ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка інформаційного web-сайту для підвищення екологічної свідомості громадян»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

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

_____ Денис ДІДИЛІВСЬКИЙ
(підпис)

Виконав: здобувач вищої освіти групи ПД-41

Денис ДІДИЛІВСЬКИЙ

Керівник: Владислав ЯСКЕВИЧ
к.т.н., доцент

Рецензент: _____

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

_____Дідилівському Денису Віталійовичому_____

1. Тема кваліфікаційної роботи: «Розробка інформаційного web-сайту для підвищення екологічної свідомості громадян»
керівник кваліфікаційної роботи к.т.н., доцент Владислав ЯСКЕВИЧ,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025 р. № 56.

2. Строк подання кваліфікаційної роботи «02» червня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про вебсайт для підвищення екологічної свідомості громадян, аналіз аналогів для підвищення екологічної свідомості, технічна документація до використаних бібліотек та фреймворків.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Огляд та аналіз існуючих веб-ресурсів, спрямованих на підвищення екологічної свідомості.
2. Проектування вебсайта для підвищення екологічної свідомості громадян.
3. Програмна реалізація та опис функціонування сайта для підвищення екологічної свідомості громадян

4. Тестування сайта.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до вебсайту.
3. Програмні засоби реалізації.
4. Діаграма варіантів використання.
5. Діаграми діяльності.
6. Мапа вебсайта.
7. Екранні форми.
8. Апробація результатів дослідження

6. Дата видачі завдання «25» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	25.02.-04.03.2025	
2	Аналіз та дослідження існуючих аналогів	05.03-13.03.2025	
3	Огляд існуючих веб-ресурсів для підвищення екологічної свідомості	14.03-18.03.2025	
4	Проектування сайта для підвищення екологічної свідомості громадян	19.03-26.03.2025	
5	Програмна реалізація застосунку	27.03-25.04.2025	
6	Тестування застосунку	26.04-28.04.2025	
7	Оформлення роботи: вступ, висновки, реферат	29.04-11.05.2025	
8	Розробка демонстраційних матеріалів	12.05-18.05.2025	
9	Попередній захист роботи	19.05-30.05.2025	

Здобувач вищої освіти _____ Денис ДІДИЛІВСЬКИЙ
(підпис)

Керівник
кваліфікаційної роботи _____ Владислав ЯСКЕВИЧ
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 54 стор., 2 табл., 31 рис., 20 джерел.

Мета роботи – підвищення екологічної свідомості громадян за рахунок використання інтерактивних методів навчання, реалізованих у вигляді інформаційного вебсайта.

Об'єкт дослідження – процес підвищення екологічної свідомості громадян за допомогою вебсайта.

Предмет дослідження – вебсайт EcoAware як засіб інтерактивного підвищення екологічної обізнаності населення.

Короткий зміст роботи: В роботі було проаналізовано вебсайт для підвищення екологічної свідомості громадян. Було проаналізовано наявні аналоги, які мають відношення до екологічної свідомості: Екологія Право Людина, Гагарка, Гагарка. Розроблено вебсайт та було реалізовано наступні функціональні можливості: реєстрація та авторизація користувачів, можливість перегляду та пошуку екологічних статей, можливість проходження тестів, можливість участі у грі, можливість переглядати персональну статистику. Проведено тестування вебсайта за допомогою створених тестових сценаріїв. В роботі використано Використано Node.js, фреймворк Express.js та нереляційну базу даних MongoDB для створення серверної частини. Для реалізації автентифікації використано бібліотеку JSON Web Token. Клієнтська частина розроблена з використанням React.

Сферою використання вебсайта є підвищення екологічної свідомості громадян.

КЛЮЧОВІ СЛОВА: ЕКОЛОГІЧНА СВІДОМІСТЬ, NODE.JS, REACT, MONGODB.

ЗМІСТ

ВСТУП

У сучасному світі екологічні проблеми набувають дедалі більшої актуальності та вимагають не лише уваги з боку державних інституцій та великих підприємств, але й свідомої участі кожного громадянина у процесі захисту довкілля. Формування екологічної свідомості населення є ключовим фактором досягнення сталого розвитку суспільства та збереження природних ресурсів для майбутніх поколінь. Інформаційні технології, зокрема вебресурси, стають потужним інструментом для поширення екологічних знань, формування навичок екологічно відповідальної поведінки та залучення різних соціальних груп до природоохоронної діяльності.

Стрімкий розвиток інтернет-технологій та зростання доступності цифрових ресурсів створюють сприятливе середовище для впровадження інтерактивних методів екологічної освіти. Використання ігрових елементів, тестових завдань та інших форм активного навчання значно підвищує ефективність засвоєння інформації та мотивацію до практичного застосування отриманих знань у повсякденному житті.

Мета роботи – підвищення екологічної свідомості громадян за рахунок використання інтерактивних методів навчання, реалізованих у вигляді інформаційного вебсайта.

