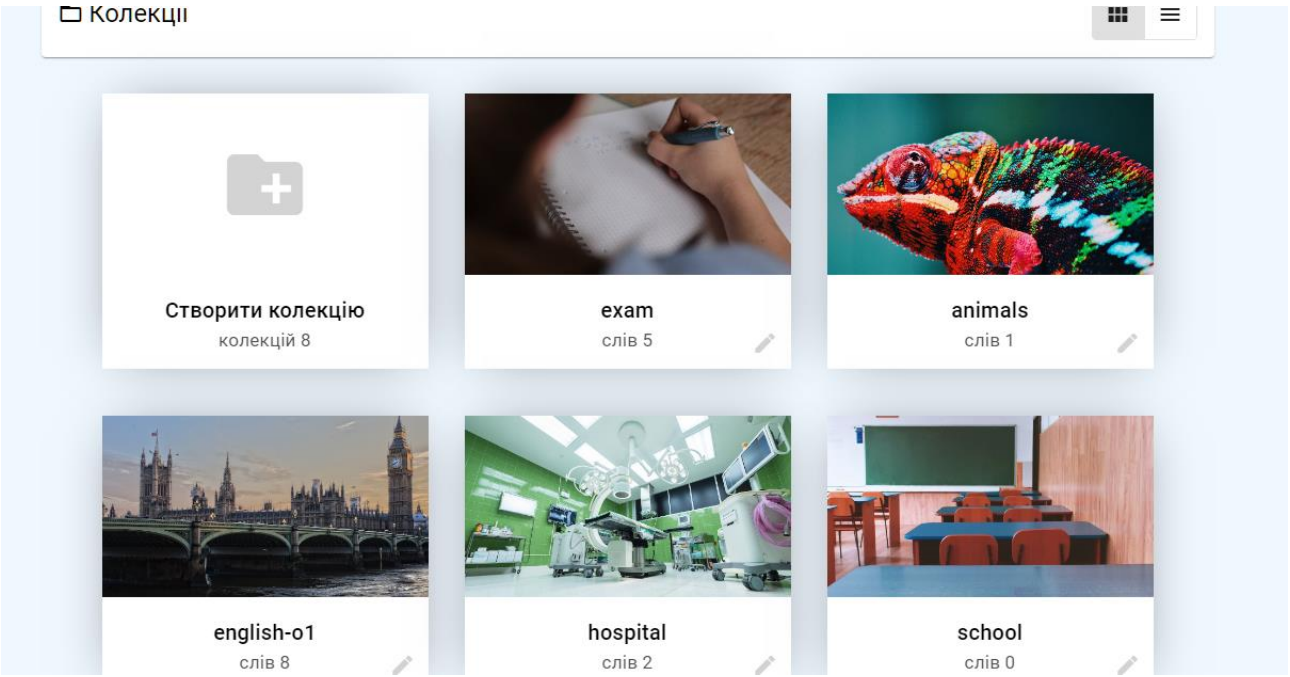

Рисунок 3.6 — Колекції слів

Звісно до колекцій також відноситься їх редагування, видалення і додавання. Що було реалізовано за допомогою створення відповідних діалогових вікон (рис. 3.7)

