МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
МАРІУПОЛЬСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ЕКОНОМІКО-ПРАВОВИЙ ФАКУЛЬТЕТ
КАФЕДРА СИСТЕМНОГО АНАЛІЗУ
ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

До захисту допустити:
В.о. зав. кафедри

Ганна МАРТИНЮК

«04» червня 2025 р.

ІНТЕРАКТИВНИЙ ВЕБ-САЙТ ДЛЯ ВДОСКОНАЛЕННЯ НАВИЧОК АНГЛІЙСЬКОЇ МОВИ: ДИЗАЙН І РЕАЛІЗАЦІЯ»

Кваліфікаційна робота
здобувача вищої освіти першого (бакалаврського) рівня вищої освіти освітньо-професійної програми «Комп'ютерні науки»
Лутчак Артема Васильовича
Науковий керівник:
Дрейс Юрій Олександрович,
кандидат технічних наук, доцент,
доцент кафедри системного аналізу та інформаційних технологій
Рецензент:
Нечипорук Олена Петрівна,
доктор технічних наук, професор, завідувач кафедри інтелектуальних кібернетичних систем Державного університету «Київського авіаційного університету»

Кваліфікаційна робота захищена
з оцінкою відмінно 90 (А)
Секретар ЕК

«11» червня 2025 р.

Київ – 2025

ЗМІСТ

ВСТУП

У сучасному світі знати англійську мову хоча б на базовому рівні є необхідною потребою для освіти, подорожей, міжнародного спілкування, кар'єрних можливостей та культурного розвитку. З розвитком цифрових технологій онлайн-навчання стало популярним серед людей різного віку. Проте більшість існуючих платформ пропонують своїм користувачам платити гроші за навчання з реальним вчителем, коли не в кожного є достатньо грошей або часу для вивчення іноземної мови. Також такі веб-сайти не враховують індивідуальні потреби учнів, мають непривабливий інтерфейс та містять рекламу, що іноді відволікає учня. Деякі ресурси не адаптуються до рівня знань користувача або не надають якісного інтерактивного контенту.

Традиційні методи навчання, зокрема друковані підручники та заняття в аудиторії, не завжди бувають ефективними, оскільки вони не забезпечують достатньої інтерактивності та можливості збереження результатів. Натомість інтерактивні веб-платформи дозволяють адаптувати навчальний процес під користувача, забезпечуючи більш гнучкий та персоналізований підхід.

Розробка сучасного веб-сайту для вдосконалення навичок англійської мови є актуальною, оскільки він може поєднувати в собі різні види навчальних програм, такі як граматичні вправи, лексичні вправи та інші адаптивні завдання. Такий інтернет ресурс буде корисним як для самостійного навчання, так і для використання їх вчителями у навчальних закладах.

В останні роки кількість користувачів онлайн-ресурсів для вивчення мов зросла. Згідно з різними дослідженнями, інтерактивні методи навчання сприяють кращому засвоєнню матеріалу, ніж звичайні підходи. Популярність мобільного навчання та самостійного вивчення мови через веб-додатки постійно збільшується, що підтверджує необхідність створення ефективних платформ. Інтерактивний веб-сайт дозволяє комбінувати візуальний та практичний підхід до навчання.

Об’єктом дослідження є процес покращення навичок англійської мови за допомогою цифрових технологій. На даний момент в світі онлайн-ресурси та інтерактивні платформи відіграють важливу роль у вивченні іноземних мов, оскільки вони забезпечують доступність, зручність та персоналізацію навчального процесу. Розвиток цифрових інструментів дозволяє адаптувати навчання до потреб кожного користувача, що значно підвищує його ефективність.

Предмет дослідження включає в себе розробку дизайну та реалізацію інтерактивного веб-сайту, який дозволяє новим користувачам ефективно практикувати граматику, лексику та інші мовні навички. Дослідження охоплює створення зручного та привабливого інтерфейсу, впровадження тестових завдань з автоматичним оцінюванням, використання сучасних веб-технологій для забезпечення адаптивності сайту та інтеграцію механізмів відстеження прогресу користувачів.

Метою цієї роботи є створення веб-сайту для користувачів із різним рівнем володіння англійською мовою, який забезпечує швидке завантаження сторінок, адаптивність до різних типів пристроїв та браузерів, оптимальне використання пам’яті браузера, а також містить систему тестування із випадковим порядком завдань, що генеруються під час запуску тесту. Окремо від завдань будуть додані пояснення, які допоможуть підготуватися до виконання тестів.

Під час виконання кваліфікаційної роботи необхідно вирішити такі завдання:

- Дослідити сучасні методи інтерактивного навчання;
- Спланувати повну структуру свого веб-сайту та дотримуватися його реалізації;
- Розробити макет та по ньому виконати повний дизайн;
- Реалізувати основний функціонал (адаптивність, інтерактивні тести з різними запитаннями, збереження результату після виконання завдань)
- Після розробки виконати тестування та аналіз зручності використання сайту.

У процесі дослідження використовується теоретичні та практичні методи, що забезпечують комплексний підхід до розробки інтерактивного веб-сайту. На теоретичному рівні проведено аналіз вже існуючих аналогів та різних наукових праць, присвячених принципам розробки освітніх платформ та особливостям веб-дизайну. Досліджено сучасні підходи до створення навчальних веб-ресурсів, їхню структуру, функціональність та вплив на засвоєння матеріалу. Практична частина передбачає процес проектування та розробки веб-застосунку на основі сучасних технологій, таких як HTML, CSS, JavaScript, Node.js та баз даних. Особлива увага приділена забезпеченню адаптивності, інтуїтивного інтерфейсу та інтерактивності ресурсу.

Розроблений продукт може використовуватися як допоміжний інструмент для самостійного навчання з низьким рівнем володіння англійської мови або для впровадження в навчальні заклади для перевірки знань учнів. Застосування сучасних веб-технологій у розробці забезпечує легку масштабованість та можливість подальшого розширення функціональності платформи, що відкриває перспективи для використання в інших мовних та освітніх проектах.

Дипломна робота складається з трьох розділів. У першому розділі розглянуто сучасні методи створення програмного забезпечення, аналіз існуючих платформ та планування всієї архітектури проекту. У другому розділі висвітлено етапи розробки дизайну, реалізація макету та створення адаптивності під різні пристрої. У третьому розділі створюється основний функціонал та клієнт-серверна архітектура.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Методи та підходи до розробки веб-застосунків

Розробка веб-застосунків передбачає використання різних методологій, технологій та інструментів, які забезпечують ефективність, адаптивність і зручність для кінцевих користувачів. Успіх проектів з розробки програмного забезпечення значною мірою залежить від використовуваної методології. Методологія розробки програмного забезпечення - це структура, яка описує процес розробки програмного забезпечення. Вибір правильної методології для проекту має важливе значення для забезпечення успіху. Існує декілька методологій розробки програмного забезпечення, і вибір правильної може бути дуже складним. Серед основних підходів до створення веб-застосунків можна виділити класичну модель розробки (Waterfall), гнучкі методології (Scrum та Kanban), спіральна методологія та багато інших.

Методологія Waterfall - це лінійний підхід до розробки програмного забезпечення. Це традиційний підхід, який дотримується суворої послідовності кроків. Методологія Waterfall включає наступні етапи: збір вимог, проектування, програмування, тестування та супровід. Кожен етап завершується перед тим, як перейти до наступного. Така методологія є досить простою до розуміння та управління, оскільки передбачає послідовне виконання етапів розробки. Вона забезпечує зрозумілі вимоги до проекту ще на початковому етапі, що сприяє стабільності процесу розробки. Цей підхід особливо добре підходить для невеликих проектів із заздалегідь визначеними та незмінними вимогами. Але така модель не підходить для великих та складних проектів, оскільки нелегко пристосовується до змін та забирає багато часу.

Методологія Agile - це ітеративний підхід до розробки програмного забезпечення. Це гнучкий і спільний підхід, який передбачає безперервну доставку робочого програмного забезпечення, що дозволяє отримувати зворотній зв’язок від замовника продукту на кожному етапі розробки. Методологія Agile включає наступні

етапи: планування, проектування, розробка, тестування та випуск. Кожен етап виконується в ітераціях. Однією з ключових переваг є активна співпраця між командою розробників та замовником, що сприяє створенню продукту, який найбільш точно відповідає потребам користувачів. Попри численні переваги, методологія Agile має і недоліки. Вона вимагає високого рівня співпраці із замовником, що може бути проблемою у випадку обмеженої доступності. Також може бути важко точно оцінити необхідні ресурси та час для кожної ітерації, що може призвести до затримок у виконанні проекту.

Методологія Scrum є підмножиною методології Agile. Це ітеративний та інкрементний підхід до розробки програмного забезпечення, який передбачає використання спринтів. Спринт - це обмежений у часі період розробки, який зазвичай триває від двох до чотирьох тижнів. Методологія Scrum передбачає наступні ролі: власник продукту, скрам-майстер і команда розробників. Скрам-майстер слідкує за всіма процесами розробки продукту та видає членам роботи завдання. Члени команди постійно спілкуються один з одним, що дозволяє швидко вирішувати проблеми та адаптувати процес розробки. Завдяки чітко визначеним ролям та регулярним зустрічам, Scrum забезпечує прозорість і зрозумілість вимог до проекту на кожному етапі. Гнучкість цього підходу дозволяє легко пристосовуватися до змін, що особливо важливо для динамічних проектів. Але він вимагає високого рівня залученості клієнтів, оскільки ефективна робота команди залежить від постійного зворотного зв’язку. Крім того, управління великими проектами може бути складним, оскільки розподіл завдань між командами та синхронізація ітерацій потребує ретельного планування. Також Scrum може виявитися менш ефективним для проектів зі складними залежностями між компонентами, оскільки його гнучкість не завжди сприяє структурованому підходу до інтеграції всіх елементів системи.

Дошка Канбан розділена на колонки, які представляють етапи розробки. Кожне завдання представлене карткою, яка переміщується по дошці в міру проходження етапів розробки. Методологія Канбан передбачає наступні етапи: “потрібно зробити”, “в процесі” і “зроблено”. Це забезпечує візуальне представлення поточного стану

роботи, що дозволяє команді легко відстежувати виконання завдань і швидко реагувати на зміни. Вона сприяє безперервному постачанню робочого програмного забезпечення, оскільки не передбачає складних ітерацій, а натомість дозволяє гнучко змінювати пріоритети завдань. Такий підхід добре підходить для проектів із високим рівнем невизначеності, де вимоги можуть змінюватися в процесі розробки. Попри ці переваги, є і недоліки використання цієї методології. Управління великими проектами може бути складним, оскільки відсутність чітко визначених ітерацій ускладнює контроль за виконанням завдань. Крім того, Kanban не завжди є оптимальним вибором для проектів із високою складністю, де необхідна сувора структурованість процесу розробки. Також ця методологія може не забезпечувати чіткого розуміння вимог до проекту на початковому етапі, що може призвести до труднощів у плануванні та координації роботи команди[1].

Спіральна модель має досить складну структуру та покладається на зниженню ризиків на ранній версії продукту. Таку модель ділять на чотири етапи: планування, аналіз ризиків, розробка та оцінка. Стосовно процесу розробки ПЗ, розробники починають на низьких рівнях і досліджують властиві йому ризики. Далі розробники створюють план ітерацій по спіралі. Спіральна модель є гарним вибором для складних і довготривалих проектів. Однією з головних переваг є значне зниження факторів ризику завдяки поетапному аналізу та постійній оцінці можливих загроз. Ця методологія добре підходить для великих і складних проектів, оскільки дозволяє швидко пристосовуватися до нових вимог та додавати нові функції на пізніх етапах розробки. Крім того, вона є ефективним підходом для проектів із високим бізнес-ризиками, оскільки забезпечує поступове вдосконалення продукту та адаптацію до змін. Із мінусів, ця модель для впровадження є досить дорогим, оскільки процес аналізу ризиків та повторних ітерацій потребує значних ресурсів та часу. Помилки на етапі аналізу ризиків можуть завдати серйозної шкоди всьому проекту, що може призвести до його провалу. Крім того, цей підхід не є оптимальним для проектів із низьким рівнем ризику, де простіші моделі, такі як Waterfall або Agile, можуть бути більш ефективними[2].

Проаналізувавши існуючі методології, ми можемо зробити певний вибір на одній із них. У таблиці нижче представлено порівняння основних методологій за ключовими параметрами.

Таблиця 1.1. Порівняння методологій розробки програмного забезпечення

Методологія	Переваги	Недоліки	Найкраще підходить для
Waterfall	Простий в управлінні, має чіткі вимоги до проекту	Не підходить для великих проектів, не є гнучким	Малих проектів з чіткими вимогами
Agile(Гнучка)	Має безперервне постачання робочого ПО	Вимагає високого рівня залучення клієнтів, складний в управлінні великими проектами	Динамічних проектів, що змінюються
Scrum	Заохочує до співпраці, чіткі вимоги до проекту	Вимагає високого рівня залучення клієнтів, складний в управлінні великими проектами	Командної розробки з активною вза’ємодією
Kanban	Забезпечує візуальне представлення незавершеної роботи, враховує невизначеність	Складність в управлінні великими проектами, може не надавати чітких вимог до проекту	Проектів із постійним потоком завдань
Spiral(Спіральна)	Зменшує проектні ризики, легко адаптується до змін	Може бути трудомістким, складним в управлінні для великих проектів	Великих і складних проектів із невизначеними вимогами

1.2. Аналіз існуючих освітніх веб-платформ

Кожна з вже існуючих освітніх веб-платформ для навчання іноземної мови має свої характерні особливості, які визначають її ефективність засвоєнню матеріалу, функціональність, та зручність у використанні. Деякі з них орієнтовані на

запам’ятовування матеріалу через навчальні ігри, інші – на глибоке занурення у контекст мови за допомогою пояснювальних текстів та прослуховувань. Також існують ресурси, що роблять акцент на граматиці та тестових завданнях.

Аналізуючи подібні сервіси, важливо враховувати низку параметрів: доступність функціоналу, рівень інтерактивності, можливість адаптації під індивідуальні потреби користувача, наявність системи зворотного зв’язку та загальну логіку побудови навчального процесу. Вивчення переваг і недоліків кожної з платформ дозволяє визначити, які рішення вже реалізовані ефективно, а які – потребують доопрацювання. Це, в свою чергу, дає змогу сформувати чітке уявлення про вимоги до нашої майбутньої системи.

Першим прикладом такої освітньої платформи буде Perfect English Grammar. Він орієнтований переважно на опанування граматики й пропонує широкий спектр матеріалів: від пояснень граматичних правил до вправ, тестів і платних курсів.

Однією з переваг цього інтернет-ресурсу є простий, але зручний і не відволікаючий дизайн, який сприяє концентрації користувача на навчальному матеріалі. Структура сайту логічна: теми згруповані за категоріями, присутня функція пошуку, а також є можливість пройти тест на визначення рівня володіння англійською мовою. Крім того, користувачі можуть купити розширені навчальні курси, що робить платформу зручною для тривалого та поступового навчання.

Рис. 1.1. «Шапинка» веб-сайту Perfect English Grammar

В цей же час, ресурс має і певні недоліки, які можуть обмежувати досвід користувача. Зокрема, тест на перевірку рівня знань не змінюється – питання завжди подаються в одному порядку і мають фіксований вигляд, що знижує об’єктивність перевірки. Візуальне оформлення сторінок з поясненнями виглядає дещо застарілим і малопривабливим, що може негативно впливати на сприйняття інформації[3].

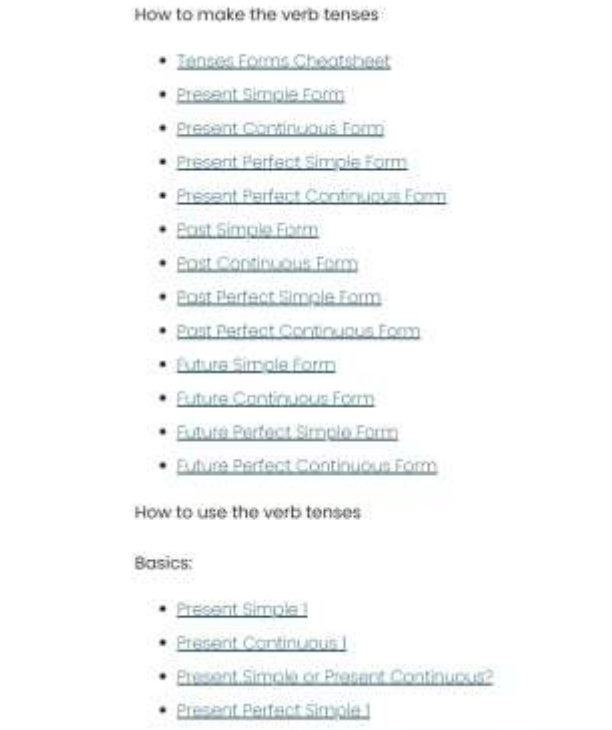


Рис. 1.2. Перелік тем із поясненнями

Ще одним аналогом освітньої веб-платформи для вивчення англійської мови є Test-English. Цей сайт поєднує інтерактивне навчання та перевірку знань користувача, пропонуючи широкий набір тематичних тестів, вправ і пояснень. Інтерфейс сайту простий, приємний для сприйняття та зручний у користуванні, що дозволяє швидко орієнтуватися навіть новим відвідувачам цього ресурсу. На сайті також доступна система пошуку, яка значно полегшує навігацію, а також тест на визначення рівня володіння англійською мовою.