Об'єкт дослідження – процес підвищення екологічної свідомості громадян за допомогою вебсайта.

Предмет дослідження – вебсайт EcoAware як засіб інтерактивного підвищення екологічної обізнаності населення.

Для реалізації поставленої мети виділено такі задачі:

1. Проаналізувати наявні веб-ресурси, спрямовані на підвищення екологічної обізнаності.
2. Визначити функціональні та нефункціональні вимоги до інформаційного вебсайту.

3. Розробити структуру сайту з урахуванням розділів: екознання, еко-гра, тести.

4. Реалізувати функціонал реєстрації користувачів, проходження тестів, еко-гри і перегляд статистики.

5. Провести тестування вебсайту.

Для розробки вебсайту EcoAware було обрано сучасний стек технологій, що включає React.js для створення інтерактивного користувацького інтерфейсу, Node.js як серверну платформу, Express.js для розробки API та маршрутизації, MongoDB для зберігання даних, Bootstrap для швидкого та адаптивного дизайну, та JSON Web Tokens (JWT) для безпечної автентифікації користувачів. Такий набір інструментів забезпечує створення зручного, швидкого та масштабованого вебзастосунку з адаптивним інтерфейсом для користувачів різних вікових категорій.

Очікуваним результатом роботи є створення функціонального вебсайту, який стане ефективним інструментом підвищення екологічної свідомості громадян. Вебсайт EcoAware надасть користувачам доступ до екологічних знань, можливість перевірити свої знання через тести та брати участь в еко-грі, що сприятиме формуванню навичок екологічно відповідальної поведінки. Завдяки зручному інтерфейсу та можливості відстежувати особистий прогрес, ресурс сприятиме поширенню екологічних знань та формуванню активної спільноти людей, зацікавлених у збереженні довкілля.

Робота пройшла апробацію: Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в ІКТ» за темою «Дослідження розробки інформаційного web-сайту для підвищення екологічної свідомості громадян»[1]; VI Науково-технічна конференція «Сучасний стан та перспективи розвитку IoT» за темою «Використання протоколу OAuth 2.0 для захисту даних користувачів у вебзастосунках» [2].

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність вебсайта для підвищення екологічної свідомості громадян

У контексті глобальних екологічних викликів XXI століття, формування екологічної свідомості є питанням виживання суспільства або його деградації [3, с. 42]. Екологічна криза, що проявляється у змінах клімату, забрудненні повітря та водойм, скороченні біорізноманіття, виснаженні природних ресурсів та накопиченні відходів, потребує не лише технологічних і законодавчих рішень, але й глибинних змін у світогляді та повсякденних практиках кожної людини.

Екологічна свідомість громадян характеризується суперечливістю: декларативне ставлення до довкілля здебільшого не відповідає реальним діям та готовності до заходів зі збереження природи [4, с. 18]. Тим самим спостерігається низький рівень практичного впровадження екологічних принципів у повсякденне життя: незначна частина населення регулярно сортує побутові відходи, обмежує використання пластику чи враховує екологічні фактори при здійсненні покупок. Особливо гострою є проблема низької екологічної свідомості серед молоді — покоління, яке найбільше постраждає від наслідків екологічної кризи в майбутньому. Вирішення цієї проблеми потребує застосування сучасних підходів для підвищення екологічної свідомості, що враховують особливості сприйняття інформації громадян.

Розвиток екологічної свідомості населення через інформаційні технології дозволяє формувати нову систему цінностей та підвищувати рівень екологічної свідомості суспільства. Ключовою перевагою цифрових технологій у підвищенні екологічної свідомості є їх доступність, інтерактивність та здатність персоналізувати навчальний процес.

На відміну від традиційних форматів екологічної просвіти (лекції, брошури, статті), інтерактивні вебсайти здатні забезпечити активне залучення користувачів через тести та ігри;

Використання цифрових платформ з елементами гейміфікації значно підвищує залученість аудиторії та збільшує ймовірність впровадження екологічних практик у повсякденне життя порівняно з традиційними методами просвіти.

На сьогоднішній день існує ряд українських та міжнародних вебресурсів екологічного спрямування, проте більшість з них мають суттєві обмеження. Інформаційні сайтів екологічних організацій містять якісний контент, але вони лише надають теоретичні знання і не використовують інтерактивні методи навчання тим самим знижується залученість користувачів.

Цільовою аудиторією вебсайту є учні старших класів та студенти, дорослі, які бажають перейти до більш екологічного способу життя, а також місцеві громади та активісти, зацікавлені в просуванні екологічних ініціатив.

Очікуваний вплив проекту включає підвищення рівня екологічної обізнаності серед користувачів сайту, збільшення кількості громадян, які впроваджують екологічні практики у повсякденне життя, та формування спільноти екологічно свідомих громадян.