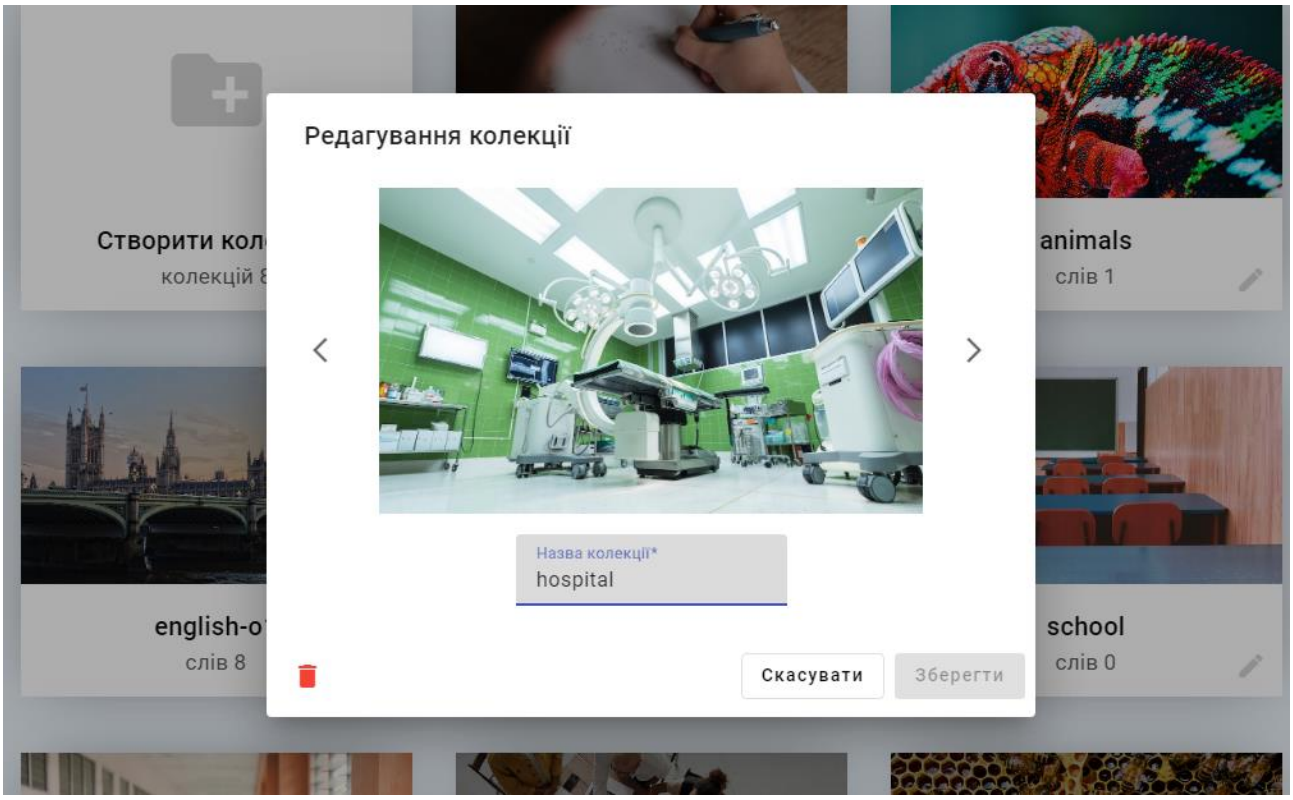

Рисунок 3.7 — Редагування колекції

Другою основною можливістю на сайті стало те, що у створені колекції реалізована можливість переглядати їх, шукати та додавати нові слова та вирази(рис 3.8).

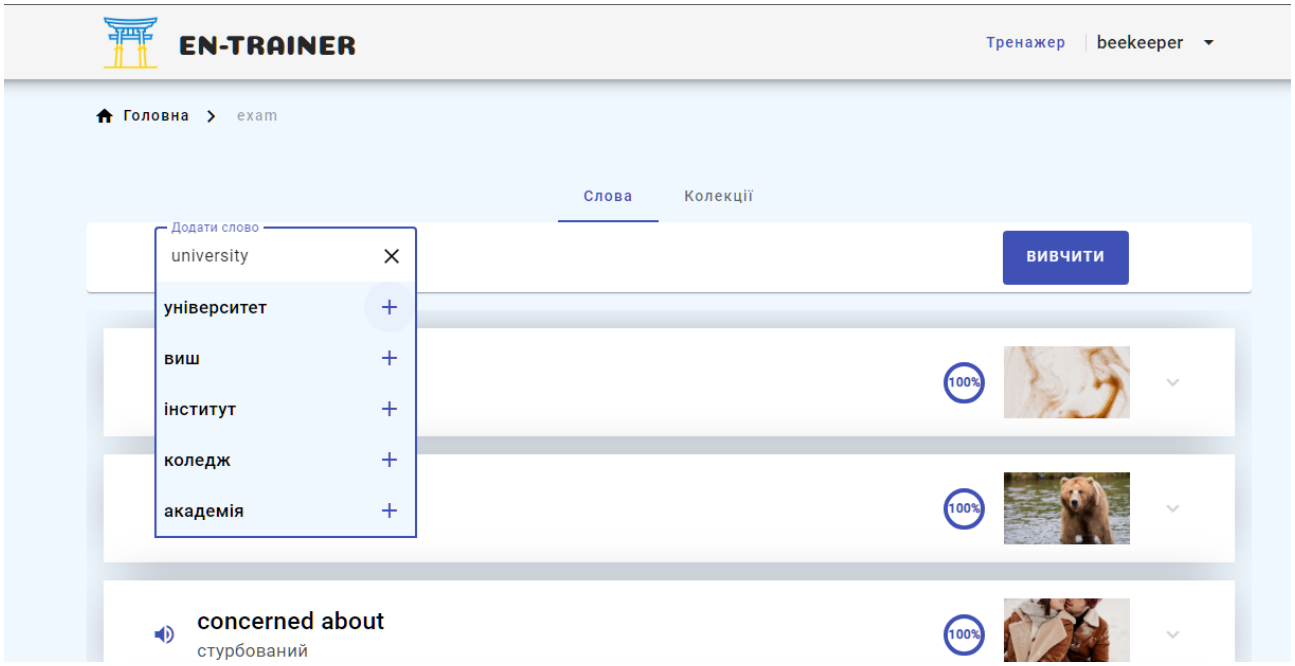

Рисунок 3.8 —Додавання нових слів до колекції

І третьою основною функцією сайта є те, що на ньому можна тренувати та повторювати слова. Режим тренування представляє собою тренажер із завдань різного рівня складності у відповідності і зрізним статусом вивчення. Перший тип завдань це звичайний вибір користувач повинен вибрати правильний переклад слова на англійську мову (рис. 3.10).