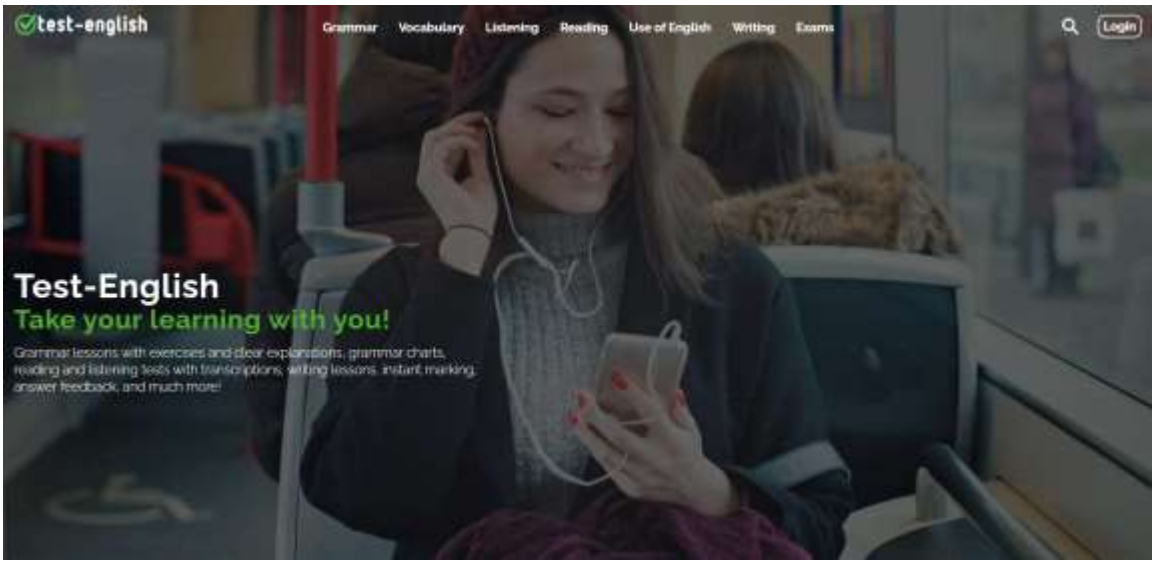


Рис. 1.3. «Шапинка» веб-сайту Test-English

Однією з основних особливостей ресурсу є використання стилізованих навчальних тем, які охоплюють граматику, лексику, аудіювання, читання тощо. Кожна тема містить чітко структуровані пояснення та супроводжується інтерактивними тестами, що дозволяє закріплювати вивчений матеріал одразу після ознайомлення. Test-English постійно оновлюється новими вправами та поясненнями, тому користувачі цього веб-сайту завжди матимуть змогу вивчити щось нове.

Рис. 1.4. Сторінка із навчальними матеріалами за темою Grammar

Однак, незважаючи на загальну якість ресурсу, сайт має й певні мінуси. Зокрема, деякий функціонал, як авторизація користувача та система платної версії, ще не реалізовані, попри те, що проект функціонує вже тривалий час. Через відсутність платного контенту, на сайті доволі часто з'являється реклама, яка може відволікати від навчального процесу та погіршувати загальний користувацький досвід[4].

Однією з найвідоміших та найпопулярніших освітніх платформ для вивчення іноземних мов є Duolingo – безкоштовний сервіс, який поєднує навчання з грою. Сайт був випущений у 2011 році американськими розробниками Луїсом фон Аном та Североном Хакером. Основна мета платформи – зробити вивчення мов доступних для всіх, незалежно від рівня доходу або місця проживання.

Doulingo підтримує десятки мов і надає можливість вибрати як мову для навчання, так і мову свого інтерфейсу, що робить платформу зручною для людей з усього світу. Користувачі можуть проходити завдання без обов'язкової реєстрації, хоча реєстрація відкриває доступ до більш широкого функціоналу, включаючи збереження прогресу та персоналізацію навчального процесу.

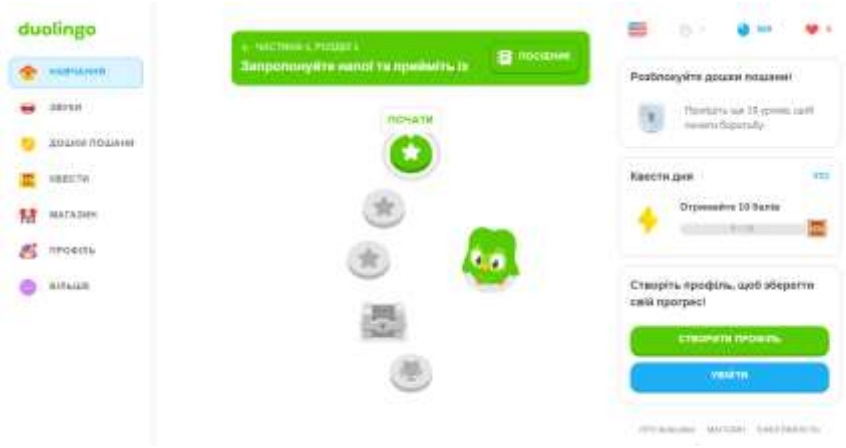


Рис 1.5. Інтерфейс веб-сайту Duolingo

Замість традиційних уроків користувачі виконують міні-завдання, отримують бали досвіду, проходять рівні, збирають віртуальну валюту(самоцвіти), за яку можна придбати різні бонуси. Вбудована система квестів, щоденних завдань і досягнень стимулює регулярне навчання. Крім того, платформа має систему змагання між користувачами, що додає елемент соціальної взаємодії та мотивації. У кінці кожного уроку користувач отримує можливість повторити помилки, які він зробив раніше. Особливість тестів в тому, що вони мають завдання із звуковим супроводом, що сприяє розвитку навичок аудіювання. Приклад такого завдання зображено на рисунку 1.6.

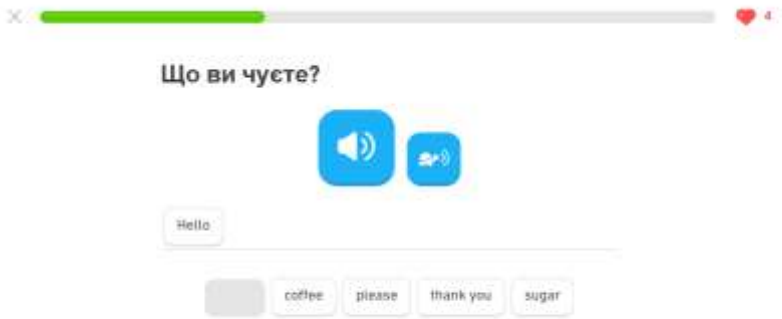


Рис 1.6. Завдання із звуковим супроводом

Сайт має величезну аудиторію і волонтерів, які допомагають у створенні та локалізації курсів. Крім того, існує версія Duolingo for Schools, яка дозволяє вчителям відслідковувати прогрес своїх учнів у реальному часі[5].

Аналіз освітніх платформ показав, що кожна з них має свої сильні сторони та обмеження. Duolingo відрізняється високим рівнем залученості користувача в гру та широкою аудиторією, Test-English пропонує якісні граматичні тести й пояснення у зрозумілому форматі, а Perfect English Grammar – простий у використанні інтерфейс і фокус на практичних завданнях. Завдяки цьому аналізу, ці аспекти будуть враховані під час розробки власного інтерактивного ресурсу для вивчення англійської мови.

1.3. Постановка задачі та визначення вимог до веб-сайту

Для розробки веб-сайту було вибрано методологію Waterfall, яка передбачає послідовне виконання етапів проекту: від планування до реалізації та тестування. Для невеликих проектів це є гарним вибором, оскільки ми уникнемо плутанини та можемо забезпечити контроль за виконанням кожного із етапів. Завдяки послідовності процесу можна ефективно розподілити час та ресурси, мінімізувати ризики, пов’язані з повторним поверненням до попередніх етапів, та зосередитися на якісній реалізації кожної частини проекту. Також для виконання мети кваліфікаційної роботи поставлено вирішити наступні задачі:

- Розробити макет веб-сайту в середовищі Figma;
- Реалізувати верстку сайту згідно з макетом із використанням HTML та CSS;
- Забезпечити адаптивність інтерфейсу для мобільних пристроїв і планшетів.
- Створити функціонал тесту для перевірки рівня володіння англійською мовою;
- Реалізувати основні навчальні тести (наприклад, з граматики);
- Здійснити збереження результатів тестування у базу даних;
- Додати систему нарахування балів за проходження тестів;
- Провести тестування всього функціоналу сайту.

РОЗДІЛ 2. РОЗРОБКА ВІЗУАЛЬНОЇ ЧАСТИНИ ВЕБ-САЙТУ

2.1. Моделювання структури та логіки веб-проекту

Для кращого розуміння функціональних можливостей створеного веб-сайту слід розглянути сценарії використання, які ілюструють типову поведінку користувача під час взаємодії з системою. Ці сценарії описують основні дії, що доступні на сайті, починаючи з реєстрації нового користувача й завершуючи переглядом результатів проходження тестів. Кожен сценарій відповідає конкретному випадку застосування, що демонструє логіку роботи системи та роль бази даних у забезпеченні її функціонування. Розглянемо опис основних сценаріїв використання веб-сайту:

1. Перевірка рівня володіння англійською мовою. Один із перших кроків, який може здійснити користувач на сайті, - це пройти тестування з метою визначення поточного рівня знань англійської мови. Перевірка рівня сприяє адаптації навчального процесу під індивідуальні потреби користувача. Після проходження вступного тесту система може запропонувати відповідний навчальний матеріал і рівень складності подальших тестів.

2. Перегляд навчального матеріалу. Користувач має змогу у будь-який момент ознайомитися з теоретичними матеріалами, що охоплюють граматичні правила, приклади речень, поради та пояснення. Доступ до контенту корисний як для підготовки до тестування, так і для самостійного вивчення теми без виконання завдань. Навчальний матеріал сприяє кращому засвоєнню знань і дає можливість повторити необхідну інформацію у зручний час.

3. Реєстрація та авторизація. Щоб зберігати результати тестів, система передбачає процес реєстрації. Новий відвідувач заповнює спеціальну форму, у якій зазначає особисті дані: ім’я, електронну адресу, пароль тощо. Після підтвердження реєстрації ці дані передаються до бази даних, де вони зберігаються для подальшого використання. У результаті формується персональний обліковий запис, у межах якого можна проходити більше тестів, переглядати накопичені бали та відстежувати

особистий прогрес. Після реєстрації вхід до системи здійснюється за допомогою облікових даних. Під час авторизації система звертається до бази даних для перевірки правильності логіну та пароля. У разі успішної автентифікації відкривається доступ до персоналізованої інформації.

4. Вибір теми тестування. Кожен має змогу самостійно обрати граматичну тему, за якою бажає пройти тест. Серед доступних варіантів - Present Simple, Past Simple, Modal Verbs, Conditionals тощо. Після вибору тема передається до логіки тестової системи, яка формує набір запитань відповідного змісту й рівня. Таким чином, ми зосереджуємося на конкретній темі та поступово вдосконалюємо знання у визначеній галузі.

5. Виконання тесту та отримання результату. Після обрання теми відбувається проходження відповідного тесту. У процесі виконання система зберігає відповіді, а після завершення автоматично проводить оцінювання та виводить результат: кількість правильних відповідей, отримані бали, а також може надавати короткий аналіз помилок. Усі результати записуються до бази даних, щоб їх можна було переглянути пізніше, а також для подальшого аналізу динаміки успішності.

6. Перегляд накопичених балів. Зареєстровані учасники мають доступ до окремого розділу веб-сайту, де відображається інформація про досягнення: загальна кількість балів, теми, за якими було пройдено тести, та дата проходження. Дані автоматично отримуються з бази. Наявність такого розділу мотивує продовжувати навчання та забезпечує можливість візуального відстеження особистого прогресу.

7. Робота з базою даних. База даних відіграє ключову роль у функціонуванні системи. Вона забезпечує збереження та обробку великого обсягу інформації:

- дані облікових записів;
- результати тестувань;
- налаштування профілю;
- аналітичні показники для відстеження успішності.

Система постійно звертається до бази для перевірки авторизації, зчитування навчальних матеріалів, відображення результатів тощо.

На діаграмі 2.1 зображено структуру роботи сайту, що ілюструє основні етапи взаємодії між користувачем і веб-додатком.

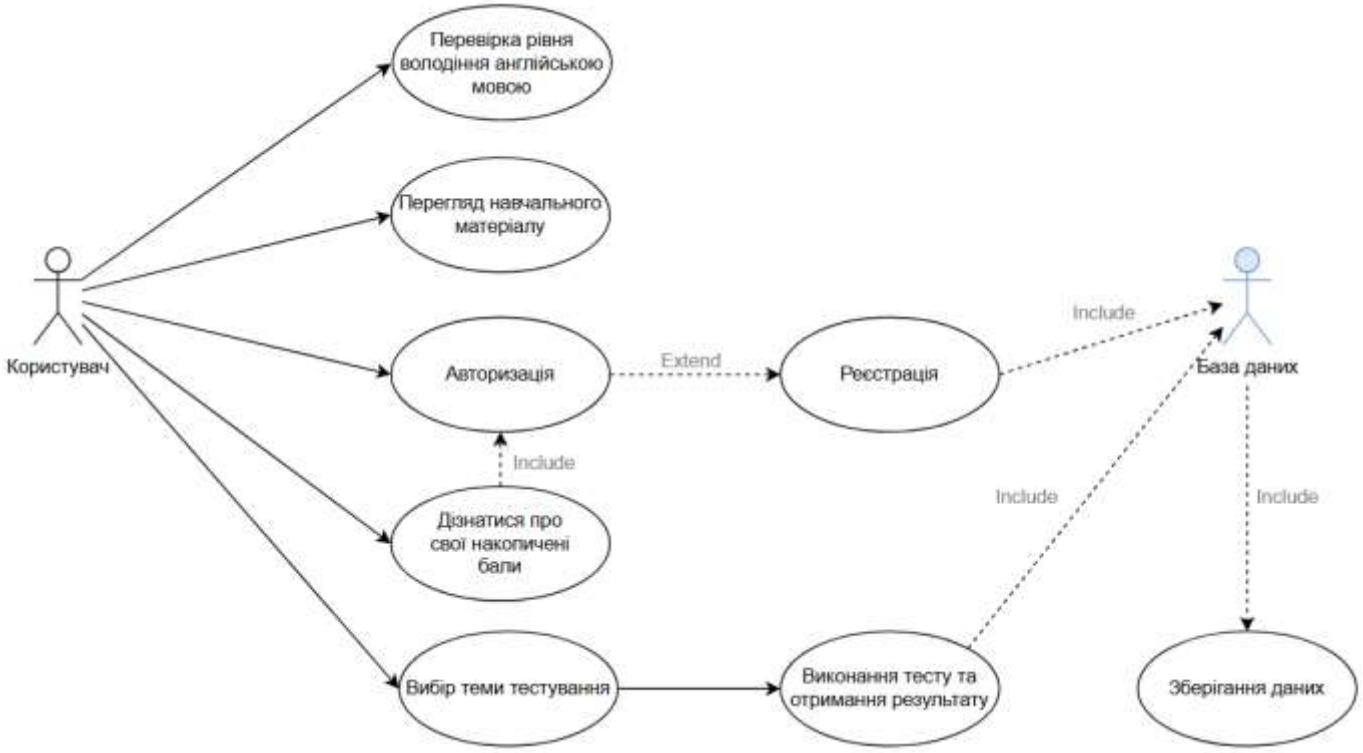


Рис 2.1. Сценарна діаграма взаємодії

Щоб дані зберігалися в базі даних, слід використовувати клієнт-серверну архітектуру, за якої взаємодія між користувачем і сервером відбувається через запити. Модель такої системи полягає в тому, що клієнт відправляє запит на сервер, де він обробляється, і готовий результат відправляється клієнтові. Сервер може обслуговувати кілька клієнтів одночасно. Якщо одночасно приходить більше одного запиту, то вони встановлюються в чергу і виконуються сервером послідовно. Іноді запити можуть мати пріоритети. Запити з більш високими пріоритетами повинні виконуватися раніше. Клієнт - комп'ютер на стороні користувача, який відправляє запит до сервера для надання інформації або виконання певних дій. Сервер - більш потужний комп'ютер або обладнання, призначене для вирішення певних завдань з виконання програмних кодів, виконання сервісних функцій за запитом клієнтів,

надання користувачам доступу до певних ресурсів, зберігання інформації і баз даних. Спосіб взаємодії між користувачем та сервером регламентується спеціальними мережевими протоколами, які встановлюють чіткі правила обміну даними між сторонами. Мережевий протокол - це набір правил, за якими відбувається взаємодія між комп'ютерами в мережі. Одним із таких протоколів є HTTP (і HTTPS) - головний протокол для перегляду веб-сторінок. В свою чергу HTTPS - це захищений варіант з шифруванням даних.

Існують такі концепції побудови системи клієнт-сервер:

1. Слабкий клієнт - потужний сервер. У такій моделі вся обробка інформації перенесена на сервер, а у клієнта права доступу суворо обмежені. Сервер відправляє відповідь, яка не вимагає додаткової обробки. Клієнт взаємодіє з користувачем: складає та відправляє запит, приймає результат і виводить інформацію на екран.
2. Сильний клієнт - концепція, в якій частина обробки інформації надається клієнтові. У такому випадку сервер виступає сховищем даних, а вся робота по обробці та подання інформації переноситься на комп'ютер клієнта.

Також прийнято розділяти клієнт-серверну архітектуру на дворівневу, трирівневу та багаторівневу архітектуру.

Дворівнева архітектура складається з двох вузлів:

- сервер, який відповідає за отримання запитів і відправку відповідей клієнту, використовуючи при цьому лише власні ресурси;
- клієнт, який представляє користувацький інтерфейс. Принцип роботи полягає в тому, що сервер отримує запит, обробляє його і відповідає безпосередньо, без використання сторонніх ресурсів.

Принцип роботи полягає в тому, що сервер отримує запит, обробляє його і відповідає безпосередньо, без використання сторонніх ресурсів.

Трирівнева архітектура складається з наступних компонентів:

- представлення даних - призначений для користувача інтерфейс;
- прикладний компонент - сервер додатків;
- керування ресурсами - сервер бази даних, який надає інформацію.

Принцип роботи полягає в тому, що декілька серверів обробляють запит клієнта. Розподіл операцій знижує навантаження на сервер.

Трирівневу архітектуру можна розширити до багаторівневої (N-tier, Multi-tier) способом встановлення додаткових серверів. Багаторівнева архітектура дозволяє підвищити ефективність роботи інформаційної системи, а також оптимізувати розподіл її програмно-апаратних ресурсів[6].

Для нашого веб-проекту гарним вибором буде дворівнева архітектура зі зв’язком клієнт - сервер - база даних, оскільки це оптимальне рішення з огляду на масштаби та функціональність системи. Ми забезпечуємо чітке розділення відповідальностей: клієнтська частина відповідає за інтерфейс та взаємодію з користувачем, серверна - за обробку логіки, а база даних - за зберігання інформації. Представлення такої архітектури зображено на діаграмі 2.2.

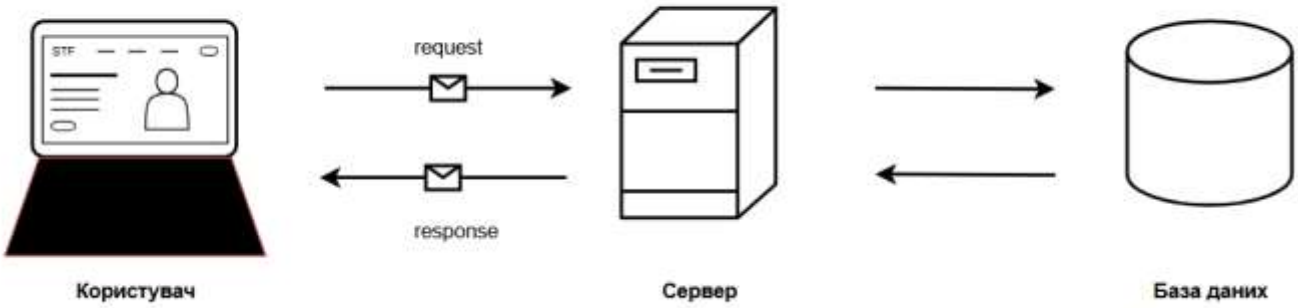


Рис 2.2. Діаграма дворівневої архітектури

2.2. Створення макету в середовищі Figma

Одним із найважливіших етапів розробки веб-сайту є створення макету, оскільки саме на цьому етапі формується загальна структура, логіка розміщення елементів та візуальний стиль майбутнього інтерфейсу. Макет дозволяє ще до етапу

верстки оцінити якість сайту, продумати навігацію, компонування блоків та взаємодію користувача з елементами сторінки.