Розробка інтерактивного вебсайту для підвищення екологічної свідомості громадян є надзвичайно актуальним завданням з огляду на критичний стан довкілля та необхідність масштабних поведінкових змін, недостатній рівень екологічної свідомості серед українців та обмежений доступ до якісної екологічної освіти, потенціал цифрових технологій для ефективного формування екологічної культури та відсутність комплексних інтерактивних україномовних ресурсів з екологічної тематики.

Реалізація даного проекту дозволить створити інноваційний інструмент екологічної просвіти, що відповідає сучасним освітнім практикам, потребам цільової аудиторії та технологічним можливостям.

1.2 Аналіз аналогів

Для підвищення екологічної свідомості громадян може бути використано різноманітні цифрові технології, серед яких важливе місце посідають інформаційні вебсайти. Вони надають доступ до актуальної інформації, освітніх матеріалів. В цьому контексті варто розглянути детальніше декілька прикладів таких сайтів та оцінити їх можливості та недоліки в підвищені екосвідомості:

- Екологія Право Людина;
- Гагарка;
- Всесвіт.

1.2.1 Екологія Право Людина

Екологія Право Людина – є важливим джерелом інформації щодо екологічних прав, законодавчих змін, випадків порушення екологічного законодавства та шляхів їх вирішення [5]. Він підвищує екологічну свідомість через інформування громадян про їхні права та можливості впливу на стан довкілля з правової точки зору.

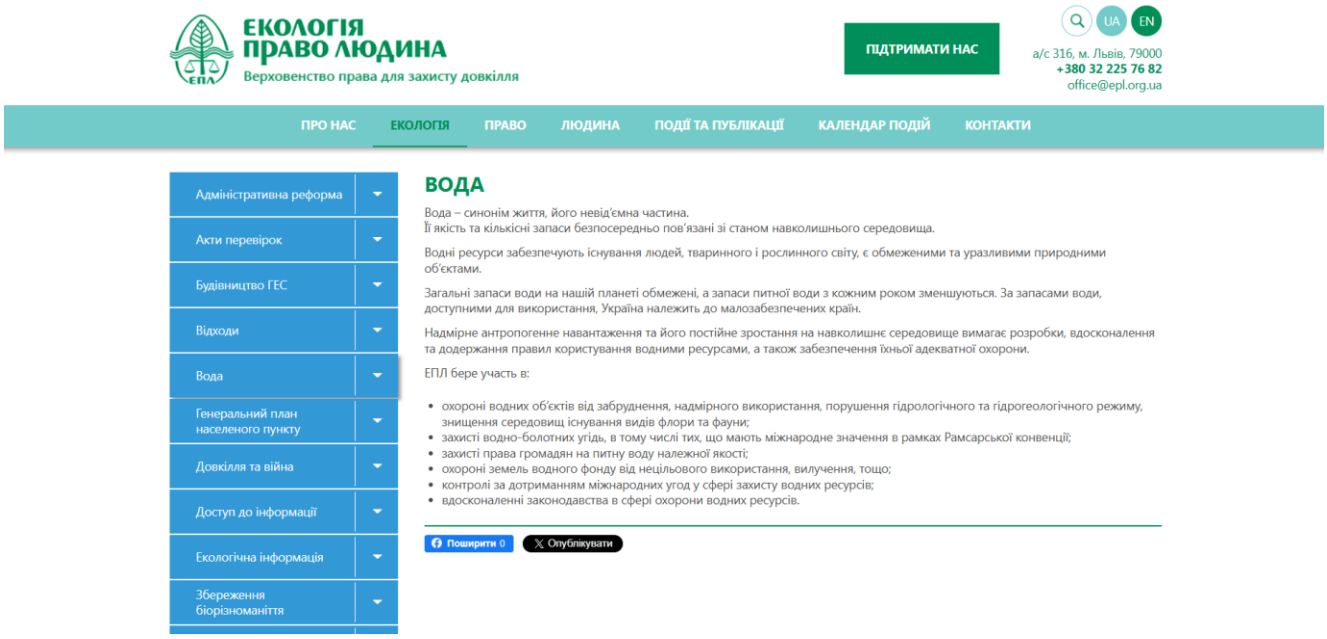


Рис. 1.1. Приклад екологічної статі у вебсайті "Екологія Право Людина"

Основні переваги вебсайта:

- Надаються конкретні статі щодо захисту екологічних прав;
- Можливість знаходити статі за категоріями і за допомогою пошуку.

Основні недоліки вебсайта:

- Немає можливості для користувачів створювати профілі, відстежувати свій прогрес;
- Сайт переважно інформаційний та аналітичний, на ньому відсутні інтерактивні тести, еко-ігри.