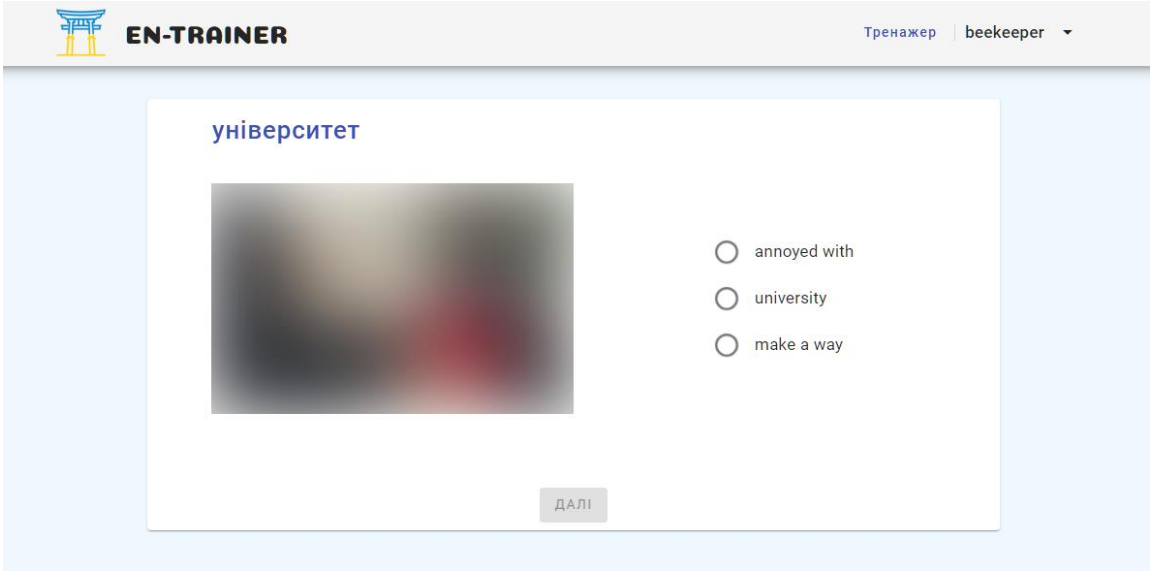

Рисунок 3.10 — Завдання на вибір правильного перекладу

Другий рівень завдань, є завданням на слухання. Користувач по озвучці слова вибрати правильний варіант відповді. (рис. 3.11)

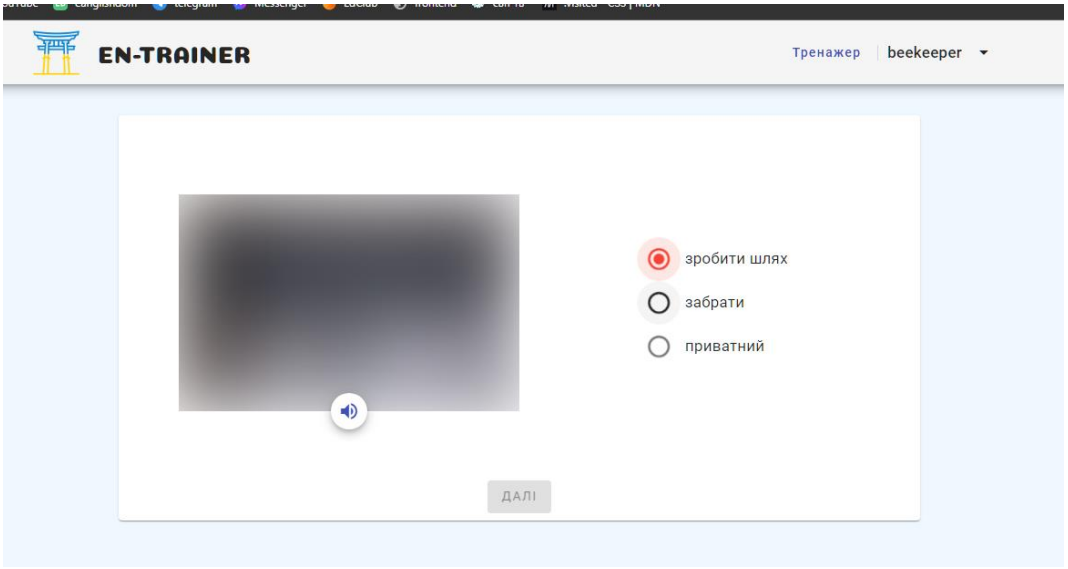

Рисунок 3.11 — Завдання на слухання

Третій тип завдань, це аудіо-пазл. Для виконання цього завдання користувач повинен написати правильно слово, яке було озвучене. І для того щоб він це зробив, йому дається пазл із букв. (рис 3.12)