Для реалізації макету даного проекту було обрано інструмент Figma. Онлайн редактор Figma дозволяє працювати віддалено на самому сайті, тому його не обов’язково завантажувати на пристрій. Figma - це векторний графічний редактор, в якому можна створювати макети, елементи інтерфейсу, зображення, ілюстрації, прототипування, проектування та багато інших креативів та полотен.

Щоб почати працювати в цьому середовищі, необхідно зареєструватися на сайті Figma через любий браузер. Смартфон в такому випадку не підходить, в основну через цей прилад можливе лише переглядати вже зроблений дизайн. Знаходимо та натискаємо на кнопку Sign Up (зареєструватися) на головній сторінці у правому верхньому куті, щоб з’явилася форма для реєстрації. На цьому кроці заповнюємо форму, вказавши свою пошту та пароль. Після цього, на пошту має прийти повідомлення для верифікації свого облікового запису. Залишається лише відповісти на кілька базових запитань про користувача та вибрати безкоштовний план. Тепер ми можемо створити свій перший файл, натиснувши на кнопку "Create" та вибравши у випадаючому меню "Design file"[7].

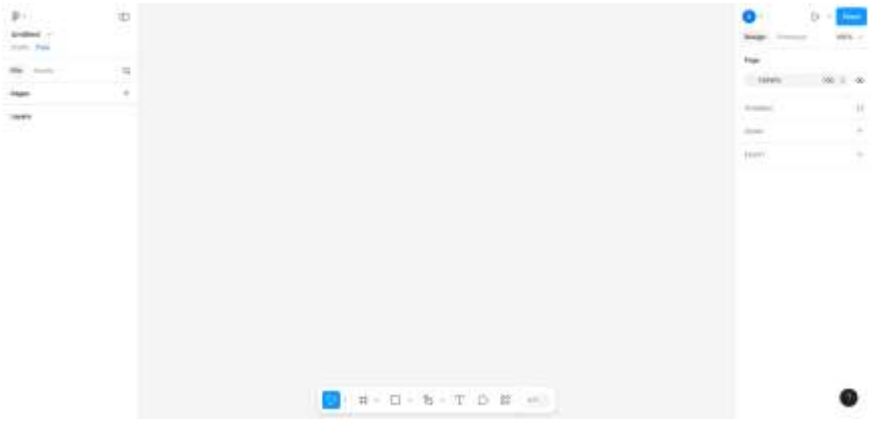


Рис. 2.3. Пусте робоче полотно в Figma

Перед створенням макету, необхідно спроектувати для себе кожну секцію на своїй сторінці, які в майбутньому будуть відображатися на головній сторінці сайту.

Проаналізувавши основні аналоги платформ для самонавчання з англійської мови, було визначено наступні елементи головного інтерфейсу:

- Шапинка з логотипом сайту, навігаційне меню (Explanation, About Us, Recommendations, Log in), яке дозволяє швидко переходити до основних розділів ресурсу.
- Головна секція макету поділена на дві частини. Ліва частина містить коротке мотиваційне звернення до користувача з основним заголовком «Learn easily, speak confidently!», коротким описом і кнопкою Start test, яка спрямовує до тестування рівня знань. Права частина представлена у вигляді великої фотографії, що візуально підсилює повідомлення заголовка.
- Блок з категоріями тестів, представлений у вигляді вертикального переліку карток: Grammar, Vocabulary, Reading, Phrasal verbs, Writing, Listening. Кожна картка містить короткий опис типу завдань і окрему кнопку See the task, що передбачає перехід до відповідного тесту.
- Блок мотивації з назвою "Level Up Your English", який повідомляє користувачеві про систему балів та розвиток мовних навичок у процесі проходження тестів.
- Кінцева інформативна частина (або футер) з додатковими посиланнями (Contact, Privacy, Membership, About Us), роком створення сайту та контактною адресою.

Середовище Figma надає користувачам широкий набір інструментів для створення високоякісних прототипів і макетів веб-інтерфейсів. Одним із ключових інструментів є фрейми, які використовуються для побудови окремих екранних елементів або цілих сторінок. У Figma доступні готові шаблони фреймів для мобільних, планшетних і десктопних інтерфейсів, однак наше полотно задаємо за шириною 1536 пікселів, а висота фрейму буде збільшуватися при кожному додаванні нової секції.

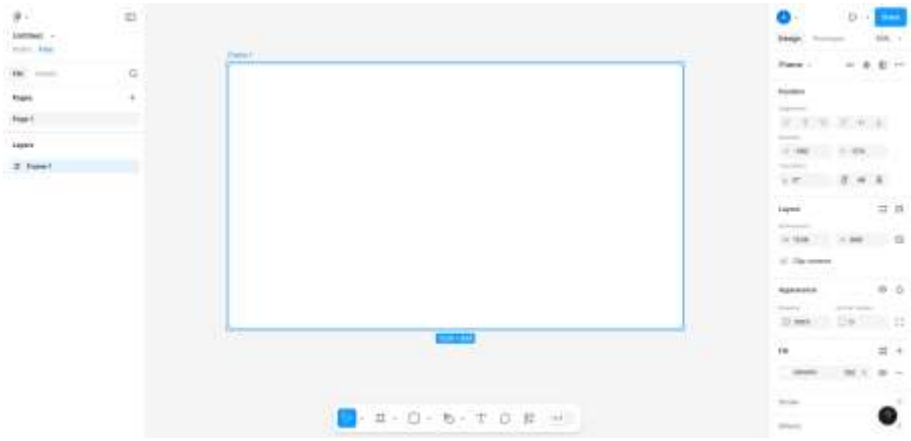


Рис. 2.4. Розміщення фрейму на полотні

Після створення основного фрейму, який слугує фоном усієї сторінки, наступним кроком стало проектування головної секції. Цей блок є центральним елементом на стартовій сторінці сайту та привертає увагу користувача першим. Для побудови фону головної секції було створено прямокутник, розміщений у верхній частині фрейму. Йому було призначено лінійний градієнт, що переходить від світлішого фіолетового кольору у верхньому лівому куті до насиченого відтінку в нижньому правому.

Наступним етапом стало розміщення навігаційного меню у верхній частині секції. Воно складається з кількох елементів:

- Логотип сайту, який було створено як окремий фрейм. У його складі - назва сайту, набрана текстом, та іконка, намальована у векторному редакторі Figma. SVG-графіка була створена вручну за допомогою інструментів побудови кривих, після чого згрупована з текстовим елементом "STF" для формування єдиного логотипу.
- Меню навігації з трьох текстових елементів (Explanations, About Us, Recommendations), які були вирівняні по горизонталі та згруповані для збереження сталої структури. Кожне слово меню має стиль тексту Poppins Bold розміром 16 pt.
- Кнопка Log in, яка була створена як прямокутник із текстовим наповненням. Для неї встановлено фіксовані розміри - 101 піксель у ширину та 33 пікселі у

висоту. Краї кнопки мають закруглення з радіусом 20, що додає візуальної м’якості. Текст усередині кнопки набрано шрифтом Poppins Bold розміром 16 pt як і елементи навігаційного меню.

Усі елементи навігації були об’єднані завдяки автоматичному компонуванню (Auto Layout) - це інструмент, створений на основі принципів HTML-верстки. Розробники прагнули спростити роботу дизайнерів і запропонували рішення, яке дозволяє ефективно розв’язувати складні завдання, пов’язані з побудовою структури інтерфейсу. Цей функціонал значно полегшує створення й упорядкування елементів, блоків або цілих сторінок, а також пришвидшує узгодження технічних деталей між дизайнерами та розробниками. Auto Layout забезпечує гнучкість дизайну, дозволяючи без зусиль адаптувати макет під різні розміри екранів. Під час редагування макета, наприклад при додаванні або видаленні елементів, контейнер автоматично змінює свої розміри, зберігаючи задані відступи. Такий підхід не лише економить час, а й забезпечує узгоджене візуальне представлення без потреби в додатковому ручному налаштуванні. Завдяки використанню автолейауту зменшується ймовірність помилок, оскільки всі параметри розташування задаються лише один раз і не потребують подальшого ретельного контролю для досягнення точного результату. Це особливо важливо при підготовці макетів до передачі в розробку, адже програмісти отримують чітку й логічну організовану структуру елементів. Використання цього інструменту також позитивно впливає на строки виконання проектів, скорочуючи час розробки майже на третину, оскільки багато рутинних дій автоматизуються[8].

Після завершення побудови навігаційного меню сайту, наступним етапом стало створення основної частини головної сторінки, також відомої як hero section. Цей фрагмент макету традиційно є першим, що бачить користувач після завантаження сторінки, тому він повинен бути інформативним, привабливим і логічно побудованим.

Головна секція була поділена на дві частини: ліву та праву, що дозволяє досягти візуального балансу між текстовим контентом та зображенням. У лівій частині

розміщено заголовок і опис, які інформують користувача про призначення сайту. Заголовок має стиль шрифта Raleway Black розміром 48 pt, а опис - Poppins Regular розміром 16 pt. Обидва текстові елементи мають білий колір із шістнадцятковим кодом #FFFFFF, що в системі RGB відповідає значенням (255, 255, 255). HEX (або шістнадцятковий (hexadecimal) код) - це система числення, яка використовує 16 символів для представлення чисел. Включає в себе цифри від 0 до 9 і шість літер від A до F, які відповідають десятковим значенням від 10 до 15. В шістнадцятковій системі числення кожна цифра представляється 4-ма бітами. Шістнадцятковий код широко використовується в програмуванні, особливо для представлення адрес пам'яті, кольорів у веб-дизайні, та інших випадків, де важлива компактність та зручність представлення бітової інформації[9]. Щоб покращити читабельність тексту, для заголовка була встановлена висота рядка 125%, а для опису - 135%, що забезпечує комфортне сприйняття інформації на різних екранах та уникнення візуальної тісноти між рядками.

Під текстовим блоком розташовується кнопка Start test, яка складається з тексту "Start test" та іконки ClipBoardCheck. Ця іконка була імпортована за допомогою плагіна Heroicons, інтегрованого у Figma. Плагіни у Figma - це розширення, які дозволяють значно розширити функціональність редактора. З їхньою допомогою можна імпортувати іконки, генерувати контент, створювати шаблони, автоматизувати рутинні завдання. У випадку з Heroicons, дизайнер отримує набір стильних SVG-іконок, які легко інтегруються в дизайн. Текст кнопки має темно-коричневий колір з кодом #382E2E для контрасту на фоні основної секції. Самій кнопці було призначено лінійний градієнт із золотистими відтінками, що надає елементу акцентованого вигляду та візуально відокремлює його від решти контенту. Усі елементи кнопки були згруповані за допомогою Auto Layout, котре забезпечує правильне вирівнювання тексту та іконки всередині компонента та зберігає задані відступи при зміні розмірів чи структури. Елементи лівої частини були розміщені на достатній відстані один від одного, що покращує читабельність.

У правій частині головної секції розміщено зображення, яке доповнює текстовий блок і посилює загальне враження від дизайну сайту. Для цього використано ілюстрацію учениці з прозорим фоном. Перед додаванням до макету, ілюстрацію було попередньо оброблено в графічному редакторі з використанням прийому маскування. Цей процес полягає у візуальному поєднанні зображення з фігурою (у нашому випадку - з білим колом), що дозволяє автоматично обрізати зображення відповідно до форми і створити єдиний композиційний об'єкт. Після завершення редагування, ілюстрацію було збережено у відповідному форматі, завантажено в середовище Figma та розміщено праворуч у межах головної секції. Щоб зберегти гармонійні пропорції та відповідність до загального дизайну, для зображення встановлено такі розміри: ширина становить 456,94 пікселя, а висота - 499 пікселів. Таке розміщення забезпечує візуальний баланс між лівою і правою частинами екрану, що особливо важливо для створення позитивного першого враження у відвідувача веб-ресурсу. На даний момент, макет головної секції зображено на рисунку 2.5.

Рис. 2.5. Головна секція (hero section) в середовищі Figma

Наступним етапом стало створення другої секції головної сторінки, яка слугує основним навігаційним блоком для переходу до тематичних тестів. Її структура передбачає заголовок та шість навчальних карток, кожна з яких відповідає окремому мовному розділу.

У верхній частині секції розміщується заголовок з текстом "Choose a topic and complete the given tests!". Для нього було використано стиль шрифта Raleway Black розміром 40 pt, а сам текст було виведено по центру секції. Відстань між заголовком та наступними елементами секції встановлено рівно 324 пікселі, що забезпечує візуальний поділ блоків і зручне сприйняття вмісту.

Далі створюється перша тематична картка за темою "Grammar", яка слугує прикладом структури для усіх подальших елементів цього типу. Картка складається з графічного зображення, заголовка, короткого опису та кнопки для переходу до відповідного тестового модуля. Зображення для картки було згенеровано за допомогою інструменту штучного інтелекту на платформі SeaArt.Ai, де для генерації вказується текстовий опис, наприклад "Робоче місце учня при денному світлі дня". Також є можливість налаштувати розмір майбутнього зображення та кількість варіантів. Після отримання відповідного зображення, воно було завантажене на комп’ютер і оброблене в графічному редакторі. У процесі редагування, зображення поміщається на полотно разом із додатковими елементами, зокрема фіолетовим прямокутником розміром 600×338 пікселів та текстовим написом "Grammar", розташованим праворуч від зображення.

У середовищі Figma створюється основа картки у вигляді прямокутника розміром 434×456 пікселів із радіусом заокруглення кутів 40 пікселів, що відповідає загальній візуальній стилістиці сайту. Поверх цієї основи додається додатковий прямокутник меншого розміру, який має однакове заокруглення верхніх кутів і розташовується у верхній частині картки. Саме в нього поміщається підготовлене зображення. Нижче, на основному тлі картки, розміщується заголовок теми та короткий опис, виконані у стилі Poppins Regular. Завершальним елементом є кнопка розміром 167×44 пікселі з текстом "See the task". Основний колір кнопки - білий, проте її контур (обрамлення) та тло основи картки мають фіолетове забарвлення, що створює візуальну цілісність елемента. Для впорядкування структури всі елементи картки виділяються одночасно, після чого групуються за допомогою комбінації клавіш Ctrl + G. За аналогічним принципом було створено п’ять наступних

тематичних карток, які відрізняються між собою лише назвою теми, описом і відповідним зображенням.

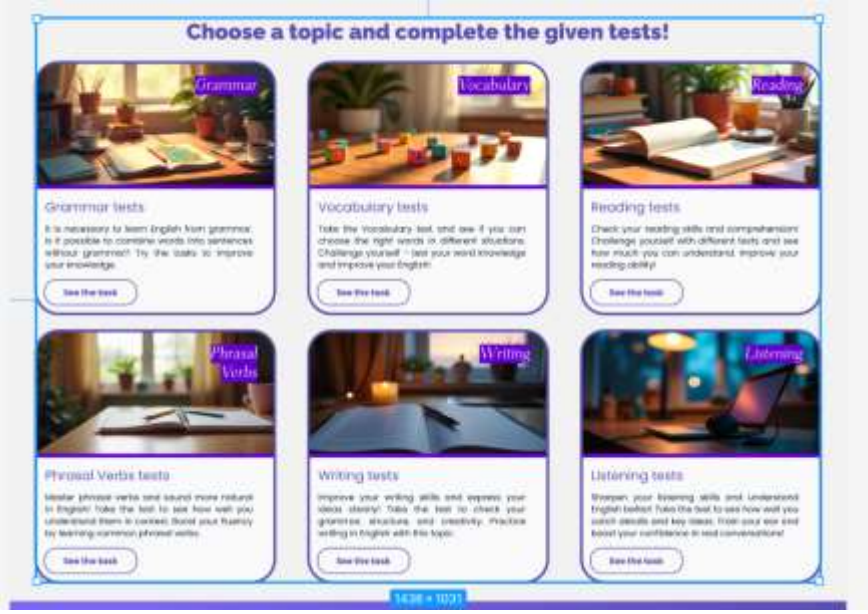


Рис. 2.6. Макет секції із навчальними картками

Після завершення блоку з навчальними картками, наступним етапом стало створення інформативної секції, яка має на меті звернути увагу користувача на можливість отримання додаткової мотивації в процесі навчання. Ця частина макету виконує роль візуальної паузи між функціональними блоками та підсилює інтерес до проходження тестів. У якості фону секції було створено прямокутник розміром 1536×797 пікселів, який заповнено фіолетовим кольором. Щоб уникнути випадкових змін під час подальшої роботи, шар із фоновим елементом було зафіксовано через меню шарів у правій частині інтерфейсу Figma за допомогою натискання на іконку замка. Як і в попередніх секціях, у верхній частині було розміщено заголовок, проте цього разу він виконаний білим кольором для створення контрасту з фіолетовим фоном. Заголовок центрується по горизонталі, зберігаючи візуальну симетрію макету.

Інформативна секція поділена на дві частини: ліву та праву. У лівій частині розміщено блок тексту, який повідомляє користувача про можливість накопичення балів за проходження тестів. Праворуч розташовано ілюстрацію у формі округленого зображення кубка, яка була знайдена у відкритому доступі. Зображення імпортовано у макет і встановлено розміром 534×476 пікселів.

Завершується секція кнопкою, яка в майбутньому матиме свій функціонал. Вона має лінійний градієнт у золотистих відтінках, який має привертати увагу користувача, та містить чорну іконку, взяту з плагіна Heroicons, а також текст "Check points", написаний чорним кольором. Весь вміст кнопки вирівняний і згрупований за допомогою Auto Layout для забезпечення стабільності структури при зміні розмірів або редагуванні. Усі елементи, що складають цю секцію, були об’єднані в окрему групу з назвою "Rating-section".