Можна зробити висновок, що вебсайт є надійним і цінним джерелом інформації для осіб, які цікавляться екологічним правом та захистом довкілля з юридичної точки зору. Проте з точки зору залучення широкої аудиторії та інтерактивної взаємодії з користувачами сайт має обмежені можливості, що може ускладнювати популяризацію екологічних знань серед пересічних громадян, особливо початківців у сфері екології.

1.2.2 Гагарка

Вебсайт Гагарка позиціонує себе як ресурс про цікаву екологію. Він спрямований на популяризацію екологічних знань у доступній та захопливій формі [6]. Сайт пропонує статті на різноманітні екологічні теми, які можуть бути цікаві широкій аудиторії. Він сприяє підвищенню обізнаності через подачу матеріалу в науково-популярному стилі.

Основні переваги вебсайта:

- Статі охоплюють широкий спектр питань – від сортування сміття та еко-звичок до глобальних екологічних проблем та цікавих фактів про природу;
- Наявність знаходження статі за категоріями і за пошуком.

Основні недоліки вебсайта:

- Немає можливості для створення особистих кабінетів;
- Сайт не пропонує інтерактивних тестів, еко-ігор.

Рис. 1.2 Приклад екологічної статі у вебсайті "Гагарка"

Можна зробити висновок, що вебсайт Гагарка є привабливим та доступним ресурсом для широкої аудиторії, яка цікавиться екологічними питаннями. Він успішно виконує просвітницьку функцію, подаючи екологічні знання у зрозумілій та візуально привабливій формі. Проте з точки зору глибини аналізу та інтерактивної взаємодії з користувачами сайт має обмеження, що може знижувати рівень залученості відвідувачів та утримання їхньої уваги в довгостроковій перспективі.

1.2.3 Всесвіт

Вебсайт vse-svit.com.ua позиціонується як екологічний портал, що має на меті всебічне висвітлення екологічних питань [7]. Сайт надає новини, статті,

інформацію про екологічні події та організації. Він сприяє підвищенню екологічної свідомості через інформування та освіту.

Основні переваги вебсайта:

- Публікуються матеріали на різноманітні екологічні теми;
- Можливість знаходити матеріали за розділами і за допомогою пошуку.

Основні недоліки вебсайта:

- Не передбачено створення профілів користувачів для відстеження їх активності чи прогресу в навчанні;
- Як і попередні ресурси, сайт переважно є інформаційним. Відсутні інтерактивні тести, ігри, система рейтингу.