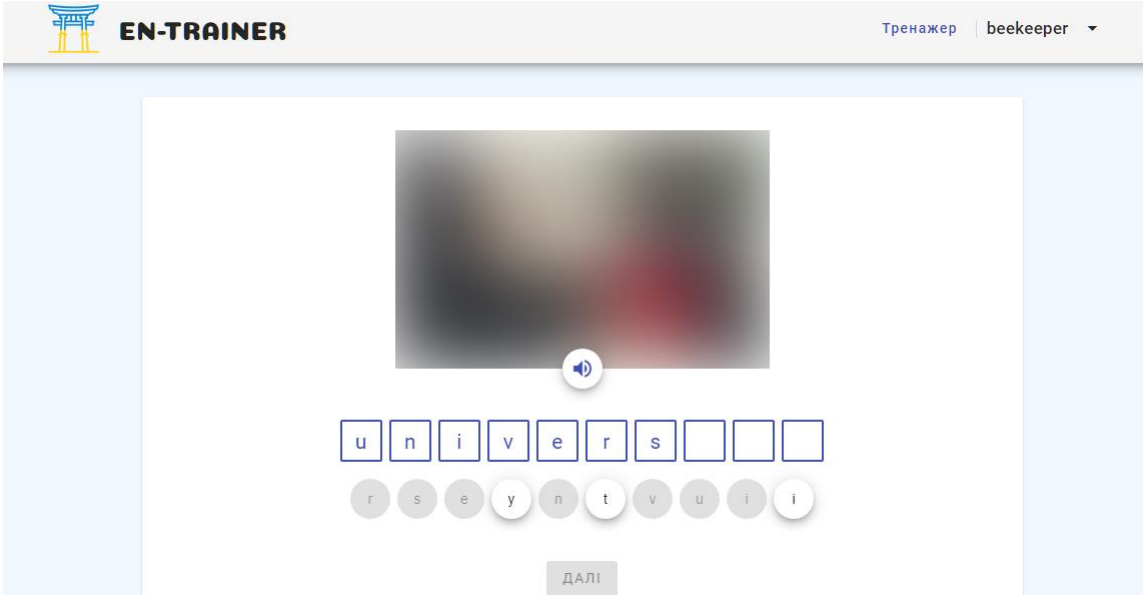

Рисунок 3.12 — Завдання аудіо-пазл

І останнім рівнем для отримання статусу вивченого слова є написання правильного слова в англійському варіанті. (Рисунок 3.13)

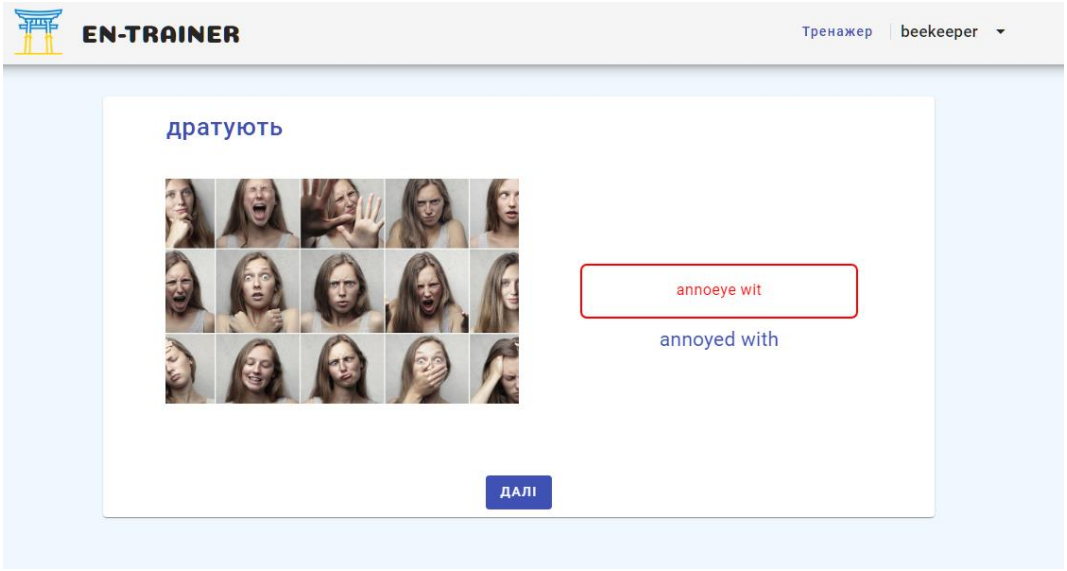

Рисунок 3.13 — Завдання на написання

І виконуючи поступово всі завдання користувач може зробити статус слова 100% тим самим вивчити його. Далі слово буде час від часу з’являтися на сторінці для повторення слів. А також буде відображено у статистиці подібним чином (рис. 3.14)