Рис. 2.7. Макет інформативної секції

Завершальним етапом проектування макету стало створення кінцевої секції сторінки - футера, яка виконує роль інформаційного блоку та містить основні довідкові дані й навігаційні посилання. У якості фону було додано прямокутник розміром 1536×350 пікселів з кольором #091A2F, який відповідає темно-синьому відтінку. Після цього, було створено логотип сайту з використанням векторного редагування жовтого квадрату та текстового напису "STF". Логотип розміщено у верхній частині футера. По центру футера додається заголовок "Made for the qualification work of MSU", під яким вертикально розміщуються чотири текстові елементи: Contact, Privacy, Membership та About Us. Вони виконані в однаковому стилі та згруповані для підтримання логічної структури й рівномірного інтервалу. Між верхньою та нижньою частинами футера розміщено векторну горизонтальну лінію жовтого кольору, яка виконує функцію візуального роздільника. У нижній частині футера розміщено блок із посиланнями на соціальні мережі, представленими білими іконками платформ Facebook, X (Twitter) та YouTube, які були імпортовані за

допомогою плагіна Iconify. Крім того, по центру розміщується текст із електронною поштою, до якого додається іконка поштового листа. Остання секція в нашому макеті має назву "Footer".

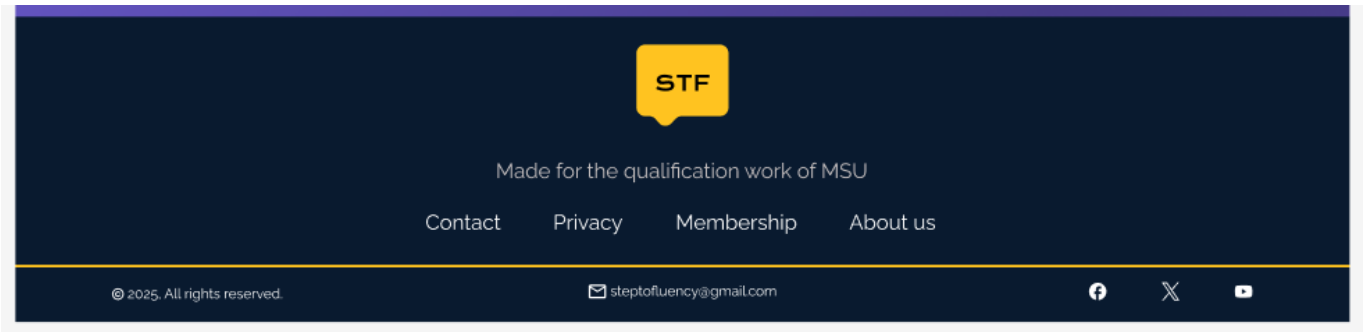


Рис. 2.8. Остання секція в макеті Figma

2.3. Реалізація дизайну та адаптивна верстка

Після завершення створення макету в середовищі Figma, стало можливим перейти до етапу реалізації дизайну безпосередньо у вигляді веб-сторінки. Макет виступає візуальною основою, що визначає розміщення елементів, кольорову гаму, шрифти та інші стилістичні характеристики інтерфейсу. Реалізація відбувається в межах фронтенд-розробки - тієї частини розробки веб-сайту, яка відповідає за зовнішній вигляд і взаємодію користувача з інтерфейсом.

Frontend - це публічна частина веб-додатків, з якою користувач може взаємодіяти і контактувати напряму. У Frontend входить відображення функціональних завдань призначеного для користувача інтерфейсу, що виконуються на стороні клієнта, а також обробка запитів користувачів. По суті, фронтенд - це все те, що бачить користувач при відкритті веб-сторінки. У свою чергу, веб-додаток – клієнт-серверний додаток, в якому клієнтом виступає в основному браузер, а сервером - веб-сервер. Логіка веб-додатку розподілена між сервером і клієнтом, зберігання даних здійснюється переважно на сервері, обмін інформацією відбувається у мережі. Простіше кажучи, це те, що бачить користувач і які дії виконує

кожен раз, коли підключається до мережі інтернет і відкриває будь-який браузер. Frontend розробка - це робота зі створення публічної частини веб-додатку, з якою безпосередньо контактує користувач, і функціоналу, який зазвичай виконується на стороні клієнта. Тобто, фронтенд розробник працює над тим, щоб на сайті кожна кнопка, іконка, текст і вікно не тільки стояли на своєму місці, не перекривали один одного і виглядали цілісно (це веб-верстка), але і щоб вони виконували своє пряме призначення - підштовхували до якоїсь дії (наприклад, щоб кнопка "купити" відкривала кошик, а "play" - запускала відтворення фільму або музики). Компонентами фронтенд розробки являються:

- HTML (HyperText Markup Language) кажучи простими словами - це мова розмітки всіх елементів і документів на сторінці, і їх взаємодія в структурі сторінки.
- CSS (Cascading Style Sheets) - це мова характеристики і стилізації зовнішнього вигляду документа. За допомогою CSS-коду браузер розуміє, як саме необхідно відображати елементи. CSS створює шрифти, кольори, визначає розташування блоків сайту, та інше. Також адаптує один і той же документ в різних стилях, виводить передачу на екран або для читання голосом.
- JavaScript - мова, створена оживляти веб-сторінки. Завдання JavaScript - відгукуватися на дії користувача, обробляти натискання клавіш, переміщення курсора, кліки мишкою. JavaScript також дає можливість вводити повідомлення, посилати запити на сервер, а також завантажує дані без перезавантаження сторінки, і так далі[10].

Для реалізації дизайну веб-сайту гарною ідеєю буде використати спеціалізоване середовище розробки - Visual Studio Code. Хоча верстку можливо здійснювати навіть у звичайних текстових редакторах, наприклад, за допомогою файлів формату .txt, використання професійних інструментів значно підвищує зручність, швидкість та ефективність роботи. Visual Studio Code, який також зазвичай називають VS Code - це редактор початкового коду, створений Microsoft із Electron Framework для Windows, Linux і macOS. Функції включають підтримку налагодження, підсвічування

синтаксису, інтелектуальне завершення коду, фрагменти, рефакторинг коду та вбудований Git. Користувачі можуть змінювати тему, комбінації клавіш, параметри та встановлювати розширення, які додають функціональність. Код Visual Studio можна розширити за допомогою розширень, доступних через центральне сховище. Це включає доповнення до редактора та підтримку мови. Примітною функцією є можливість створювати розширення, які додають підтримку нових мов, тем, налагоджувачів, налагоджувачів подорожей у часі, виконують статичний аналіз коду та додають лінтери коду за допомогою протоколу Language Server. Керування джерелом є вбудованою функцією Visual Studio Code. Він має спеціальну вкладку всередині панелі меню, де користувачі можуть отримати доступ до налаштувань керування версіями та переглянути зміни, внесені до поточного проєкту. Щоб використовувати цю функцію, Visual Studio Code має бути зв’язано з будь-якою підтримуваною системою керування версіями (Git, Apache Subversion, Perforce тощо). Це дозволяє користувачам створювати репозиторії, а також робити запити push і pull безпосередньо з програми Visual Studio Code[11].

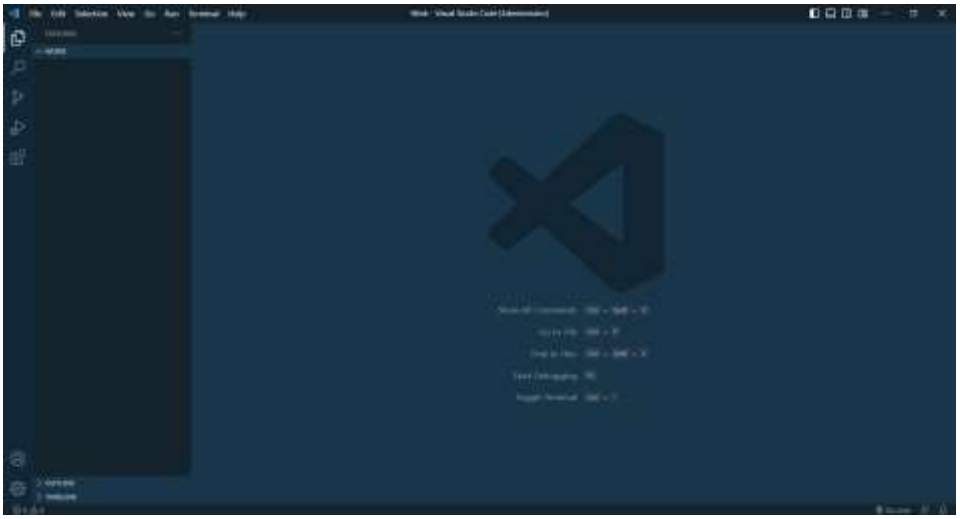


Рис 2.9. Початок роботи в VS Code

Однією з важливих переваг середовища Visual Studio Code є можливість розширення функціональності за допомогою плагінів (extensions). Вони дозволяють адаптувати редактор під конкретні потреби розробника, спрощують рутинні процеси та підвищують ефективність роботи з кодом. У межах реалізації верстки веб-сайту

було використано кілька ключових розширень, які стали особливо корисними на практиці.

- Prettier - Code formatter - це розширення, яке автоматично форматувало HTML, CSS та JavaScript-код згідно з обраними правилами стилю. Його використання дозволяє підтримувати єдину структуру коду в проекті, зменшує ймовірність помилок через неправильне форматування та підвищує загальну читабельність коду. Завдяки Prettier усі відступи, лапки, дужки та інші синтаксичні елементи автоматично вирівнюються відповідно до заданих параметрів.
- Live Server - розширення, яке забезпечує можливість миттєвого перегляду результатів змін у браузері без потреби постійного оновлення сторінки вручну. Після збереження змін у коді, розширення автоматично перезавантажує сторінку, дозволяючи в реальному часі спостерігати за тим, як виглядає сайт.
- Cobalt2 Theme Official - це тема оформлення для Visual Studio Code, яка змінює кольорову палітру інтерфейсу редактора. Не являється дуже важливим розширенням, але візуальне бачення написаного коду стає кращим.

Початковим етапом реалізації веб-сайту є створення двох основних файлів: index.html та style.css. Файл index.html слугує основним HTML-документом, у якому зберігається структура головної сторінки сайту. Саме в ньому розміщуються всі теги, що визначають вміст сторінки - текст, зображення, кнопки, посилання, секції тощо. Файл style.css виконує функцію зовнішнього стилю, який відповідає за візуальне оформлення елементів, таких як кольори, шрифти, відступи, тіні, розміри блоків тощо. Без використання CSS-схеми сайт залишався б суто структурним, примітивним і маловиразним, що значно ускладнювало б взаємодію користувача з інтерфейсом.

У файлі index.html створюється базова HTML-структура, яка включає обов’язкові елементи будь-якого HTML-документа. Цю структуру можна записати вручну, але в середовищі Visual Studio Code передбачено можливість автоматичного створення шаблону за допомогою клавіші ! із подальшим натисканням Tab. Після цього у файл автоматично додаються основні елементи:

- <!DOCTYPE html> - декларація типу документа, що вказує браузеру на версію HTML.
- <html> - кореневий тег, який охоплює весь вміст HTML-документа.
- <head> - секція, яка містить службову інформацію: метадані, підключення стилів, скриптів, шрифтів, а також заголовок сторінки.
- <title> - тег, що визначає назву сторінки, яка відображається у вкладці браузера.
- <body> - основна частина HTML-документа, у якій розміщується контент, доступний для перегляду користувачем: текстові блоки, кнопки, секції, зображення, таблиці тощо.

Рис. 2.10. Базовий вміст файлу index.html

Усі HTML теги можна розділити на дві основні категорії: блочні та вбудовані. Блочні теги використовуються для визначення блокових елементів на веб-сторінці, таких як абзац, заголовки та інші елементи, які займають цілі рядки. Вбудовані теги використовуються для визначення елементів, які розміщені в інших елементах, таких як зображення, посилання та інші. Щоб використовувати HTML теги, потрібно включити їх в код веб-сторінки, обрамивши їх знаками "<" та ">". Деякі теги можуть мати атрибути, які допомагають визначати додаткові властивості для елементів. Наприклад, атрибут "src" для тегу <img>, який вказує шлях до зображення, або атрибут "href" для тегу <a>, який вказує URL-адресу посилання[12].

Щоб веб-сторінка відповідала розробленому макету, важливо не лише зберегти структурну побудову елементів, але й використати ті самі шрифтові стилі, які були застосовані у Figma. Оскільки браузери не завжди мають у своєму розпорядженні

потрібні шрифти за замовчуванням, необхідно попередньо додати ці шрифти до проекту вручну. Для цього використовується офіційний сервіс Google Fonts, який надає широкий вибір безкоштовних веб-шрифтів із підтримкою підключення до HTML-документів. Щоб додати потрібний шрифт, користувач заходить на сайт Google Fonts і вводить назву шрифта у пошуковий рядок. Наприклад, для нашого проекту використовуються Poppins та Raleway. Після введення назви у пошук система пропонує список варіантів. Потрібно обрати відповідний стиль (наприклад, Regular, Bold, Black тощо), після чого натиснути на кнопку "Get font", яка додає обраний стиль до загального списку шрифтів, що будуть використовуватися на сайті. Таку саму дію слід повторити для шрифту Raleway. Після завершення вибору всіх потрібних стилів шрифтів, у нижній частині сайту стає доступною кнопка "Get embed code". Натискання на неї відкриває код, який містить HTML-тег <link> із посиланням на обрані шрифти. Цей тег слід скопіювати та вставити у секцію <head> HTML-документа, що дозволяє браузеру завантажити шрифт із зовнішнього ресурсу та застосувати його до елементів сторінки. Завдяки цьому, усі заголовки, абзаци й інші текстові блоки набувають того самого вигляду, що був заданий у дизайні макету.

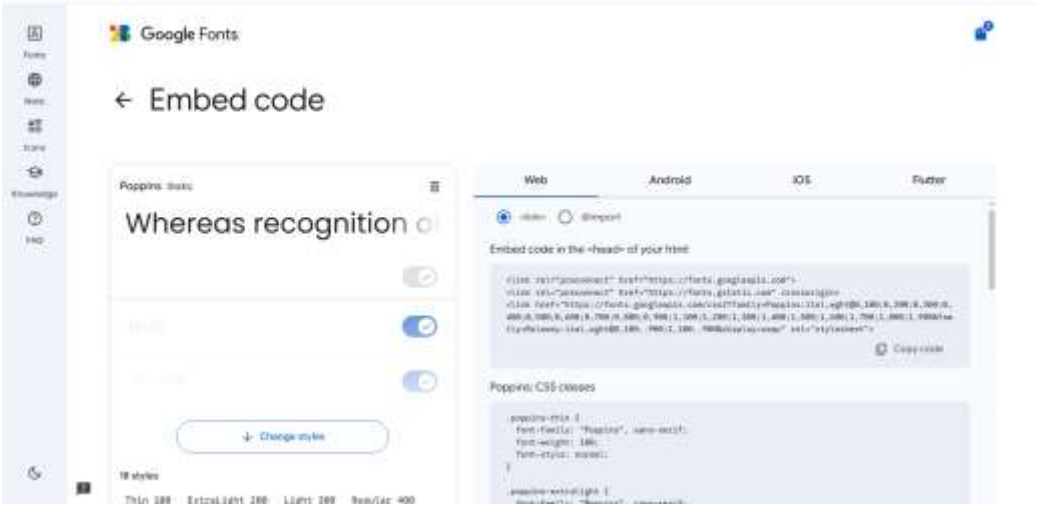


Рис. 2.11. Отримання посилання на шрифт Poppins та Raleway

Після підключення основних ресурсів, зокрема HTML-файлу та стилів, можна перейти до створення структурної верстки сторінки в межах тега <body>. На цьому етапі реалізується початковий каркас сайту - без стилізації, але з логічним розміщенням усіх ключових елементів. Починається верстка з секції <header>, яка

традиційно містить навігаційне меню та, у даному випадку, головну частину першого екрану. Для організації контенту всередині тега <header> створено контейнер div з класом wrapper. У веб-розробці класи використовуються для позначення та групування елементів, до яких у майбутньому застосовуватимуться CSS-стилі або JavaScript-функції. Завдяки класам забезпечується гнучкість у стилізації та структуризації HTML-коду, а також спрощується повторне використання стилів для схожих елементів. Клас wrapper використовується як центральний контейнер, що обмежує ширину контенту, вирівнює його та задає основну сітку сторінки. Усередині контейнера wrapper розміщується тег <nav>, який відповідає за навігаційне меню - один з найважливіших елементів будь-якого сайту. У ньому розташоване посилання на головну сторінку з класом logo, яке містить іконку логотипу (елемент <img>) та назву сайту у вигляді текстового елементу з класом logo-text. Під логотипом розміщується список навігаційних посилань, оформлений за допомогою тегів <ul> (ненумерований список) та <li> (елементи списку). Кожен пункт містить тег <a> (гіперпосилання) з класом underline для подальшої стилізації підкресленням. Останній пункт меню - посилання з класом btn-nav, яке оформлюється як кнопка входу (Log in). Така структура дозволяє реалізувати класичну горизонтальну навігацію.

Паралельно створено адаптивну мобільну версію меню, що реалізується другим елементом <nav> із класом mobile-nav. У ньому повторно використано логотип та додано іконку мобільного меню, представлено у вигляді зображення з класом mobile-menu. Відкривання та закривання мобільної навігації забезпечується окремим контейнером з класом mobile-nav-container, який містить іконку закриття меню (close-icon) та список навігаційних елементів, аналогічних до десктопної версії. Весь функціонал навігаційного меню для мобільних пристроїв буде зроблено пізніше. Після завершення побудови навігації, в межах тієї ж секції <header>, створюється блок , що формує першу візуальну секцію сторінки. Вона поділена на дві частини: ліва частина (left-side) містить заголовок (<h1>), описовий текст (<p>) і кнопку для початку тесту (Start

test), а права частина (hero-photo) - ілюстрацію (фотографію), розміщену за допомогою тега <img>. В атрибут тегу img, а саме src, слід написати шлях до нашого зображення на нашому пристрої. Структура нашої головної секції в VS Code зображено на рисунку 2.12.