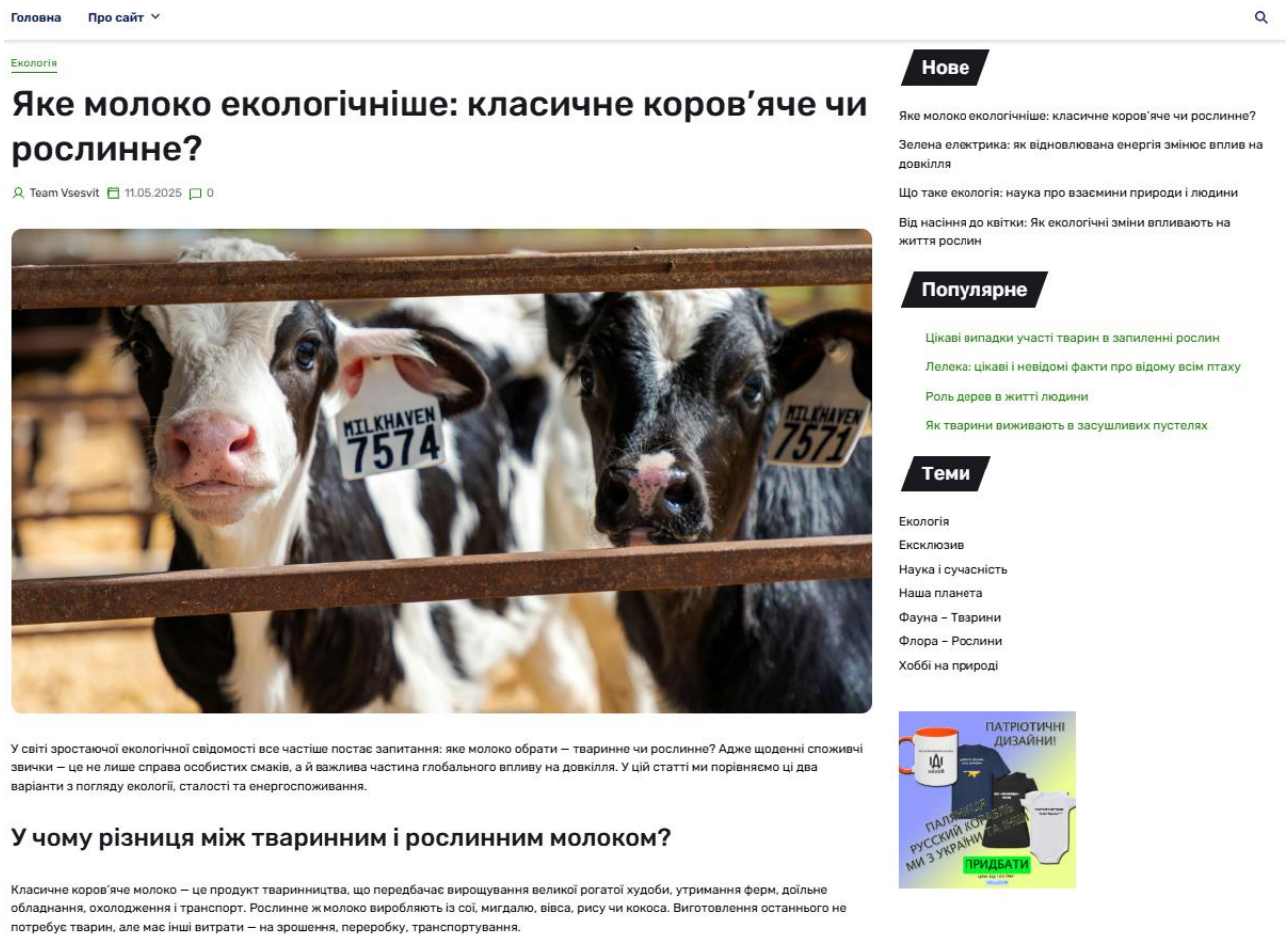


Рис. 1.3 Приклад екологічної статі у вебсайті "Всесвіт"

Можна зробити висновок, що Всесвіт є корисним інформаційним екологічним сайтом з потенціалом для широкої аудиторії. Сайт успішно забезпечує інформаційну підтримку з екологічних питань, пропонуючи різноманітний контент про новини, технології, здоров'я та екологічний туризм. Водночас, з точки зору залучення користувачів та глибини взаємодії, ресурс має значні обмеження, оскільки бракує інтерактивних елементів, персоналізації та поглибленого аналізу тематичних матеріалів.

1.2.4 Таблиця порівнянь аналогів

Зведені результати аналізу характеристик розглянутих вебзастосунку наведено у таблиці 1.1.

Таблиця 1.1

Порівняння існуючих аналогів

Характеристика	**Екологія Право Людина**	**Гагарка**	**Всесвіт**	**EcoAware**
Екологічні статі	+	+	+	+
Екологічні тести	-	-	-	+
Інтерактивна гра	-	-	-	+
Персональна статистика	-	-	-	+
Пошук і фільтрація по категоріям	+	+	+	+

2 ПРОЄКТУВАННЯ ВЕБСАЙТА

2.1 Функціональні та нефункціональні вимоги

За дослідженими аналогами були сформовані основні функціональні вимоги для вебсайта:

1. Реєстрація та авторизація користувачів. Сайт повинен надавати користувачам можливість створювати нові облікові записи та безпечно авторизуватися для доступу до функціоналу вебсайту. Це забезпечує персоналізований досвід та відстеження даних.

2. Можливість перегляду, пошуку та фільтрації екологічних статей. Користувачі повинні мати можливість переглядати колекцію екологічних статей з можливістю пошуку конкретних тем та фільтрації контенту за категоріями.

3. Можливість проходження екологічних тестів з отриманням результатів. Сайт повинен пропонувати інтерактивні тести з екологічних знань, які зберігають результати для майбутнього використання, дозволяючи користувачам відстежувати свій прогрес та покращення знань.

4. Можливість участі у екологічній грі з системою рейтингу. Користувачі повинні мати доступ до інтерактивної екологічної гри, яка включає елементи гейміфікації, у тому числі систему рейтингу для залучення нових користувачів.

5. Можливість переглядати персональну статистику прочитаних матеріалів, пройдених тестів та ігрового прогресу. Сайт повинен підтримувати та відображати персоналізовану статистику користувача, яка відстежує їхню взаємодію зі статтями, результати тестів та прогрес в іграх, створюючи комплексний огляд навчального шляху.

Нефункціональні вимоги визначають важливі аспекти вебсайту, впливаючи на зручність використання, надійність та продуктивність:

1. Зберігання персональних даних користувачів в зашифрованому вигляді за допомогою JWT. Система повинна забезпечувати належну безпеку для особистої інформації користувачів шляхом впровадження шифрування JWT (JSON Web Token), захищаючи конфіденційні дані від несанкціонованого доступу.

2. Вебсайт має працювати в браузерах Chrome, Edge, Opera. Платформа повинна правильно функціонувати та відображатися в кількох популярних веб-браузерах, включаючи Chrome, Edge та Opera, забезпечуючи широку доступність.

3. Завантаження сторінки сайту до двох секунд. Вебсайт повинен бути оптимізований для продуктивності з часом завантаження сторінок, що не перевищує двох секунд, забезпечуючи плавний та відповідний досвід користувача.