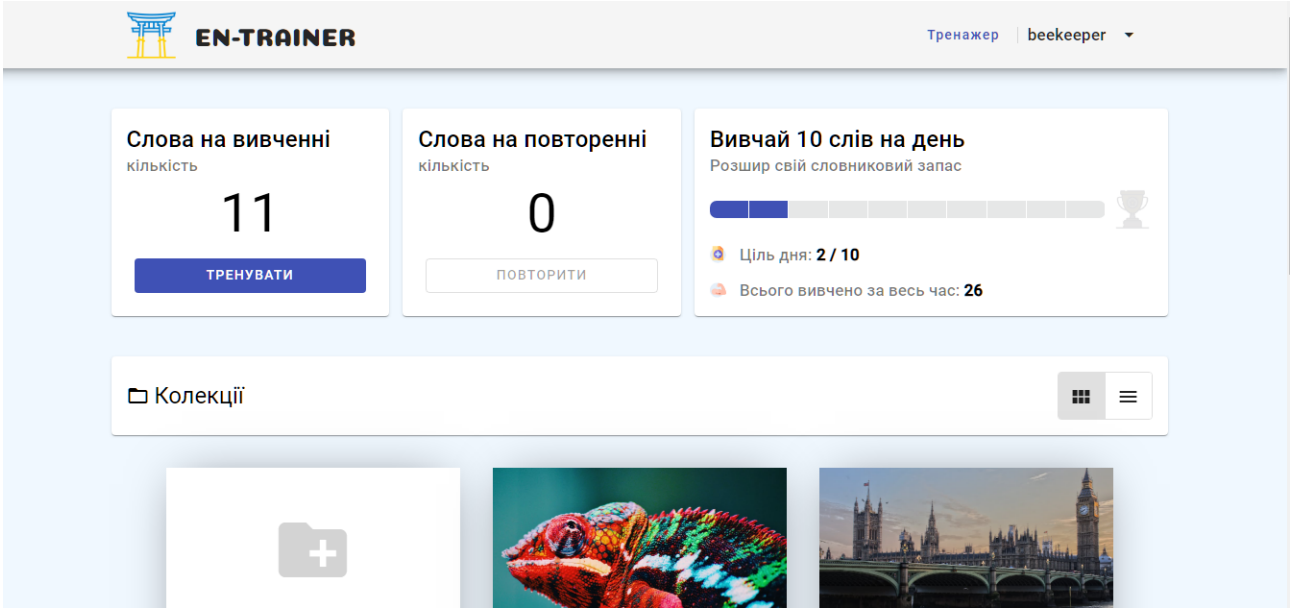

Рисунок 3.14 — Панель статистики повторення.

ВИСНОВКИ

Для виконання кваліфікаційної роботи було розглянуто і досліджено достатня кількість різних аспектів проєкту, зокрема структури бази даних, архітектури back-end та front-end, реєстрації та авторизації користувачів, основного функціоналу додатку, зокрема тренажера слів, пошуку нових слів та додавання їх до колекцій, файлова система колекцій, перегляд результатів, повторення слів.

У першому пункті "Інформаційний огляд" було проведено аналіз проблемної області та існуючих web-ресурсів, що пов'язані з нашим проєктом. Також була обґрунтована актуальність розробки проєкту, а постановленою задачею було сформульовано основну мету проєкту.

У другому пункті "Вибір методу рішення" було розглянуто питання вибору програмної реалізації та середовища розробки для проєкту.

Третій пункт "Інформаційне та програмне забезпечення системи" містить опис програмної реалізації проєкту, зокрема початок розробки та основний функціонал додатку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Галицький Б. Фреймоврки та їх значення у проєкті / Богдан Галицький. – Київ, 2019. – 180 с.
2. Robbins, J. N. (2018). Learning Web Design: A Beginner's Guide to HTML, CSS, JavaScript, and Web Graphics (5th ed.). O'Reilly Media, 2018 – 500 p.
3. Computer programming [Електронний ресурс] https://www.khanacademy.org/computing/computer-programming/sql/sql-basics/v/welcome-to-sql
4. J. Duckett, HTML and CSS: Design and Build Websites. Wiley, 2011.
5. Хантів К. Конверсія сайту. Перетворюємо відвідувачів на покупців / Кен Хантів., 2018. – 250 с.
6. M. Myers, A Smarter Way to Learn HTML & CSS: Learn it faster. Remember it longer. Createspace Independent Pub, 2015, vol. 1.
7. Bourne M. The success and failure of performance measurement initiatives: Perceptions of participating managers / Bourne M., Neely A., Platts K., Mills, J // Int. J. Oper. Prod. Manag. 2002 – pp. 22, 1288–1310
8. 10 іноземних мов, які українці найчастіше шукають в Google [Електронний ресурс]. URL: https://sostav.ua/publication/10-nozemnikh-mov-yak-ukra-nts-najchast-she-shukayut-v-google-88759.html.
9. Друга державна мова в Україні: міфи та реальність [Електронний ресурс]. URL: https://tsn.ua/blogi/themes/law/druga-derzhavna-mova-v-ukrayini-mifi-ta-realnist-353878.html.
10. The 10 Best Instagram Accounts for Learning English [Електронний ресурс]. URL: https://oxfordhousebcn.com/en/the-10-best-instagram-accounts-for-learning-english/.
11. VSCode Extensions [Електронний ресурс]. URL: https://eugenemir.com.ua/page/vscode-extensions.