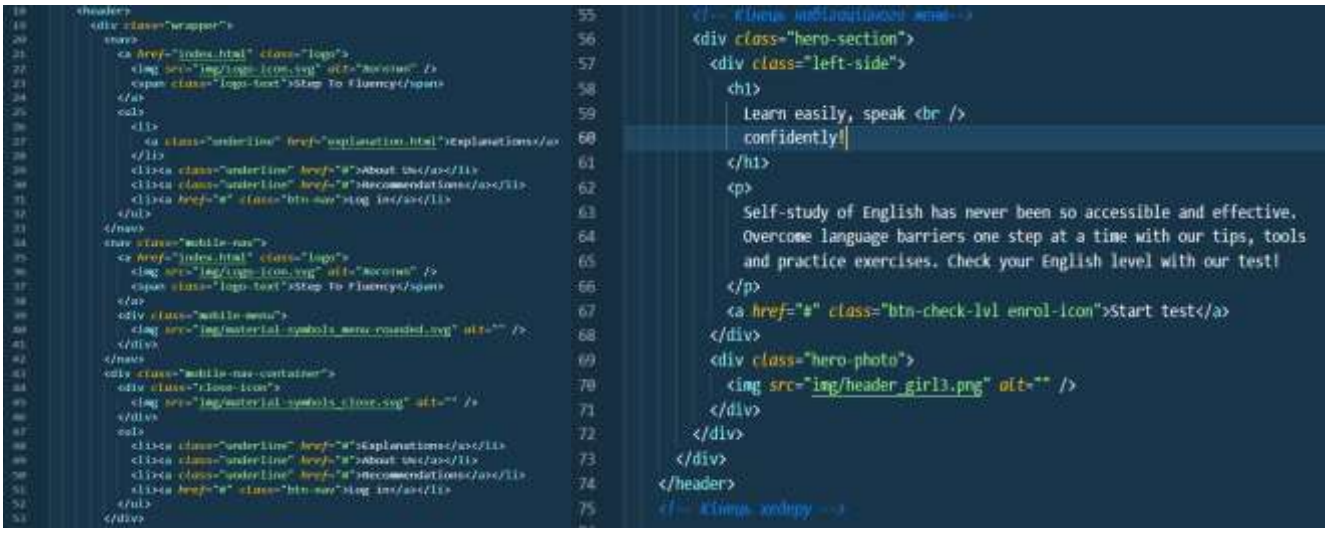


Рис. 2.12. Верстка першої секції: а) навігаційне меню; б) hero section

Наступним функціональним блоком веб-сторінки є секція навчальних завдань за темами, яка реалізована за допомогою тега <section> з класом section-container. Вона має на меті надати користувачеві можливість швидко обрати мовну категорію для проходження тестів. У верхній частині секції розміщується заголовок другого рівня <h2> з класом section-title і додатковим класом violet-color, що буде відповідати за фіолетовий колір саме для цього заголовку. Безпосередньо під заголовком знаходиться контейнер з класом cards-container, який містить набір карток - кожна з них відповідає окремій темі тестування. Для прикладу, розглянемо структуру першої картки з темою Grammar. Картка оформлена у вигляді блоку з класом card. У верхній частині блоку розміщується зображення (<img>), що виконує роль візуального представлення теми. У атрибуті src вказано шлях до зображення, яке вже було використовано в макеті Figma. Під зображенням розміщується внутрішній контейнер card-body, в якому міститься основний текстовий вміст картки. Заголовок теми реалізований за допомогою тега <h4> з класом card-title, а пояснювальний опис - тегом <p> з класом card-text. Завершальним елементом картки є кнопка у вигляді

гіперпосилання <a> з класом card-btn. Кожна картка секції побудована за аналогічною структурою.

Секція з класом information-section реалізує окремий блок мотиваційного контенту. Її структура починається з заголовка другого рівня <h2> з класами section-title, white-color та margin-t, які будуть відповідати за зовнішній вигляд заголовка згідно з кольоровою палітрою сайту та відступами. Основний вміст секції розміщується у контейнері , який складається з двох частин. У блоці міститься текст, оформлений за допомогою тега <p>, включаючи внутрішній
 для поділу абзацу. У другій частині, , розміщується зображення у форматі SVG через тег <img>. Після контенту створюється додатковий блок , у якому розташовується гіперпосилання-кнопка <a> з класами btn-check-points і star-icon. Така структура дозволяє зручно стилізувати як саму кнопку, так і можливу іконку, що входить до її складу. Усі елементи, як і в інших секціях, оформлені за допомогою класів, що спрощує подальше CSS-стилізування та адаптацію.

Завершується HTML-документ секцією <footer>, яка містить усі необхідні елементи для реалізації нижньої частини веб-сторінки. Уся структура футера поділена на три основні блоки. Перший блок розміщується всередині контейнера . У ньому знаходиться логотип сайту, представлений у вигляді зображення з атрибутом alt="STF Logo" і класом logo. Нижче розташовано параграф з коротким підписом щодо призначення сайту. Після цього йде навігаційний елемент <nav class="footer-nav">, який містить чотири посилання: Contact, Privacy, Membership та About us. Кожне посилання реалізовано за допомогою тега <a>, і їх розташування дозволяє зручно формувати горизонтальне або вертикальне меню залежно від застосованої стилізації. Другий блок футера представлений . Цей контейнер виконує функцію візуального розділювача між основною частиною футера та нижньою інформаційною панеллю. Третій блок розміщується у контейнері . У ньому містяться три структурні частини. Перша - посилання з класом copyright, яке включає зображення з

іконкою авторського права (ph_copyright-bold.svg) і текстовий елемент , що містить відповідний рік і фразу "All rights reserved.". Друга частина - контактна інформація у вигляді посилання з класом email, яке містить іконку електронної пошти та адресу поштової скриньки. Останнім елементом є блок , у якому розміщуються посилання на соціальні мережі: Facebook, Twitter (X) і YouTube. Кожне посилання містить відповідну іконку, представлену як зображення з тегом <img> та атрибутом alt, що забезпечує доступність коли відсутній зв’язок з Інтернетом.

На цьому етапі веб-сторінка має базовий вигляд без кольорового оформлення, шрифтів, відступів чи позиціонування, проте вже містить усі текстові, графічні та навігаційні компоненти. Приклад головної секції зображено на рисунку 2.13.

Рис 2.13. Головна секція веб-сторінки без стилізації

Після створення повної HTML-структури сторінки наступним етапом є стилізація інтерфейсу за допомогою CSS (Cascading Style Sheets). Саме стилі визначають зовнішній вигляд сайту: кольорову палітру, шрифти, розміри елементів, відступи, вирівнювання, а також взаємне розташування блоків на сторінці. Без використання CSS навіть правильно структурована сторінка матиме базовий і маловиразний вигляд, позбавлений сучасного дизайну. У цьому етапі застосовується зовнішній файл стилів style.css, який було підключено до HTML-документа за допомогою тегу <link>.

Усі стилі в CSS записуються у вигляді пари: селектор та блок оголошень. Селектор вказує, до якого HTML-елемента застосовується стилізація, а блок оголошень береться в фігурні дужки {}. Усередині блоку перелічуються властивості та їхні значення, які розділяються двокрапкою, а кожен рядок закінчується крапкою з комою. Наприклад, щоб змінити колір тексту або задати відступи, використовуються такі властивості як color, margin, padding, font-size тощо. Стилі можна задавати як для окремих тегів, так і для класів або ідентифікаторів. Для класів перед назвою додається крапка, а для ідентифікаторів - символ #.

Стилізацію починаємо з основного блоку сторінки - <body>. Спочатку задаємо margin: 0, щоб усунути зовнішні відступи за замовчуванням, які автоматично додають деякі браузери. Це дозволяє уникнути зсувів вмісту та забезпечити повний контроль над розміщенням елементів. Фон сторінки оформлюємо світлим відтінком, використовуючи background-color: #f2f2f2. Для структурування вмісту застосовуємо сіткову модель за допомогою display: grid. Далі задаємо grid-template-rows: auto 1fr, що дозволяє розмістити шапку сайту у верхній частині з автоматичною висотою, а основний вміст - на залишковій висоті сторінки. Для центрального контейнера встановлюємо max-width: 1316px, обмежуючи ширину контенту до заданого значення. Щоб вирівняти цей контейнер по центру, додаємо margin: 0 auto. Завдяки цьому всі внутрішні елементи розміщуються симетрично, незалежно від ширини екрана.

Для кожного наступного класу задаємо свою стилізацію згідно створеному раніше макету. Розглянемо для прикладу стилізацію класів btn-nav, header nav, hero-section, card, information-section та footer-bottom.

Клас btn-nav відповідає за стилізацію кнопки входу в навігаційному меню. Для її оформлення задаємо рамку товщиною 2 пікселі білого кольору (border: 2px solid #fff) та скруглення кутів на 20 пікселів (border-radius: 20px), що створює м’який вигляд. Внутрішній відступ налаштований значенням padding: 4.5px 26.5px, що означає 4.5 пікселя зверху й знизу та 26.5 пікселів з боків. Кнопка відображається як блочний елемент в рядку завдяки display: inline-block. Фон встановлено прозорим

(background: rgba(217, 217, 217, 0)). Для тексту обираємо шрифт Poppins жирного накреслення, зі значенням font-weight: 700, розміром font-size: 16px та білим кольором (color: #fff). Вимикаємо стандартне підкреслення посилання через text-decoration: none, а також задаємо overflow: hidden, щоб приховати анімовані ефекти, що виходять за межі елемента. Щоб забезпечити плавну зміну кольору при наведенні, використовується transition для властивостей color та border-color. Також використовується псевдоелемент ::before для створення ефекту заповнення фону знизу вгору на hover - для цього при наведенні висота (height) змінюється від 0 до 100%.

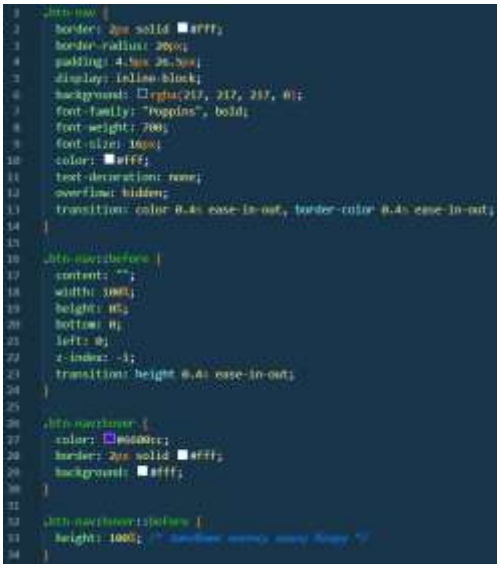


Рис. 2.14. Стилізація класу btn-nav

Клас header nav стилізує навігаційне меню. Йому задаємо display: flex для горизонтального вирівнювання елементів. Властивість justify-content: space-between розміщує елементи по краях контейнера, а align-items: center вирівнює їх по вертикалі. Внутрішній відступ налаштовано через padding: 10px 120px, що забезпечує достатній простір по боках. Меню закріплено у верхній частині сторінки завдяки position: fixed, top: 0 і left: 0, а також розтягується на всю ширину (width: 100%). Встановлюємо z-index: 100, щоб переконатись, що меню буде поверх інших елементів. Фон - прозорий,

але при прокрутці змінюється через клас scrolled. У модифікованому стані (.scrolled) додаємо напівпрозорий фон (background: rgba(9, 26, 47, 0.93)), тінь (box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1)) і ефект розмиття (backdrop-filter: blur(10px) та його підтримка для Safari).

Рис. 2.15. Стилізація класу header nav

Клас hero-section використовується для головної секції з текстом і зображенням. Задаємо display: flex, щоб розташувати два блоки - текстовий і візуальний - у ряд. Встановлюємо верхній відступ margin-top: 60px і внутрішній відступ по горизонталі padding: 0 20px. Вирівнюємо елементи вертикально по центру (align-items: center) і розподіляємо простір між ними (justify-content: space-between). Ліва частина секції має фіксовану ширину в 457 пікселів. Зображення у правій частині має висоту 499 пікселів і зсувається праворуч завдяки margin-right: 100px.

```
171   .hero-section {
172     display: flex;
173     margin-top: 60px;
174     align-items: center;
175     justify-content: space-between;
176     padding: 0 20px;
177   }
```

Рис. 2.16. Стилізація класу hero-section

Клас card відповідає за стилізацію карток із темами. Встановлюємо гнучкість за допомогою flex: 1 1 445px, що дозволяє картці адаптуватися в межах мінімальної ширини. Встановлюємо також фіксовану ширину (width: 445px) та максимальну

ширину (max-width: 445px). Висота - 465 пікселів. Задаємо м’яку тінь через box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1) для візуального відділення від фону. Overflow: hidden приховує контент, що може виходити за межі елементу. Для плавної анімації використовується transition: transform 0.3s ease-in-out, box-shadow 0.3s ease-in-out. Обрамлення - рамка товщиною 5 пікселів із кольором #5a4caf та радіус заокруглення 40 пікселів. Фон картки - світлий (background: #fafafa).

Рис. 2.17. Стилізація класу card

Клас information-section відповідає за інформативну секцію з балами. Для фону використовується градієнт: background: linear-gradient(149deg, #7b68ee 0%, #463b88 100%). Мінімальна висота секції складає 737.6 пікселів (min-height), ширина - 100% (width: 100%). Вирівнювання вмісту здійснюється по центру: text-align: center. Елементи розташовуються вертикально в колонку (flex-direction: column) з центруванням по вертикалі (justify-content: center). Для нижнього відступу додаємо padding-bottom: 33px.

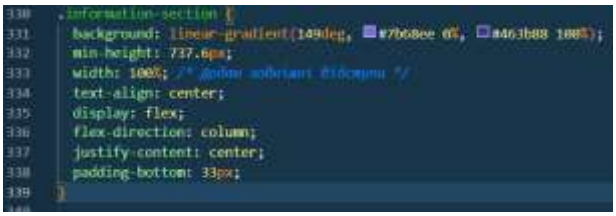


Рис. 2.18. Стилізація класу information-section

Клас footer-bottom відповідає за нижню частину футера. Використовується display: flex, а також justify-content: space-between, що розподіляє елементи зліва направо, та align-items: center для вирівнювання по вертикалі. Встановлюємо внутрішній відступ padding: 20px 100px. Внутрішні елементи, такі як іконки авторських прав і електронної пошти, стилізуються окремо: для зображення

авторських прав задаємо width: 16px і height: 16px, а для іконки пошти - 24 пікселі відповідно. Обидва мають плавну зміну за допомогою transition: filter 0.3s ease. Для текстових блоків з авторськими правами та поштою застосовується display: flex, вертикальне вирівнювання (align-items: center) і проміжок між елементами - gap: 3px.

```css
.footer-bottom {
  display: flex;
  justify-content: space-between;
  align-items: center;
  padding: 20px 100px;
}
```

Рис. 2.19. Стилізація класу footer-bottom

Однією із головних частин розробки веб-сайту є створення адаптивності під різні типи пристроїв, зокрема десктопи, планшети та смартфони. У сучасному веб-просторі користувачі дедалі частіше взаємодіють із сайтом не лише з комп’ютера, а й з мобільних пристроїв, тому важливо забезпечити коректне відображення контенту на будь-якому екрані. Для перевірки дизайну веб-сайтів не обов’язково запускати його через різні пристрої, бо є простіший метод, а саме за допомогою Dev Tools. Dev Tools - це вбудовані функції в браузерах, які надають доступ до інструментів для розробки та налагодження веб-сторінок. Вони дозволяють переглядати HTML, CSS, JavaScript-код, а також аналізувати продуктивність, мережеві запити та інші аспекти роботи сайту. Однією з ключових функцій Dev Tools є режим симуляції різних пристроїв. Приклад дизайну нашого веб-сайту без адаптивності під розширення екрану 1024 на 595 зображено на рисунку 2.20.

Рис. 2.20. Веб-сайт "Step To Fluency " за шириною 1024 без адаптивності

Медіазапити @media (min-width) і @media (max-width) є важливими інструментами для реалізації адаптивного дизайну в CSS. Вони дозволяють застосовувати різні стилі в залежності від розміру екрана користувача, що дає змогу налаштувати вигляд веб-сторінки під різні пристрої (мобільні телефони, планшети, персональні комп’ютери тощо). Ми можемо також написати мінімальний та максимальний діапазон, при якому буде застосовуватися адаптивність за допомогою слова " and ". Для нашого веб-сайту, виберемо такі екранні діапазони: від 0 пікселів до 678 пікселів, від 0 до 768, від 679 до 768, від 769 до 920, від 679 до 1152 та від 1153 пікселів до 1440 пікселів.

Стилі для адаптивного дизайну оформлюються аналогічно до звичайних стилів, що використовуються при розробці веб-сайтів. Для прикладу, розглянемо написання адаптивності для екранного діапазону від 679 пікселів до 1152 пікселів. Для забезпечення коректного відображення сайту на пристроях із середньою шириною екрана використовується медіа-запит @media (min-width: 679px) and (max-width: 1152px). Усередині нього задаються стилі, які змінюють розміщення, розміри, шрифти та відступи елементів для зручного перегляду на планшетах або невеликих ноутбуках. Зменшуємо горизонтальні відступи в навігації (padding: 10px 40px) для економії простору. Основна секція (hero-section) переводиться з горизонтального в вертикальне розміщення блоків, встановлюючи flex-direction: column та text-align:

center. Таким чином, ми адаптували розміщення тексту та зображення одне під одним. Розмір шрифтів збільшується (font-size: 18px), а відступи налаштовуються під нову компоновку. Зображення стає гнучким - height: auto, max-width: 80%, а також вирівнюється по центру автоматичними відступами (margin: 20px auto 20px). Зменшується розмір логотипу (width: 40px) і шрифт текстової частини (font-size: 24px), щоб не перевантажувати верхню частину сторінки. Картки тем також адаптуються - зменшується ширина (max-width: 380px) та висота (height: 420px). Відстань між ними збільшується до gap: 60px для кращого сприйняття на середніх екранах. Зображення всередині карток обмежуються по висоті (height: 200px) і обрізаються по формі контейнера (object-fit: cover). Розміри заголовків, тексту та кнопок також підлаштовуються відповідно (font-size: 22px, 14px, зміна розмірів кнопки). Інформативна секція переходить на вертикальне розміщення (flex-direction: column) з центруванням тексту і зменшеними внутрішніми відступами (padding: 0 20px). Це дозволяє зберегти читабельність і цілісність структури. Також зменшується розмір блоку з ілюстрацією, щоб він органічно вписувався в нове розташування. Футер також адаптується: змінюються внутрішні відступи (padding: 20px 50px) та зменшуються проміжки між іконками соціальних мереж (gap: 20px).