2.2 Проектування варіантів використання для вебсайта

Варіанти використання ілюструють методи взаємодії користувача з екологічним вебсайтом для отримання знань та підвищення екологічної свідомості, задовольняючи їхні освітні потреби. Діаграма варіантів використання слугує важливим візуальним інструментом, що демонструє ці взаємодії та допомагає краще зрозуміти функціональні вимоги системи з перспективи користувача.

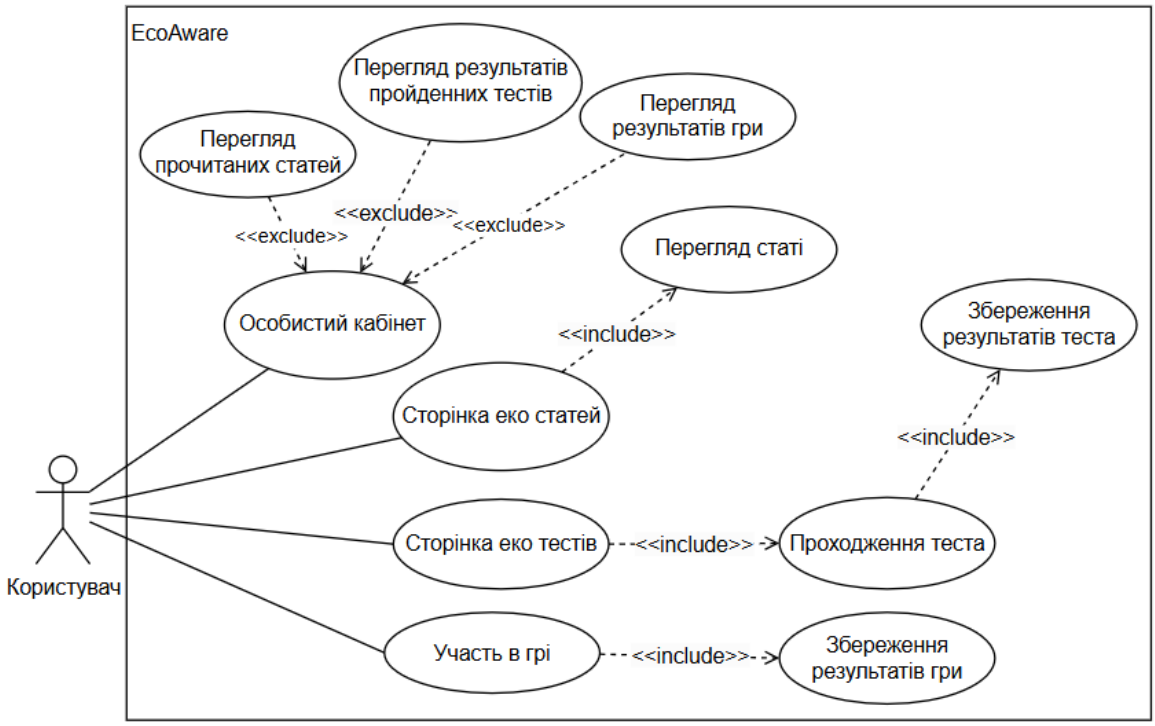


Рис. 2.1 Діаграма використання вебсайта EcoAware

На даній діаграмі основним актором є користувач вебсайту. Випадки використання включають такий перелік дій:

- Особистий кабінет: Дозволяє користувачеві доступ до персоналізованого простору, який виключає в собі перегляд прочитаних статей, перегляд результатів пройдених тестів та перегляд результатів гри, забезпечуючи цілісний огляд прогресу навчання;
- Сторінка еко статей: Дозволяє користувачеві отримати доступ до екологічних статей та перегляд статті для детального ознайомлення з конкретною темою;
- Сторінка еко тестів: Дозволяє користувачеві переходити до розділу з екологічними тестами та включає в себе проходження тесту, яке, в свою чергу, включає збереження результатів тесту для подальшого аналізу;
- Участь в грі: Дозволяє користувачеві брати участь в інтерактивній екологічній грі та включає в себе збереження результатів гри, що дозволяє відстежувати прогрес та досягнення.

2.3 Архітектура вебсайта

Вебсайт EcoAware, спрямований на підвищення екологічної обізнаності населення, реалізовано на основі клієнт-серверної архітектури. Така структура передбачає поділ системи на два взаємодіючі компоненти:

- Клієнтська складова: Створена на базі сучасної бібліотеки React, ця частина забезпечує візуалізацію користувацького інтерфейсу та опрацювання взаємодій відвідувачів сайту. Фронтенд комунікує з серверною частиною через запити для отримання й збереження інформації та подальшого її представлення користувачам. Для забезпечення естетичного та адаптивного дизайну інтерфейсу застосовано фреймворк React Bootstrap, який пропонує готові компоненти, що гарантують належне відображення контенту на різноманітних пристроях – від смартфонів до настільних комп'ютерів;
- Серверна складова: Побудована з використанням платформи Node.js і фреймворку Express.js, ця частина функціонує на стороні сервера та відповідає за логіку роботи сайту, управління інформацією в базі даних MongoDB, гарантування безпеки та надання програмного інтерфейсу для взаємодії з клієнтською частиною. Express.js оптимізує створення API, обробку HTTP-запитів і керування маршрутизацією. Обмін даними реалізовано у форматі JSON через протоколи HTTP/HTTPS.