ДОДТОК А

Registration.component.html

```
<div *ngIf="this.validator.checkLoadingData()" class="loader">
  <mat-spinner></mat-spinner>
</div>
<mat-card class="register">
  <mat-progress-bar
   mode="indeterminate"
   class="progress-bar"
   *ngIf="isRegistrationInProgress">
  </mat-progress-bar>
  <mat-card-title class="register__title">
    <h1>{{formTitle}}</h1>
  </mat-card-title>

    <mat-card-content class="register__content">
      <form [formGroup]="registrationForm"
            (ngSubmit)="onSubmit()"
           class="register__form">
        <div class="row"
           [ngClass]="{'spotForMatError' : addSpotForMatError()}">
            <mat-form-field>
              <mat-label>Ім'я</mat-label>
              <input
               matInput
               formControlName="name"
               class="matHeight">
                <mat-error >
                  {{ this.validator.getValidationErrorMessage('name') }}
                </mat-error>
            </mat-form-field>

            <mat-form-field>
              <mat-label>Прізвище</mat-label>
              <input
               matInput
               formControlName="surname"
               class="matHeight">
              <mat-error>
                {{ this.validator.getValidationErrorMessage('surname') }}
              </mat-error>
            </mat-form-field>

        </div>
        <mat-form-field>
          <mat-label>Email</mat-label>
          <input
           matInput
           placeholder="en_trainer@gmail.com"
           formControlName="email"
           class="matHeight">
          <mat-error>{{ this.validator.getValidationErrorMessage('email') }} </mat-
error>
        </mat-form-field>

        <mat-form-field>
          <mat-label>Пароль</mat-label>
          <input
           matInput [type]="hide ? 'password' : 'text'"
           formControlName="password"
           class="matHeight">
          <button
           mat-icon-button
           matSuffix
           type="button"
           (click)="hide = !hide"
```

```
            [attr.aria-label]="'Hide password'"
            [attr.aria-pressed]="hide">
             <mat-icon>{{hide ? 'visibility_off' : 'visibility'}}</mat-icon>
           </button>
           <mat-error >
             {{ this.validator.getValidationErrorMessage('password') }}
           </mat-error>
         </mat-form-field>

         <button
          mat-mdc-form-field-type-mat-input
          mat-raised-button
          color="primary"
          type="submit"
          class="registerFormFontSize">
             Зареєструватися
         </button>
         <app-mat-field-padding></app-mat-field-padding>
         <app-google-reg-btn class="registerFormFontSize"></app-google-reg-btn>
       </form>
    </mat-card-content>
    <app-mat-field-padding></app-mat-field-padding>
    <mat-card-subtitle>Вже маєте аккаунт ? <a routerLink="/auth">Увійти</a></mat-
card-subtitle>
  </mat-card>
```

Login.component.html

```
<div *ngIf="this.validator.checkLoadingData()" class="loader">
    <mat-spinner></mat-spinner>
</div>
<mat-card class="register">
    <mat-progress-bar
       mode="indeterminate"
       class="progress-bar"
       *ngIf="isLoginInProgress">
    </mat-progress-bar>
    <mat-card-title class="register__title">
    <h1>{{formTitle}}</h1>
    </mat-card-title>
    <mat-card-content class="register__content">
    <form [formGroup]="loginForm"
          (ngSubmit)="onSubmit()"
          class="register__form">

        <mat-form-field>
            <mat-label>Email</mat-label>
            <input
               matInput
               placeholder="en_trainer@gmail.com"
               formControlName="email"
               class="matHeight">
            <mat-error>{{ this.validator.getValidationErrorMessage('email') }} </mat-error>
        </mat-form-field>

        <mat-form-field>
            <mat-label>Пароль</mat-label>
            <input
               matInput [type]="hide ? 'password' : 'text'"
               formControlName="password"
               class="matHeight">
            <button
               mat-icon-button
               matSuffix
               type="button"
               (click)="hide = !hide"
               [attr.aria-label]="'Hide password'"
               [attr.aria-pressed]="hide">
```

```
                <mat-icon>{{hide ? 'visibility_off' : 'visibility'}}</mat-icon>
            </button>
            <mat-error>
                {{ this.validator.getValidationErrorMessage('password') }}
            </mat-error>
        </mat-form-field>

        <button
           mat-mdc-form-field-type-mat-input
           mat-raised-button
           color="primary"
           type="submit"
           class="registerFormFontSize">
            Увійти
        </button>
    </form>
    </mat-card-content>
    <app-mat-field-padding></app-mat-field-padding>
    <mat-card-subtitle>Вже маєте акаунт? <a
routerLink="/registration">Зареєструватися</a></mat-card-subtitle>
</mat-card>
```