Рис. 2.21. Головна секція після додання адаптивності під екрани невеликих ноутбуків

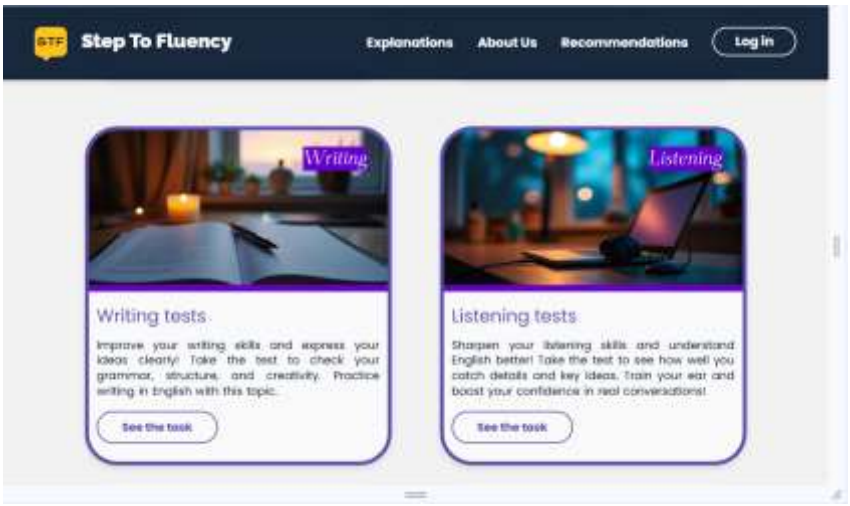


Рис. 2.22. Навчальна секція після додання адаптивності під екрани невеликих ноутбуків

Для різних екранів адаптивність реалізується схожим чином, однак для мобільних пристроїв є один виняток. Раніше у файлі index.html було створено дві версії навігаційного меню - стандартну для великих екранів і мобільну (mobile-nav), що містить іконку відкриття меню та окремий контейнер із посиланнями. Завдяки адаптивності можна контролювати, яке меню буде видно в залежності від ширини екрана. Можливість відкриття такого навігаційного меню реалізується за допомогою мови програмування JavaScript, яку буде докладніше розглянуто у наступному розділі. У межах діапазону ширини екрана від 920 до 4000 пікселів встановлюється значення display: none, щоб мобільне меню не відображалося на пристроях із великим екраном.

Для мобільних пристроїв з шириною екрана до 768 пікселів застосовується окреме меню. За допомогою медіа-запиту приховуємо стандартне навігаційне меню (display: none) і вмикаємо відображення мобільного (display: flex). Основу мобільного меню становить клас mobile-nav, який містить логотип сайту та іконку відкриття меню. Саме розгорнуте меню представлено у вигляді бічної панелі (mobile-nav-container), яка за замовчуванням прихована за межами екрана з правого боку (встановлено right: -300px). При додаванні класу active панель плавно з’являється,

зсуваючись на екран (right: 0) із затримкою, що задається через transition. Усередині цієї панелі знаходиться список пунктів навігації, який оформлено вертикально, із центральним вирівнюванням та відступами між елементами (margin: 30px 0). Для кожного пункту такого меню встановлено збільшений розмір шрифту (font-size: 20px), жирність (font-weight: 700) та світлий колір тексту, що відповідає дизайну сайту. Також передбачено окрему кнопку закриття меню - іконку у верхньому правому куті, яка з'являється лише при активному стані меню. Її видимість та можливість взаємодії змінюються завдяки властивостям opacity та pointer-events, які дозволяють забезпечити плавне зникнення або поява елемента. Відображення мобільного навігаційного меню зображено на рисунку 2.23.

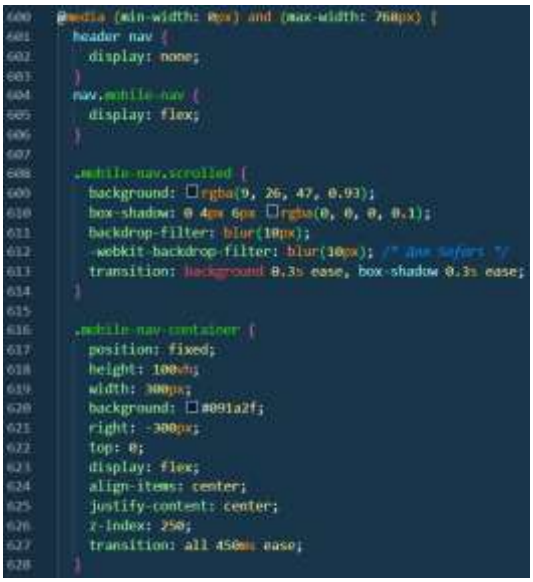


Рис 2.23. Реалізація мобільного меню через медіа-запит

Рис. 2.24. Відображення мобільної навігації за допомогою іконки «бургер»-меню.

Для повноцінного функціонування веб-сайту важливо створити не лише головну сторінку, а й інші сторінки, які розширюють можливості ресурсу та відповідають його функціональному призначенню. Серед них можуть бути сторінки з поясненнями, розділами тестів, рекомендаціями тощо. Перед тим як приступити до верстки цих додаткових сторінок, доцільно створити окремий файл стилів, який буде містити базові стилі, спільні для всіх сторінок сайту. Ми уникнемо дублювання коду, забезпечимо узгодженість зовнішнього вигляду між сторінками та спростимо подальшу підтримку проекту. Назва цього файлу - style_base.css. У нього переносяться ті стилі з основного файлу style.css, які є універсальними та застосовуються на кожній сторінці. Зокрема, це стилі для тега body (фон, відступи, сітка), обгортки wrapper, а також усіх елементів футера - footer, footer-container, foot-center, footer-bottom тощо. Доцільно також включити стилі для заголовків, шрифтів, кнопок та інших елементів, які повторюються у структурі сайту.

Для розширення функціональності нашого веб-сайту слід створити окрему сторінку, яка міститиме пояснення до різних тем англійської мови. Серед основних тем - граматика, лексика, вимова та інше. Ця сторінка дозволяє користувачу обрати потрібну категорію та ознайомитись із відповідним переліком підтем. Для реалізації цієї сторінки створюється окремий HTML-файл з назвою explanation.html. Він структурно нагадує головну сторінку сайту, однак має індивідуальний дизайн із помаранчевою палітрою кольорів, що візуально вирізняє його серед інших. Основною частиною сторінки є список тем, кожна з яких може розгортатися або згортатися за потреби користувача. У структурі HTML-розмітки використовується список <ul>, всередині якого розміщуються пункти <li>, що містять посилання на конкретні сторінки з навчальними матеріалами (наприклад, present-simple.html, phrasal-verbs.html тощо). Кожна категорія тем оформлена у вигляді блоку з класом topic. Цей клас відповідає за зовнішнє оформлення блоку однієї теми, наприклад, "Grammar" або "Vocabulary". Усередині topic міститься назва теми, позначена класом topic_title, який відповідає за відображення тексту теми у стилізованому вигляді. Поруч з текстом розміщується стрілка, яка вказує на можливість розгортання або згортання блоку.

Вона позначена окремим класом topic_arrow і розташовується справа від назви теми. Блок із поясненнями до певної теми має окремий ідентифікатор, що співпадає з назвою теми (наприклад, grammar, vocabulary, listening тощо), і клас content, який визначає оформлення розгорнутої частини зі списком підтем. Усі підпункти в кожній темі оформлені у вигляді списку, що дозволить користувачу легко переміщатись між конкретними сторінками з поясненнями.

```
<div class="topic">
  <span class="topic_title">Vocabulary</span>
  <span class="topic_arrow">&#9660;</span>
</div>
<div id="vocabulary" class="content">
  <ul>
    <li><a href="phrasal-verbs.html">Phrasal verbs</a></li>
    <li><a href="idioms.html">Idioms</a></li>
    <li><a href="synonyms-antonyms.html">Synonyms and antonyms</a></li>
    <li><a href="false-friends.html">False friends</a></li>
  </ul>
</div>
```

Рис 2.25. Верстка списку за темою Vocabulary

Далі створюємо файл style_exp.css, який буде складатися із різних стилів. До цього файлу додаються схожі стилі, як і в основному стилізованому файлі style.css, зокрема стилі для таких структурних елементів, як header, header nav, footer, footer-bottom та footer-nav. Крім того, у style_exp.css також прописуються адаптивні стилі, аналогічні до тих, що використовуються в основному файлі.

Для іконки стрілки вниз, яка вказує на необхідність прокрутки, клас .arrow_down отримує анімацію плавного руху вниз slowDrop, властивість display: block та центрування через margin: 0 auto. Контейнер для всіх тем .topics_container має вертикальні внутрішні відступи padding: 50px 0, що дозволяє розділити вміст від інших блоків сторінки. Блок окремої теми .topic оформлюється у вигляді помаранчевого прямокутника з округленням кутів (border-radius: 30px), внутрішніми відступами (padding: 10px), flex-розміщенням заголовка й стрілки по краях (justify-content: space-between) та зміною кольору фону при наведенні. Це створює ефект натискання на елемент. Блок .content відповідає за прихований список підтем. На початку він не відображається (display: none, max-height: 0), а при взаємодії - плавно розгортається завдяки властивості transition. Усередині ul вилучаються стандартні

маркери списку (list-style: none), а сам список має відступи, які вирівнюють його відносно батьківського елемента. Кожен пункт списку .content li стилізується як кнопка з округленням, помаранчевим фоном, світлим текстом і темною рамкою. При наведенні фоновий колір змінюється на темніший. Посилання в цих пунктах (.content li a) займають всю ширину, мають великий шрифт, жирне накреслення та не мають підкреслення. Назва теми стилізується класом .topic_title, де задається великий розмір шрифту та білий колір. Стрілка праворуч (.topic_arrow) має аналогічний розмір і колір, а також плавну зміну положення при розгортанні завдяки transition. Далі стилізується клас topic span, який додає горизонтальні відступи, щоб забезпечити рівномірність між текстовими та графічними елементами теми. Кінцевий результат оформлення сторінки із поясненнями зображено на рисунку 2.26.

Рис. 2.26. Список навчальних тем на сторінці пояснень

Для прикладу створення навчального блоку на сайті оформлюється пояснення теми Present Simple Tense з категорії Grammar. Для цього створюється окремий HTML-файл, наприклад present-simple.html, у якому розміщується уся необхідна інформація - заголовки, теоретичні пояснення, приклади. Стилі задаємо безпосередньо в самому файлі за допомогою тега <style>, який розміщується в блоці <head>. Між відкриваючим і закриваючим тегом <style>...</style> записуються ті CSS-властивості, які будуть застосовуватись лише до цього конкретного документа. Серед основних елементів, які стилізуються: body, container, h1, h2, p та example.

Також цей файл матиме невелику таблицю із поясненням до правила Present Simple Tense, яка зроблена за допомогою тегів table, th та td. Готова сторінка із поясненням можна побачити на рисунку 2.27.

Рис. 2.27. Форма-пояснення до теми Present Simple Tense

Ми розглянули процес створення візуальної частини веб-сайту, починаючи з розробки макету в середовищі Figma. Далі на основі цього макету було виконано верстку сторінок у середовищі Visual Studio Code з використанням HTML та CSS, із забезпеченням адаптивності під різні типи екранів. Також реалізовано навігаційне меню для мобільних пристроїв та окрема сторінка з поясненнями тем англійської мови. На цьому етапі проект має завершений дизайн без функціональної частини. У наступному розділі буде розглянуто одну з популярних мов програмування для реалізації логіки сайту та динамічної взаємодії з користувачем.

РОЗДІЛ 3. ПОБУДОВА ФУНКЦІОНАЛЬНОЇ ЧАСТИНИ ТА РОБОТА З БАЗОЮ ДАНИХ

3.1. Основи мови JavaScript та її роль у веб-розробці

JavaScript (скорочено JS) є повноцінною, динамічною мовою програмування, що активно застосовується у зв'язці з HTML-документами для надання інтерактивності веб-сторінкам. Ця мова дозволяє додавати на сайти анімації, реагувати на дії користувачів, обробляти події, змінювати вміст сторінки без її перезавантаження та реалізовувати інші елементи поведінки, які перетворюють статичні сторінки на живі веб-застосунки. Автором JavaScript є Брендан Айк - один із співзасновників проєкту Mozilla, Mozilla Foundation та Mozilla Corporation. Саме завдяки його розробці, що з’явилася у 1995 році, мова швидко здобула популярність і стала невіддільною частиною сучасного вебу. Однією з головних переваг JavaScript є його універсальність та доступність для новачків. Навіть з мінімальними знаннями можна створити базові скрипти, що вже надають сайту інтерактивних можливостей. А з часом, опанувавши більш складні концепції, розробник отримує змогу реалізовувати інтерактивні ігри, 2D- і 3D-анімацію, динамічні графічні елементи, а також повноцінні веб-застосунки, які взаємодіють з базами даних, користувачами та іншими сервісами. Незважаючи на свою компактність, JavaScript надзвичайно гнучкий і масштабований. На його основі створено велику кількість додаткових інструментів, які суттєво розширюють можливості мови. Ці інструменти дозволяють вирішувати складні задачі із мінімальними зусиллями з боку розробника, підвищуючи продуктивність розробки та забезпечуючи ефективну побудову складних функціональних рішень.

Після ознайомлення з основними можливостями мови JavaScript варто розглянути її базові конструкції, зокрема змінні, які є одним із ключових елементів у програмуванні. Змінні - це контейнери, всередині яких можна зберігати значення. Ми починаємо з того, що оголошуємо змінну за допомогою ключового слова var або let, за яким слідує будь-яке ім'я, яким ми захочемо її назвати. Окрім змінних, важливим

елементом структури програм у JavaScript є умови та функції. Функції дозволяють нам краще організовувати код і робити його більш зручним для повторного використання. Завдяки функціям складні задачі ми можемо розбити на менші підзадачі, що значно спрощує процес розробки. Функції - це спосіб упаковки певної логіки, яку ми плануємо використовувати неодноразово. Кожного разу, коли нам потрібно виконати якусь дію або послідовність дій, ми просто викликаємо функцію за її ім’ям замість того, щоб знову писати той самий код. Це дозволяє уникати дублювання, спрощує супровід програми та робить її структуру зрозумілішою. У JavaScript ми можемо створювати як власні функції, так і використовувати вбудовані - ті, що вже надає сама мова програмування. Умови - це конструкції в коді, які дозволяють перевірити істинність або помилковість виразу та виконати інший код залежно від отриманого результату. Найпоширеніша форма умови - інструкція if ... else.

Для реалізації справжньої інтерактивності на нашому веб-сайті необхідне використання подій. Події - це структура, яка стежить за тим, що відбувається у браузері, а потім дозволяє нам запускати код у відповідь на це. Коли ми взаємодіємо з веб-сторінкою, браузер фіксує ці дії як події. Це може бути клік мишею, натискання клавіші, прокручування сторінки, наведення курсору, надсилання форми тощо. Кожен із цих типів взаємодії можна відстежувати за допомогою JavaScript і відповідно реагувати. Наприклад, якщо користувач натискає кнопку, скрипт може відкрити нове вікно, змінити стиль елемента, надіслати запит на сервер або показати сповіщення. Щоб працювати з подіями, JavaScript пропонує механізм обробників подій (event handlers). Обробник - це функція, яка "слухає" певну подію на елементі сторінки[13].

На даному етапі розробки веб-застосунку в головному HTML-файлі вже реалізовано два типи навігаційного меню: стандартне для десктопної версії та мобільне з власним випадаючим списком. Основне меню фіксується у верхній частині сторінки за допомогою стилів, прописаних для елемента header nav, що забезпечує його постійну присутність у полі зору користувача незалежно від прокрутки.

Для покращення візуального сприйняття інтерфейсу слід реалізувати додаткову динамічну функцію: при прокручуванні сторінки вниз фоновий колір навігаційного меню змінюватиметься на чорний, а при поверненні до верхньої частини сторінки - зникатиме, повертаючи прозорий вигляд. Для реалізації цього ефекту вже визначено окремий CSS-клас .scrolled, який містить відповідні стилі, тож залишилось лише написати подію, яка додає або видаляє цей клас залежно від положення сторінки.

Щоб почати писати JavaScript-код для цієї функціональності, спершу створюється окремий файл із розширенням .js. Його необхідно підключити до HTML-документа за допомогою тега <script>. Оскільки мова йде саме про поведінку навігаційного меню, файл називається відповідно - navbar.js.

Для реалізації зміни зовнішнього вигляду навігаційного меню під час прокручування сторінки, необхідно створити функцію, яка буде реагувати на подію прокрутки. У JavaScript для цього використовується спеціальний метод addEventListener(), який дозволяє "слухати" певні події та виконувати функцію у відповідь на них. У нашому випадку об'єктом, до якого прикріплюється слухач події, є глобальний об’єкт window, що представляє вікно браузера. Ми додаємо до нього обробник події "scroll" - вона спрацьовує щоразу, коли користувач прокручує сторінку вгору або вниз. Це дає можливість постійно відстежувати положення сторінки відносно верхньої межі.

Всередині функції, яка виконується під час прокрутки, ми використовуємо властивість window.scrollY. Вона повертає кількість пікселів, на яку сторінка була прокручена вертикально. Далі реалізується умовна конструкція: якщо значення scrollY перевищує 50 пікселів, до елемента nav (навігаційного меню) додається CSS-клас "scrolled". Саме цей клас відповідає за зміну стилів - зокрема, за встановлення чорного фону меню, що візуально робить його більш помітним під час перегляду контенту сторінки. У протилежному випадку - якщо користувач повертається до самого верху сторінки, і значення scrollY знову стає менше або дорівнює 50, цей клас навпаки видаляється методом classList.remove().

```
window.addEventListener("scroll", function () {
  if (window.scrollY > 50) {
    nav.classList.add("scrolled");
  } else {
    nav.classList.remove("scrolled");
  }
});
```

Рис. 3.1. Обробка події scroll для зміни вигляду навігації

Щоб покращити зручність користування веб-сайтом на мобільних пристроях, ми розширюємо обробку події прокрутки сторінки, додаючи підтримку мобільного меню. До вже описаної частини, де при прокрутці додається або видаляється клас scrolled у основного меню, додається аналогічна дія і для мобільної навігації. Для цього спершу здійснюється пошук необхідних елементів за допомогою методу document.querySelector(). Зокрема, ми отримуємо доступ до:

- елемента з класом .mobile-nav, що відповідає за мобільне меню зверху;
- кнопки з класом .mobile-menu, яка відкриває повне мобільне меню;
- іконки закриття .close-icon всередині контейнера мобільного меню;
- та самого контейнера .mobile-nav-container, в якому відображається мобільна навігація.

Далі ми доповнюємо вже наявний обробник події scroll, додавши умову, щоб одночасно з основним меню клас scrolled додавався і до мобільного меню. Таким чином, при прокручуванні сторінки більше ніж на 50 пікселів обидва меню змінюють свій вигляд. Якщо ж сторінка прокручена назад догори, клас scrolled знову видаляється, як і на більш широких екранах. Окрім цього, додаються два нових обробника подій:

- при натисканні на кнопку відкриття мобільного меню (.mobile-menu), до контейнера додається клас active, що візуально відкриває меню;
- при натисканні на іконку закриття (.close-icon) цей клас видаляється, і меню зникає з екрану.

На поточному етапі реалізовано повноцінне адаптивне навігаційне меню завдяки невеликій частині коду на мові JS. Для мобільних пристроїв передбачено бічну панель, яка плавно з’являється з правого краю екрана після натискання на іконку меню. Закриття здійснюється натисканням на іконку хрестика, після чого меню так само плавно зникає, повертаючись за межі екрана.