ДОДТОК Б

validator.ts

```
import { ValidatorFn, AbstractControl } from "@angular/forms";
import { ValidationDataService } from "../services/validation-data.service";
import { IForbiddenWord } from "../models/forbiddenWords.model";
import { FormGroup } from "@angular/forms";
import { IAllowedDomain } from "../models/allowedDomain.mode";


export class Validator {
    public forbiddenWordsList: IForbiddenWord[];
    public allwedDomainsList: IAllowedDomain[];
    private _formGroup: FormGroup;
    private _loadingData = {
        "forbiddenWords": true,
        "allowedDomains": true
    }


    constructor(private validationData: ValidationDataService) {
    }

    get loadingData() {
        return this._loadingData
    }

    get formGroup() {
        return this._formGroup;
    }

    set formGroup(newFormGroup: FormGroup) {
        this._formGroup = newFormGroup
    }

    public checkLoadingData(): boolean {
        return Object.values(this.loadingData).includes(true);
    }

    public getValidationData():void {
        this.validationData.getForbiddenWords().subscribe(this.initForbiddenWords.bind(
this))
        this.validationData.getAllowedDomains().subscribe(this.initAllowedDoamains.bind
(this))
    }

    private initForbiddenWords(data: IForbiddenWord[]): void {
        this.forbiddenWordsList = data;
        this._loadingData.forbiddenWords = false;
    }

    private initAllowedDoamains(data: IAllowedDomain[]):void  {
```

```
        this.allwedDomainsList = data;
        this._loadingData.allowedDomains = false;
    }

    validateUserNameField(): ValidatorFn {
        return (control: AbstractControl): {[key: string]:any } | null => {
            const value = control.value as string;

            if (!value) {
                return { required: true };
            }

            if (this.forbiddenWordsList.some(word => word.word ===
value.toLocaleLowerCase())) {
                return { forbiddenWord: true };
            }

            if (value.length < 2) {
                return { minLength: true };
            }

            if (value.length > 50) {
                return { maxLength: true };
            }

            const pattern = /^[a-zA-Za-яA-ЯiїєIЇЄ'`-]+$/;

            if (!pattern.test(value)) {
                return { pattern: true };
            }

            return null
        }
    }

    validateEmailField(): ValidatorFn {
        return (control: AbstractControl): { [key: string]: any } | null => {
            const email = control.value as string;

            if (!email) {
                return { required: true };
            }

            const emailRegex = /^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/;

            if (!emailRegex.test(email)) {
                return { 'invalidEmail': true };
            }

            const emailDomain = email.split('@')[1];
            const isDomainAllowed = this.allwedDomainsList.some(email => email.domain
=== emailDomain);

            if (!isDomainAllowed) {
              return { 'forbiddenDomain': true };
```

```
        }

        return null
    };
}

validatePasswordField(): ValidatorFn {
    return (control: AbstractControl): {[key: string]:any } | null => {
        const value = control.value as string;

        if (!value) {
            return { required: true };
        }

        if (value.length < 8) {
            return { minPaswordLength: true };
        }

        const passwordAtLeastOneDigit = /^(?=.*\d).{6,}$/;

        if (!passwordAtLeastOneDigit.test(value)) {
            return { passwordAtLeastOneDigit: true };
        }

        return null
    }
}

public getValidationErrorMessage(controlName: string): string {
    const control = this.formGroup.get(controlName);
    if (!control) {
      return '';
    }

    const errorMessages = [
      { error: 'required', message: 'Поле обовя’зкове' },
      { error: 'invalidEmail', message: 'Не правильний email' },
      { error: 'forbiddenWord', message: 'Містить заборонені слова' },
      { error: 'minLength', message: 'Містить менше 2 символів' },
      { error: 'maxLength', message: 'Містить більше 50 символів' },
      { error: 'pattern', message: 'Містить не коректні символи' },
      { error: 'minPaswordLength', message: 'Містить менше 8 символів' },
      { error: 'passwordAtLeastOneDigit', message: 'Повинен містити хоча б 1 число' },
      { error: 'forbiddenDomain', message: 'Містить заборонений домен' },

    ];

    for (const errorMessage of errorMessages) {
      if (control.hasError(errorMessage.error)) {
        return errorMessage.message;
      }
    }

    return '';
  }
```

```
}
```

ДОДТОК В

Authorization.service.ts

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable, tap } from 'rxjs';
import { BASE_URL } from 'src/assets/data/global-variables';
import { Router } from '@angular/router';