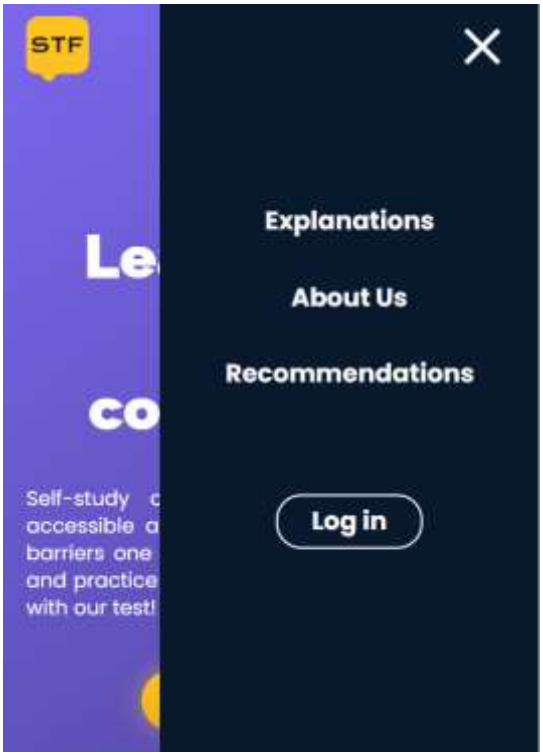


Рис 3.2. Відображення мобільного навігаційного меню з активованим списком

За допомогою мови JavaScript ми можемо вдосконалити сторінку з поясненнями, зробивши взаємодію з навчальним списком більш динамічною та зручною. Зокрема, потрібно реалізувати функцію розгортання та згортання кожної навчальної теми, а також анімацію повороту стрілки - вгору при розгортанні й вниз при згортанні. Крім того, варто додати автоскрол до останньої підтеми у списку для кращої видимості всіх елементів, особливо якщо їх кількість є обмеженою (наприклад, менше шести).

Щоб реалізувати згортання та розгортання навчальних тем на сторінці з поясненнями, створюється функція toggleContent, яка приймає два параметри - ідентифікатор блоку з контентом (id) та сам HTML-елемент, який викликав цю функцію (через ключове слово this у події onclick). Цю функцію ми підключаємо до

кожного елементу списку тем, додавши до блоку з класом topic атрибут onclick, який при натисканні передає потрібні значення у toggleContent.

Всередині функції ми спочатку отримуємо доступ до потрібних елементів DOM. Зокрема:

- content - це блок, у якому міститься список підтем. Ми знаходимо його за допомогою методу getElementById, передаючи туди id.
- arrow - це стрілка (іконка), яка розміщена в заголовку теми. За допомогою element.querySelector(".topic_arrow") ми отримуємо її для подальшої анімації.
- listItems - це список всіх пунктів (<li>) всередині блоку content. Він потрібен для перевірки кількості підтем і реалізації автоскролу.

Далі йде перевірка стану контенту. Якщо вміст наразі прихований (тобто його висота дорівнює 0px або взагалі не задана), ми виконуємо наступні дії:

- Встановлюємо стиль display: block, щоб елемент став видимим.
- Визначаємо максимальну висоту відповідно до фактичного розміру вмісту через scrollHeight, що дозволяє показати повний список із плавною анімацією.
- Змінюємо стиль стрілки (transform: rotate(180deg)), щоб вказати на зміну стану (відкрите положення).

Якщо в темі менше або рівно шести підтем, тоді виконується додаткове покращення інтерфейсу - автоскрол. Через невелику затримку (setTimeout) відбувається обчислення, чи нижня частина теми не виходить за межі видимого екрану. Якщо виходить - вмикається плавне прокручування (scrollBy) вниз, щоб вся тема була повністю доступна для перегляду. У випадку, якщо тема вже відкрита, функція виконує зворотну дію, щоб повернути стрілку в початковий стан. На рисунку 3.3 представлено результат роботи функції toggleContent.

Рис 3.3. Відображення розгорнутої навчальної теми на сторінці пояснень

Для реалізації функціоналу перевірки рівня знань володіння англійською мовою створюється окрема HTML-сторінка, яка структурно відповідає загальному дизайну веб-сайту. Вгорі сторінки розміщується стандартне навігаційне меню, а в центрі - фіолетовий контейнер, що вміщує весь основний вміст тесту. Його центральне розташування забезпечує зручність користування та візуальну цілісність сторінки. Для побудови логіки та візуального оформлення цієї сторінки використовуються наступні CSS класи: quiz-container, quiz-title, question, options, чотири кнопки вибору, question-number, progress-bar, progress-fill, result-container, result-text, level-circle, level-label, level-description та footer. Кожен із цих класів відповідає за окремі елементи структури: заголовок тесту, блок із запитанням, варіанти відповідей, поточний номер питання, панель прогресу, а також область з результатом тестування. Зі сторони дизайну сторінка гармонійно вписується в загальний стиль сайту - домінують фіолетові та білі кольори, всі елементи вирівняні по центру для зручності сприйняття та взаємодії.

Рис. 3.4. Дизайн сторінки для оцінювання рівня знань з англійської мови

На даному етапі необхідно створити повноцінний функціонал перевірки рівня знань за допомогою мови програмування JavaScript. Після вибору відповіді на запитання система одразу реагуватиме: правильна відповідь буде підсвічуватись зеленим кольором, а неправильна - червоним. Це забезпечить зручний візуальний зворотний зв'язок для користувача. Усі завдання тесту відображатимуться у випадковому порядку, який генеруватиметься під час кожного нового запуску тесту. Таким чином, кожне проходження матиме індивідуальний вигляд, що зменшує ймовірність запам’ятовування правильних варіантів лише за позицією. Кожне запитання супроводжуватиметься номером, який змінюється при переході до наступного, а також прогрес-баром, що поступово заповнюється у відповідності до кількості пройдених запитань. Система буде підраховувати набрані бали: за кожну правильну відповідь нараховується певна кількість балів. Їхня кількість буде залежити від складності конкретного завдання - чим складніше завдання, тим вищою буде винагорода. У разі, якщо користувач не знає правильної відповіді, передбачається можливість пропуску запитання через окрему кнопку, розміщену як четвертий варіант. Після завершення проходження тесту скрипт автоматично підрахує загальну кількість балів. Відповідно до набраної суми буде виведено підсумковий рівень знань, який супроводжуватиметься мотиваційним повідомленням. Текст цього повідомлення буде обумовлений від досягнутого

результату від рівня A1 до C1. Нижче наведено таблицю, що демонструє необхідну кількість балів для досягнення відповідного рівня.

Таблиця 3.1. Відповідність кількості балів рівню володіння англійською мовою

Рівень	Набрані бали
A1	0 - 9
A2	10 - 17
B1	18 - 27
B2	28 - 39
C1	40 - 50

Як і раніше, для реалізації логіки функціонування тесту на визначення рівня знань створюється окремий JavaScript-файл під назвою level_test.js. Цей файл буде відповідати за обробку всієї взаємодії користувача з тестовими завданнями: вибір відповіді, перевірка правильності, підрахунок балів, оновлення прогрес-бару, відображення номера запитання, формування фінального результату після завершення тесту, а також збереження прогресу після закриття сторінки та можливість повернення до поточного завдання. Після створення файл підключається до HTML-документу за допомогою тегу <script>, що забезпечує взаємодію між структурою сторінки та динамічними діями користувача. Для зберігання всіх наших завдань використовується масив із двадцятьма об’єктами, які будуть оброблюватися програмою під час виконання цього тесту. Масиви JS - це основна структура даних для зберігання впорядкованих колекцій елементів. Вони можуть містити будь-які типи даних, такі як числа, рядки, об’єкти і навіть інші масиви. Масиви мають динамічну природу - їхній розмір може змінюватися під час виконання програми. У контексті JS термін “асоціативний масив” зазвичай використовується замість поняття “об’єкт”. Об’єкти в JS - це структури даних, які містять пари ключ-значення. По суті, вони являють собою асоціативні масиви, де ключі є рядками (або символами), а значення можуть бути будь-якого типу даних, включно з іншими об’єктами, функціями, масивами та примітивними значеннями. Кожен елемент об’єкта являє собою пару ключ-значення, де ключ є ідентифікатором, за яким можна отримати

доступ до значення. Наприклад, в об’єкті {name: ‘Igor’, age: 35} ключами є name і age, а їхніми значеннями відповідно ‘Igor’ і 35[14].

Починатися наш скрипт буде з логіки початкового завантаження тесту, обробки прогресу користувача та збереження поточного стану у локальне сховище браузера (localStorage). Такий підхід дозволить продовжити виконання тесту з моменту зупинки, навіть після випадкового оновлення або закриття сторінки. На першому етапі перевіряється наявність раніше збережених даних за допомогою методу getItem, який отримує з localStorage об’єкт "quizState". Якщо такі дані знайдено, вони розпаковуються через JSON.parse() і використовуються для відновлення перемішаного списку запитань, номера поточного запитання та кількості вже набраних балів. Якщо збережених даних немає, скрипт виконує випадкове перемішування масиву quizData, щоб кожен запуск тесту відображав завдання у новому порядку. Далі відбувається ініціалізація змінних, які відповідають за логіку тестування, таких як currentQuestion, totalScore, а також вибираються необхідні HTML-елементи для подальшої взаємодії з DOM-структурою. Для фіксації проміжного результату реалізується функція saveProgress(), яка записує в локальне сховище поточний номер запитання, суму балів та порядок завдань. У разі потреби очистити ці дані використовується функція clearProgress(). Основне завантаження поточного завдання реалізується через функцію loadQuestion(), яка виводить текст запитання, генерує кнопки з варіантами відповідей, оновлює індикатор прогресу (progress-bar) та номер поточного запитання у форматі x / y. Кожна з кнопок створюється динамічно та отримує обробник події, який реагує на вибір користувача.

Рис. 3.5. Використання локального сховища для зберігання даних

Після завантаження запитань і виведення їх на екран наступним кроком є обробка дій користувача при виборі варіанту відповіді. Для цього створено функцію handleAnswer(), яка відповідає за перевірку правильності відповіді, зміну кольору кнопок залежно від результату, нарахування балів та плавне переключення до наступного запитання. Функція приймає два параметри: HTML-кнопку, на яку натиснув користувач, та значення вибраного варіанту. Спочатку визначається поточне запитання з масиву shuffledQuiz, а також отримуються всі кнопки відповідей у поточному блоці, щоб у подальшому керувати їхнім станом. Щоб запобігти повторному натисканню на кнопки після вибору відповіді, для контейнера з відповідями додається клас disabled, який блокує подальші дії до завершення обробки.

Особливий випадок обробляється окремо - якщо користувач натискає кнопку з текстом "I don't know how to answer this question" (тобто вирішує пропустити завдання), то система автоматично підсвічує правильну відповідь зеленим кольором, використовуючи клас correct, після чого через короткий проміжок часу автоматично переходить до наступного завдання. Якщо відповідь є правильною, до вибраної кнопки додається клас correct, і до загального результату (змінна totalScore) додається

кількість балів, передбачених для цього завдання. Якщо відповідь неправильна, то спершу кнопка набуває стилю incorrect, а потім, з невеликою затримкою, на екрані також підсвічується правильна відповідь, щоб користувач міг одразу побачити, яка саме опція була правильною.

Після цього викликається функція saveProgress(), яка оновлює локальне сховище інформацією про поточний стан тестування. Завдяки цьому зберігається безперервність процесу навіть у разі оновлення сторінки. Після завершення обробки відповіді викликається функція nextQuestion(), яка переходить до наступного елемента у масиві shuffledQuiz. Якщо запитання ще залишились, викликається функція loadQuestion(), яка повторно оновлює інтерфейс. У протилежному випадку, коли користувач пройшов усі завдання, основний вміст контейнера приховується, і відображається фінальний результат через функцію showFinalResult().

Фінальна частина скрипту завершує логіку проходження тесту, виводячи підсумковий результат користувача та визначаючи рівень володіння англійською мовою на основі кількості набраних балів. Цей блок відповідає за формування результату, його відображення на сторінці, очищення збережених даних і забезпечення повторного завантаження тесту після оновлення сторінки.

Розпочинається все з функції showFinalResult(). Вона аналізує фінальний результат користувача, збережений у змінній totalScore, і присвоює відповідний рівень володіння мовою: від A1 до C1. Оцінювання побудовано на умовних межах: чим більше балів, тим вищий рівень. Далі, на основі досягнутого рівня, формується мотивуюче повідомлення - воно відрізняється залежно від того, чи це початковий (рівень A), середній (рівень B) чи просунутий рівень (рівень C). Після обчислення рівня та повідомлення елементи сторінки оновлюються: змінюється заголовок тесту, повідомлення про завершення, очищується блок відповідей, встановлюється фінальний прогрес (100%) та відображається спеціальний контейнер з результатом - блок .result-container, який до цього був прихований. У ньому виводиться рівень (через .level-label) і текстове пояснення (через .level-description), що дозволяє користувачу оцінити свої досягнення та отримати додаткову мотивацію. Після

завершення тесту викликається clearProgress(), що очищає локальне сховище браузера від проміжних даних (localStorage), аби наступного разу почати тест із самого початку.

Останнім у скрипті є обробник події DOMContentLoaded, який запускається автоматично після повного завантаження HTML-документу. Цей блок перевіряє, чи є в локальному сховищі дані про незавершений тест. Якщо такі є, користувачеві пропонується вибір: продовжити з місця, де він зупинився, або почати тест спочатку. У випадку початку заново очищається сховище, обнуляється лічильник поточних питань, обнуляються бали, і запитання знову перемішуються випадковим чином. Далі завантажується перше запитання за допомогою функції loadQuestion(). Результат роботи всієї функціональної частини тесту для перевірки знань зображено на рисунку нижче.

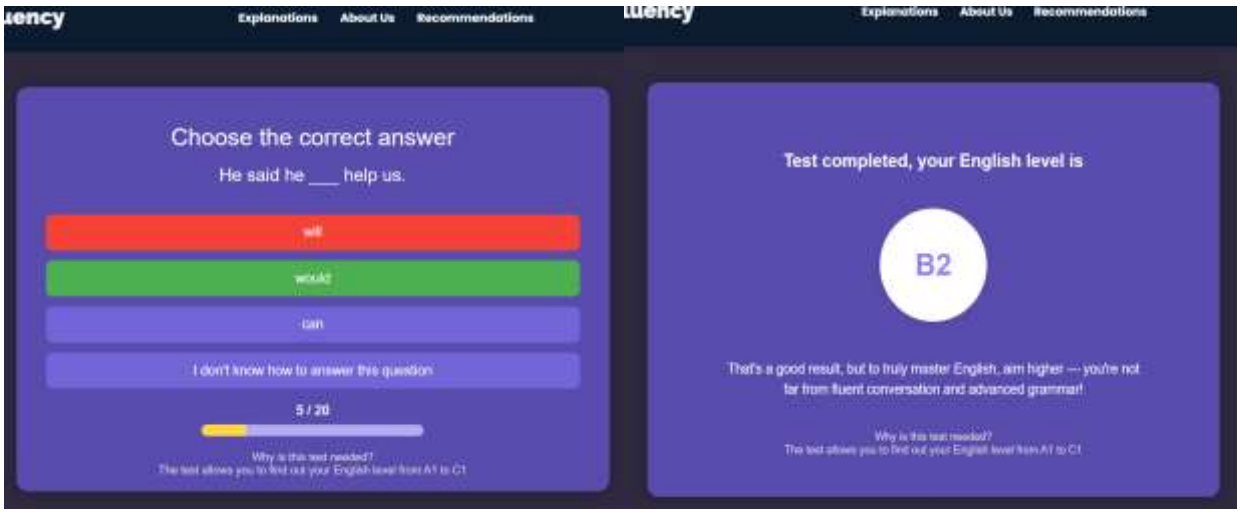


Рис. 3.6. Сторінка визначення рівня знань: а) проходження тесту; б) відображення результату

У результаті виконання даного підрозділу було реалізовано низку функціональних можливостей за допомогою мови програмування JavaScript, що значно підвищило динамічність і зручність веб-інтерфейсу. Використання JS у веб-розробці дозволяє створювати інтерактивні сценарії, які безпосередньо реагують на дії користувача, покращують візуальне сприйняття сайту та забезпечують логіку поведінки елементів без потреби в перезавантаженні сторінки. Було розроблено кілька важливих скриптів, серед яких: скрипт зміни вигляду навігаційного меню під час прокрутки, логіка відкриття і закриття мобільного меню, механізм розгортання навчального матеріалу із плавною анімацією та автоскролом, а також функціонал повноцінного онлайн-тесту для визначення рівня знань з англійської мови. Тест включає в себе динамічну генерацію завдань, обробку відповідей, підрахунок балів за складністю, збереження прогресу в локальному сховищі, виведення результату та визначення рівня володіння мовою згідно зі шкалою від A1 до C1.

3.2. Побудова backend-частини проекту за допомогою Node.js та Express

У сучасній веб-розробці серверна частина відіграє ключову роль у забезпеченні взаємодії між клієнтом (браузером користувача) та базою даних або іншими ресурсами системи. Без стабільної та безпечної серверної частини жоден динамічний веб-застосунок не зможе повноцінно функціонувати. Саме бекенд розробник відповідає за те, щоб сайт чи застосунок працював надійно, швидко та безпечно. В свою чергу, Back-end - це серверна частина додатку, яка виконує певні дії приховано від очей користувача. Вона відповідає за логіку обробки запитів, збереження та отримання даних, управління обліковими записами, авторизацію, взаємодію з зовнішніми API тощо. Back-end розробка спирається передусім на фреймворки, що базуються на Node.js, Python, Java, PHP тощо.

Коли користувач відправляє запит до серверу, бекенд обробляє його згідно зі своєю логікою: звертається до бази даних для отримання або оновлення інформації, виконує необхідні обчислення або взаємодіє з іншими сервісами через API. Після

обробки бекенд надсилає відповідь назад на фронтенд. Ця відповідь може містити інформацію, повідомлення про успішне виконання операції або інформацію про помилку, яку фронтенд відображає користувачеві. Таким чином, фронтенд і бекенд працюють як одне ціле, забезпечуючи повноцінну функціональність сайту чи застосунку[15].

Node.js - це однопоточне кросплатформове середовище виконання з відкритим вихідним кодом і бібліотека, яка використовується для запуску веб-додатків, написаних на JavaScript, поза браузером клієнта. Раніше виходило так, що JavaScript використовувався виключно у веб-браузерах, де його виконання забезпечувалося спеціальними вбудованими рушіями. Поза межами браузера ця мова практично не функціонувала. Ситуація змінилася з появою середовища Node.js, в основі якого лежить V8 - високопродуктивна машина виконання JavaScript, розроблена компанією Google для браузера Google Chrome. Завдяки Node.js стало можливим запускати JavaScript-код не лише у браузері, а й у будь-якому іншому середовищі, що дозволило використовувати цю мову як для створення клієнтської, так і серверної частини веб-застосунку[16].

Щоб наочно ознайомитися із серверною архітектурою, було створено сторінку з тестом на тему "Present Simple or Present Continuous". Основою для цієї сторінки є теги із класами test-container, pagination, test-page, stage-title, question та check-btn. До цієї сторінки також підключено файл стилів і скрипт, що відповідає за навігацію між завданнями.

Структурно реалізований тест складається з трьох окремих сторінок із завданнями, які послідовно проходить користувач для отримання фінального результату та нарахування балів. Перехід до наступної сторінки можливий лише після виконання завдань на поточній. На першій сторінці передбачено завдання на вибір правильного варіанту між двома часовими формами - Present Simple та Present Continuous. На другій - потрібно заповнити речення, вибравши правильну відповідь з трьох запропонованих варіантів. На третій, заключній сторінці, користувач самостійно вписує потрібні слова у відповідності до правильного граматичного часу.

Для забезпечення зручного переміщення між сторінками тесту було розроблено невеликий скрипт, що містить методи showPage, changePage, updateArrows та updateActiveArrows. Вони відповідають за виведення поточної сторінки, зміну активної секції, оновлення стану стрілок для навігації та візуальне підсвічування активної позиції в елементі pagination. Окрему увагу приділено контролю завершеності тесту: на останній сторінці розміщено кнопку "Results", яка залишається неактивною до моменту, поки користувач не виконає всі завдання на кожній зі сторінок.

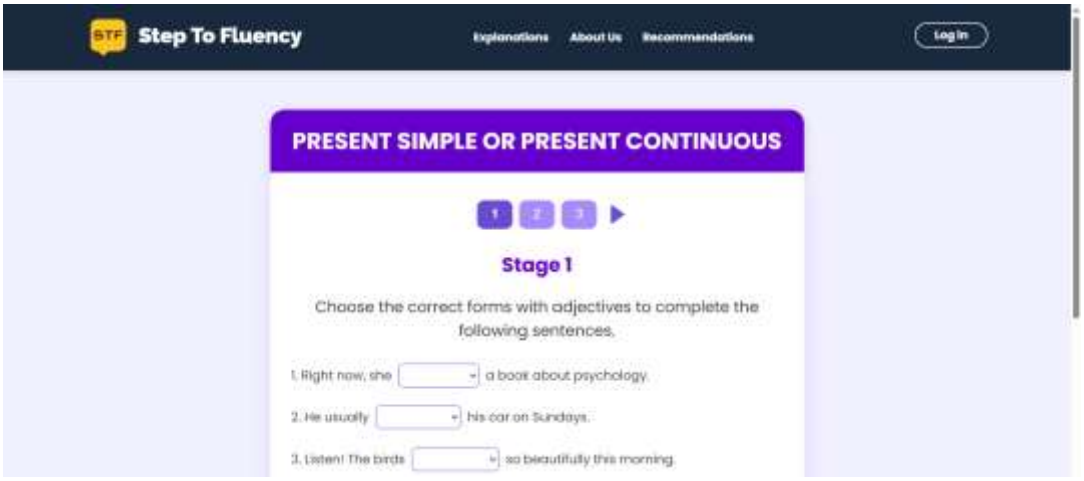


Рис. 3.7. Структура сторінки тесту Present Simple or Present Continuous

Для реалізації серверної частини веб-застосунку попередньо було встановлено останню стабільну версію Node.js. У середовищі Visual Studio Code створено дві окремі папки: frontend - для HTML, CSS і JavaScript-файлів, та server - для логіки бекенду. Далі в директорії server була ініціалізована система керування пакетами за допомогою команди npm init -y, що дозволило створити файл package.json для зберігання інформації про проект і залежності. До проекту також додаємо фреймворк Express, який значно спрощує створення веб-серверів і обробку HTTP-запитів. Його було встановлено через систему npm командою npm install express. У папці server створюється файл server.js, який відповідає за запуск сервера, налаштування маршрутів та підключення сторінок сайту. Сервер прослуховує локальний порт 3000, а завдяки підключенню Express налаштовано відображення основних HTML-сторінок сайту, включаючи головну сторінку, сторінку з поясненнями та окрему сторінку з

правилами Present Simple. При зверненні до зазначених URL сервер обробляє запити та надсилає відповідні HTML-файли. Якщо певна сторінка не знайдена, користувачу повертається повідомлення про помилку. На даний момент ми маємо повноцінну серверну частину, яка забезпечує коректну маршрутизацію та доставку контенту користувачу через браузер.

Рис. 3.8. Створення маршрутизації за допомогою Node.js та Express

У сучасних веб-застосунках надзвичайно важливо мати можливість зберігати, обробляти та відновлювати дані користувачів. Для реалізації цієї функціональності необхідно використовувати систему керування базами даних. У межах розробленого проекту база даних потрібна для збереження інформації про зареєстрованих користувачів, їхні результати тестувань, накопичені бали та інші динамічні дані. Створення першої бази даних доцільно почати з використання PostgreSQL. PostgreSQL - це об'єктно-реляційна система управління базами даних (СУБД), відома своєю надійністю, потужністю та гнучкістю. PostgreSQL є проектом з відкритим вихідним кодом, що робить його привабливим вибором для багатьох компаній і розробників. Відсутність витрат на ліцензії дозволяє ефективніше розпоряджатися ресурсами, а можливість адаптації системи під конкретні потреби дає гнучкість у розробці. Над її вдосконаленням постійно працює велика спільнота, яка додає нові можливості, усуває помилки та підтримує актуальність системи. Крім того, користувачі мають доступ до великої кількості довідкових матеріалів, форумів і

документації. СУБД PostgreSQL сумісна з офіційними стандартами SQL і підтримує розширення, що дозволяє легко інтегрувати її з іншими програмними системами. Вона також забезпечує можливість створення власних типів даних, функцій і операторів, що значно розширює можливості налаштування системи під специфічні вимоги. Щодо забезпечення стабільності та збереження даних, PostgreSQL надає потужні засоби, зокрема підтримку транзакцій з дотриманням властивостей ACID, можливість відновлення системи до конкретного моменту часу (point-in-time recovery), засоби реплікації даних і автоматичного відновлення, що гарантує надійну роботу навіть у разі збоїв[17].

Щоб розпочати роботу з системою керування базами даних PostgreSQL, насамперед потрібно завантажити її з офіційного веб-сайту. Під час встановлення важливо також обрати пункт для завантаження pgAdmin4 - графічного інтерфейсу керування базами даних, що значно спрощує процес адміністрування. Після завершення встановлення ми відкриваємо pgAdmin та здійснюємо підключення до сервера PostgreSQL. Це є обов’язковою умовою, без якої подальша робота з базою даних неможлива. Після успішного підключення створюємо нову базу даних і надаємо їй ім’я english_test. Згідно зі структурою нашого веб-сайту, ця база повинна містити таблиці, що відображають інформацію про обраний тест (разом з його номером), завдання, відповіді, обрані користувачем, а також підсумкові результати. Один тест може містити декілька завдань, кожне завдання - декілька варіантів відповідей, а до одного тесту може бути прив’язано кілька результатів, які відображають різні проходження. Рисунок 3.10 демонструє SQL-запити створення таблиць, що відповідають структурі, зображеній на рисунку 3.9.

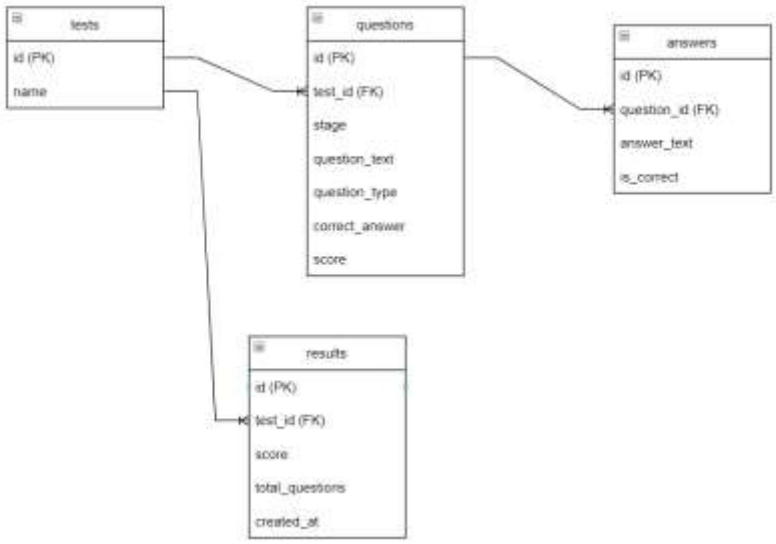


Рис. 3.9. Схема структури бази даних системи тестування

```sql
CREATE TABLE tests (
  id SERIAL PRIMARY KEY,
  name VARCHAR(100) NOT NULL
);

CREATE TABLE questions (
  id SERIAL PRIMARY KEY,
  test_id INTEGER REFERENCES tests(id),
  stage INTEGER NOT NULL,
  question_text TEXT NOT NULL,
  question_type VARCHAR(20) NOT NULL,
  correct_answer TEXT,
  score INTEGER NOT NULL DEFAULT 1
);

CREATE TABLE answers (
  id SERIAL PRIMARY KEY,
  question_id INTEGER REFERENCES questions(id),
  answer_text TEXT NOT NULL,
  is_correct BOOLEAN DEFAULT FALSE
);

CREATE TABLE results (
  id SERIAL PRIMARY KEY,
  test_id INTEGER REFERENCES tests(id),
  score INTEGER NOT NULL,
  total_questions INTEGER NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Рис. 3.10. Створення таблиць в PostgreSQL

Щоб забезпечити взаємодію з базою даних PostgreSQL у середовищі розробки Visual Studio Code, необхідно налаштувати з’єднання між серверною частиною проекту, що реалізована на Node.js, і самою базою даних. З цією метою створюється окремий модуль, db.js, який відповідатиме за підключення до PostgreSQL. У цьому файлі використовується бібліотека pg, що надає всі необхідні інструменти для створення підключення та виконання SQL-запитів. Також підключається пакет

dotenv, що дозволяє зчитувати параметри підключення (логін, пароль, назву бази даних, порт тощо) із зовнішнього конфігураційного файлу .env. Файл .env містить усю конфіденційну інформацію про базу даних, яку не рекомендується розміщувати безпосередньо в коді.

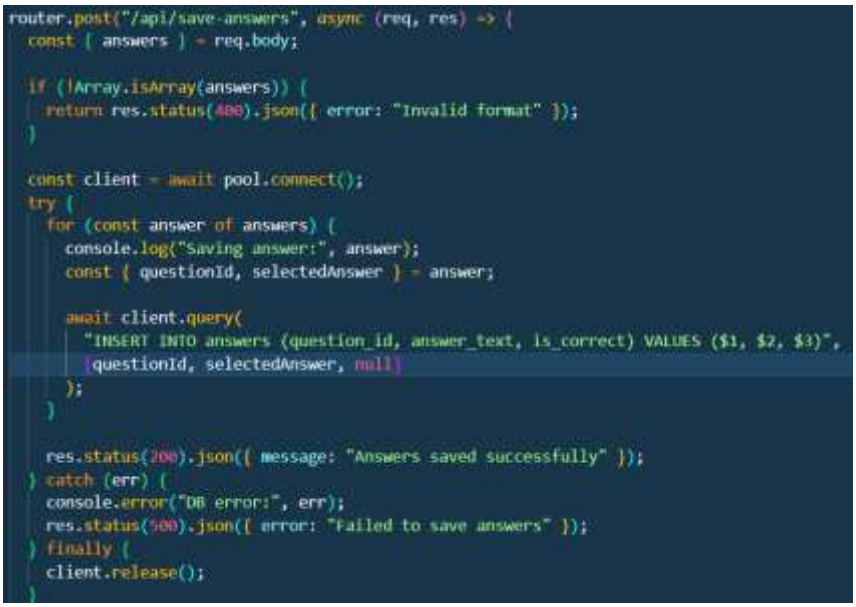
Для забезпечення ефективної взаємодії між серверною частиною проекту на Node.js і базою даних PostgreSQL необхідно створити окремі модулі - контролери та роути. Вони дозволяють логічно структурувати код, розділивши функціональність на окремі частини: контролери відповідають за обробку запитів і бізнес-логіку, а маршрути (роути) - за визначення шляхів, за якими ці запити обробляються. Один із контролерів, наприклад, містить функцію, яка повертає користувачеві HTML-сторінку тесту на тему Present Simple or Present Continuous. У цьому контролері використовується модуль path, а також fileURLToPath, щоб динамічно сформувати шлях до необхідного файлу в папці frontend. Це забезпечує коректне завантаження тестової сторінки через маршрут.

Рис. 3.11. Кодова структура файлу testController.js

Файл testRoutes.js буде відповідати за обробку маршрутів, пов’язаних із тестами. У ньому реалізовані шляхи, які дозволяють:

- зберігати відповіді користувача до бази даних;
- перевіряти правильність обраних відповідей;
- передавати результати тестування для подальшого аналізу або відображення.

Таким чином, контролери виконують роль "посередника" між користувачем, сервером та базою даних, тоді як роутери визначають, за якими URL-адресами ці дії повинні виконуватись.



```
router.post("/api/save-answers", async (req, res) => {
  const { answers } = req.body;

  if (!Array.isArray(answers)) {
    return res.status(400).json({ error: "Invalid format" });
  }

  const client = await pool.connect();
  try {
    for (const answer of answers) {
      console.log("Saving answer:", answer);
      const { questionId, selectedAnswer } = answer;

      await client.query(
        "INSERT INTO answers (question_id, answer_text, is_correct) VALUES ($1, $2, $3)",
        [questionId, selectedAnswer, null]
      );
    }

    res.status(200).json({ message: "Answers saved successfully" });
  } catch (err) {
    console.error("DB error:", err);
    res.status(500).json({ error: "Failed to save answers" });
  } finally {
    client.release();
  }
```

Рис. 3.12. Шлях до API для збереження результатів тесту в PostgreSQL

У процесі реалізації функціоналу для повного проходження граматичного тесту на тему Present Simple or Present Continuous створюється окремий JavaScript-файл із назвою collectAnswer.js. Його основне призначення - зібрати відповіді користувача з усіх типів запитань (випадаючі списки, радіокнопки, текстові поля), відправити ці відповіді на сервер, перевірити їхню правильність і показати пояснення.

На першому етапі в коді реалізується функція collectAnswers(), яка проходиться по всіх елементах із відповідним атрибутом data-question-id. Усі відповіді записуються у масив об'єктів, кожен з яких містить ідентифікатор запитання та вибрану відповідь. Завдяки цьому ми отримуємо універсальне рішення для збору даних незалежно від типу запитань. Після цього, коли користувач натискає на кнопку перевірки (.check-btn) на одній із сторінок тесту, дані передаються на сервер за допомогою HTTP-запиту POST на маршрут /api/save-answers, де відбувається збереження відповідей у базі даних. Одразу після цього виконується другий запит на маршрут /grammar/present-simple-or-continuous-test/check для перевірки правильності відповідей. У відповідь від сервера клієнтська частина отримує результат перевірки, після чого виводиться пояснення до кожного запитання. Ці пояснення беруться зі змінної feedbackData, у якій для кожного questionId зберігається короткий коментар щодо правильності вибору і граматичне правило, що пояснює відповідь.

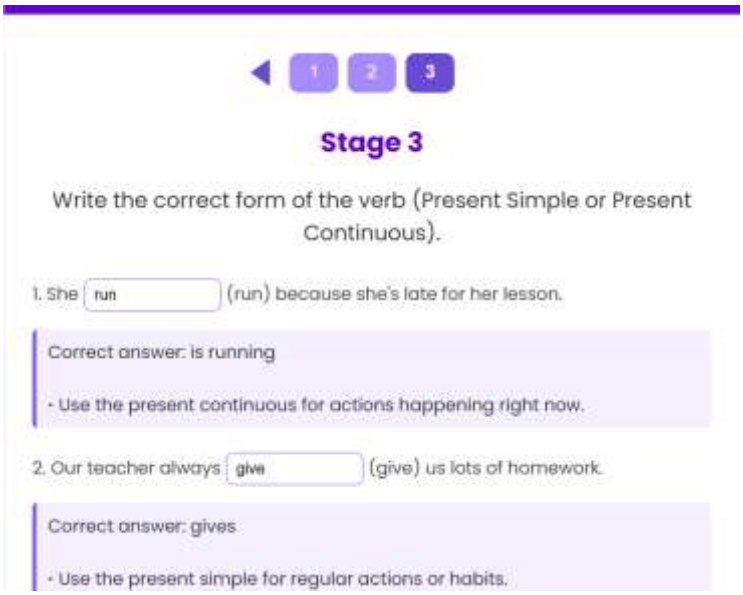


Рис. 3.13. Пояснення до кожного завдання після перевірки відповідей

Після проходження всіх сторінок тесту, з’являється блок із підсумковим результатом - кількістю балів, які набрав користувач. Система нараховує ці бали на основі складності завдань та збережених відповідей із таблиці "Results".

Таким чином, реалізований скрипт дозволяє інтерактивно обробляти відповіді, зберігати їх у базу даних, перевіряти на коректність, пояснювати помилки користувача та формувати фінальний результат, що робить тест повноцінним та придатним до навчального використання.

ВИСНОВОК

У ході виконання кваліфікаційної роботи було створено інтерактивний веб-застосунок, призначений для вдосконалення знань з англійської мови, з акцентом на доступність, зручність використання та функціональну повноту.

Першим етапом став аналіз предметної області: розглянуто сучасні освітні онлайн-платформи, порівняно їхні можливості, виявлено переваги та обмеження. Це дозволило сформулювати вимоги до структури та функціоналу майбутнього застосунку. Наступним кроком було проєктування зовнішнього вигляду сайту. Створено макети сторінок, реалізовано стильний інтерфейс у біло-фіолетовій кольоровій гамі, враховано адаптивність під різні типи пристроїв. Особливу увагу приділено логіці навігації та зручності користування ресурсом.

Фінальна частина роботи зосереджувалася на реалізації функціональної логіки. За допомогою JavaScript створено динамічні елементи, зокрема тест з перевіркою рівня володіння мовою. Серверну частину розгорнуто на Node.js з використанням фреймворку Express. Для зберігання даних інтегровано базу PostgreSQL, яка обробляє відповіді користувачів, формує результат та зберігає історію проходження. Результатом стало повноцінне рішення, яке може бути використане як самостійний інструмент для онлайн-навчання, так і як додатковий ресурс у навчальному процесі. Веб-застосунок легко розширюється, має гнучку архітектуру та відповідає сучасним вимогам веб-розробки.