@Injectable({
  providedIn: 'root'
})
export class AuthorizationService {
  public loggedIn: boolean;
  public isLoading: boolean = true;
  public authorizationError: any;

  constructor(private http: HttpClient,
    private router: Router) {

  }

  public checkAuthorizationStatus()  {
    this.isLoading = true;
    const token = localStorage.getItem('token')
    if(token) {
      this.verifyToken(token).subscribe(
        {
          next: () => {
            this.isLoading = false;
          },
          error: (e) => {
            this.isLoading = false;
            localStorage.removeItem('token');
            this.router.navigate(['/auth'])
            this.authorizationError = e.error.message;
          }
        }
      )
    }
  }

  set isloggedIn(state: boolean) {
    this.loggedIn = state;
  }

  get isloggedIn(): boolean {
    return this.loggedIn;
  }

  registerUser(userData: any): Observable<any> {
    return this.http.post(`${BASE_URL}/registration`, userData);
```

```
  }

  public verifyToken(token: string): Observable<any> {
    return this.http.post(`${BASE_URL}/check`, { token }).pipe(
      tap((response: any) => {
        if (response.newToken) {
          localStorage.removeItem('token')
          localStorage.setItem('token', response.newToken);
          this.isloggedIn = true;
        }
      })
    );
  }

  login(credentials: any): Observable<any> {
    return this.http.post<any>(`${BASE_URL}/login`, credentials).pipe(
      tap((response) => {
        // Check if the response contains a token
        if (response.token) {
          // Store the token in localStorage
          localStorage.setItem('token', response.token);
          // Update the logged-in status
          this.loggedIn = true;
        }
      })
    );
  }

  logout(): any {
    localStorage.removeItem('token');
    this.isloggedIn = true;
  }
}
```

ДОДТОК Г

Role.js

```
import { Schema, model } from "mongoose";

const Role = new Schema({
    value: {type: String, required: true, unique: true, default: "USER" },
})

export default model('Role', Role)
```

User.js

```
import { Schema, model } from "mongoose";

const User = new Schema ({
    userName: {type: String, required: true},
    userSurname: {type: String, required: true},
    email: {type: String, required: true, unique: true},
    password: {type: String, required: true },
    roles: [{type: String, ref: 'Role'}]
})

export default model('User', User)
```

Router.js

```
const AuthRouter = Router();

AuthRouter.post('/registration', AuthController.registration)
AuthRouter.post('/login', AuthController.login)
AuthRouter.post('/auth', AuthMiddlware, AuthController.check)

export default AuthRouter;
```

AuthMiddleware.js

```
function AuthMiddlware(req, res, next) {
    if(req.method === "OPTIONS") {
        next()
    }

    try {
        const token = req.headers.authorization.split(' ')[1]
        if(!token) {
            return next(ApiError.forbidden("Користувач не авторизований!"))
            // return res.status(403).json({message: "Користувач не авторизований"})
        }

        const decodedData = Jwt.verify(token, process.env.SECRET_KEY)
        req.user = decodedData
        next()
    } catch (err) {
```

```
        console.log(err)
        return next(ApiError.forbidden("Користувач не авторизований!"))
        // return res.status(403).json({message: "Користувач не авторизований"})
    }
}

export default AuthMiddlware
```

AuthController.js

```
function AuthController() {

    return {
        async registration (req, res, next) {
            try {
                const { name, surname, email, password } = req.body;

                const isUserExist = await User.findOne({email: email});

                if(isUserExist) {
                    return next(ApiError.unprocessableEntity("Користувач вже існує!"))
                }

                const hashedPassword = await AuthService.hashPassword(password);

                const userRole = await Role.findOne({value: "USER"});
                const user = await User.create({
                    userName: name,
                    userSurname: surname,
                    email: email,
                    password: hashedPassword,
                    roles: [userRole.value]
                })

                res.status(200).json({message: "Користувач успішно зареєстрований!"});
            }
            catch(error) {
               next(error)
            }
        },

        async login (req, res, next) {
            try {
                const {email, password} = req.body;
                const user = await User.findOne({email: email})
                if(!user) {
                    return next(ApiError.authorize("Акаунту не існує!"))
                }

                let comparePassword = bcrypt.compareSync(password, user.password)

                if(!comparePassword) {
                    return next(ApiError.authorize("Невірний пароль!"))
                }
```

```
            const token = AuthService.generateGWT(user._id, user.email, user.roles)
            res.status(200).json({token: token})
        }
        catch(error) {
            next(error)
        }
    },

    async check (req, res, next) {
        const token = AuthService.generateGWT(req.user.id, req.user.email,
req.user.role)
        return res.json({newToken: token})
    },

    async getUsers (req, res) {
        try {
            res.json(200)
        } catch (error) {
            console.log(error)
            res.status(500).json("Error with getting database field users")
        }
    }
}
}
```