Для організації даних обрано MongoDB – документо-орієнтовану NOSQL систему керування базами даних, яка зберігає інформацію у гнучкому JSON-подібному форматі, що ідеально узгоджується з JavaScript та забезпечує високу швидкодію при обробці інформації. Інтеграція з MongoDB реалізована за допомогою бібліотеки Mongoose, що надає зручні інструменти для моделювання та маніпулювання даними.

Впроваджена клієнт-серверна архітектура забезпечує чіткий розподіл функціональних обов'язків між фронтендом і бекендом, що покращує структуру коду, спрощує процес розробки та підтримки, а також сприяє масштабованості проекту. React гарантує динамічний і відгукливий інтерфейс користувача, тоді

як зв'язка Node.js/Express.js формує потужну та гнучку основу для опрацювання запитів та управління даними. MongoDB, як NoSQL рішення, забезпечує гнучкість у зберіганні різноманітної інформації, пов'язаної з екологічними матеріалами, тестуванням та ігровою статистикою.

2.4 Проектування варіантів використання для вебсайта

Діаграма діяльності є важливим інструментом моделювання поведінки системи, який відображає послідовність дій та потоків управління у вебсайті EcoAware. Ця діаграма допомагає візуалізувати логіку роботи системи та взаємодію користувачів з основними функціональними модулями платформи, забезпечуючи розуміння всіх можливих шляхів навігації та обробки даних.

Діаграма діяльності для вебсайту EcoAware відображає чотири основні сценарії використання системи, які відповідають ключовим функціональним вимогам та охоплюють весь спектр можливостей платформи для підвищення екологічної свідомості користувачів.

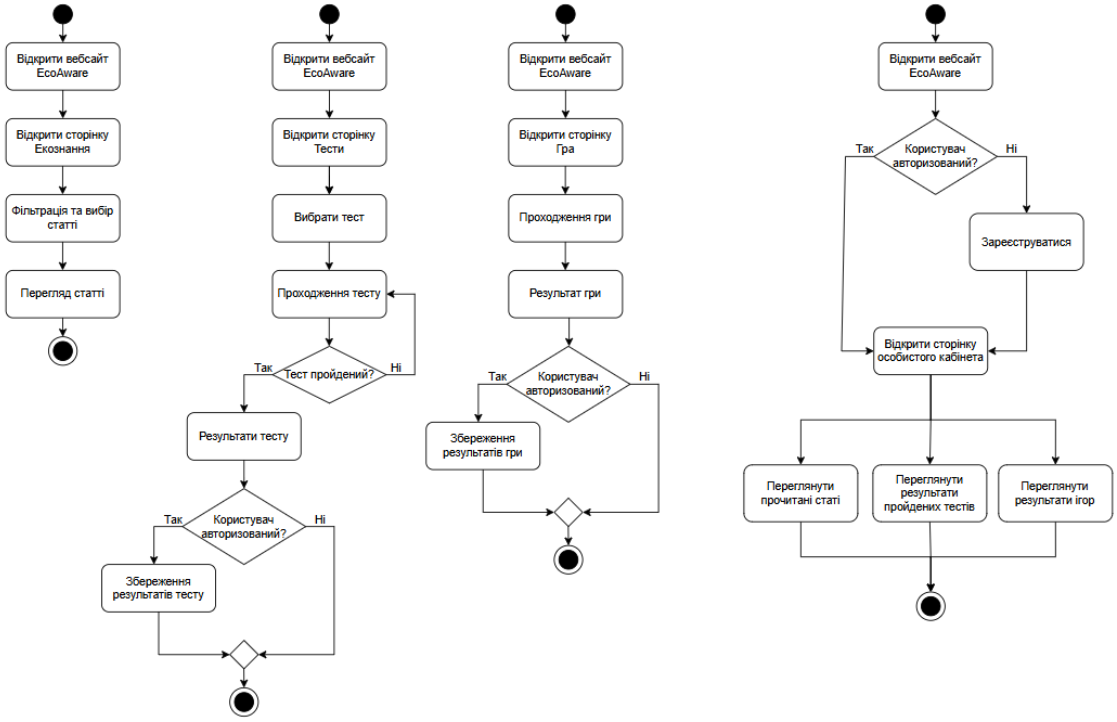


Рис. 2.2 Діаграма діяльності вебсайта EcoAware

На даній діаграмі такі випадки діяльності в сайті:

- Перегляд екологічних статей: Перша діаграма демонструє найпростіший сценарій взаємодії з платформою. Користувач відкриває вебсайт EcoAware, переходить на сторінку "Екознання", де має можливість фільтрувати та обирати статті за категоріями. Після вибору конкретної статті відбувається її детальний перегляд. Цей процес доступний для всіх користувачів без необхідності авторизації, що забезпечує максимальну відкритість освітнього контенту;
- Проходження екологічних тестів: Друга діаграма ілюструє інтерактивний процес тестування знань. Користувач обирає тест на спеціалізованій сторінці, проходить послідовність запитань з можливістю навігації між ними, після чого система автоматично обчислює результати. Ключовою особливістю є умовне збереження результатів - якщо користувач авторизований, його результати зберігаються в профілі для формування історії навчального прогресу, якщо ні - відображаються лише в поточній сесії;
- Участь в екологічній грі: Третя діаграма показує найбільш інтерактивний компонент сайта. Користувач переходить на сторінку "Гра", де бере участь в інтерактивних екологічних завданнях з динамічною підстройкою складності. Система надає миттєвий зворотний зв'язок та здійснює комплексну оцінку результатів. Аналогічно до тестів, збереження ігрових результатів та статистики відбувається лише для авторизованих користувачів, що стимулює створення облікового запису;
- Управління особистим кабінетом: Четверта діаграма охоплює найскладніший сценарій з обов'язковою авторизацією. Спочатку система перевіряє статус користувача - якщо він не авторизований, пропонується реєстрація або вхід в існуючий акаунт. Після успішної автентифікації користувач отримує доступ до персонального кабінету з трьома основними розділами: історією прочитаних статей з рекомендаціями, детальною статистикою результатів тестів з аналізом прогресу, та ігровими досягненнями з рейтингами і порівнянням з іншими користувачами.

2.5 Мапа вебсайта

Мапа вебсайта представляє ієрархічну структуру платформи EcoAware, що складається з головної сторінки та чотирьох основних розділів. Така організація забезпечує логічну навігацію та швидкий доступ до всіх функціональних можливостей сайту.

Рис. 2.3 Мапа вебсайта EcoAware

На даній мапі зображені такі сторінки сайта:

- Головна сторінка: Центральна точка входу до всіх розділів вебсайту. Містить загальну інформацію про платформу, основне навігаційне меню та блоки з рекомендованим контентом для швидкого доступу до популярних матеріалів;
- Особистий кабінет: Персоналізований розділ, доступний лише для авторизованих користувачів. Структурований у три підрозділи: перегляд прочитаних статей з історією взаємодії та можливістю швидкого повторного доступу, перегляд пройдених тестів з аналітикою результатів, та перегляд результатів гри зі статистикою ігрової активності;
- Сторінка статей: Освітній розділ з екологічним контентом, що включає головну сторінку розділу з каталогом статей та можливостями фільтрації за

категоріями, а також підсторінку перегляду статі для детального відображення обраного матеріалу;

- Сторінка тестів: Інтерактивний модуль оцінювання знань, який містить головну сторінку з каталогом доступних тестів та інформацією про тематику і складність, а також підсторінку проходження тесту з інтерфейсом для відповіді на запитання, навігацією між питаннями та отриманням детальних результатів;
- Сторінка Еко-гри: Ігровий модуль для закріплення знань, що складається з головної сторінки з описом ігрового процесу та правилами участі, і підсторінки таблиці рекордів з рейтингами.

2.6 Проектування інтерфейсу вебсайта

Архітектура інтерфейсу спроектована відповідно до функціональних вимог системи. Основний акцент зроблено на швидкому доступі до ключових розділів: екознання з можливістю пошуку та фільтрації статей, інтерактивні тести з системою відстеження результатів, та екологічна гра з рейтинговою системою. Навігація організована таким чином, щоб користувачі могли легко переходити між освітніми матеріалами та відстежувати свій прогрес у персональній статистиці.

Інтерфейс оптимізований для різних рівнів цифрової грамотності, забезпечуючи комфортне використання як для молоді, так і для дорослих користувачів, які цікавляться екологією. Мінімізація кількості кроків для виконання основних дій сприяє підвищенню залученості до освітнього процесу.

Для створення макетів використано платформу Figma, яка дозволила розробити комплексну систему візуальних компонентів та інтерактивних прототипів. Figma забезпечила можливість швидкого тестування навігаційних рішень та оптимізації користувацького досвіду на етапі проектування.

Розроблені прототипи стали основою для реалізації технічної архітектури сайту, включаючи систему реєстрації та авторизації користувачів з використанням JWT-шифрування, функціонал проходження тестів та участі в екологічній грі з відстеженням персональної статистики.

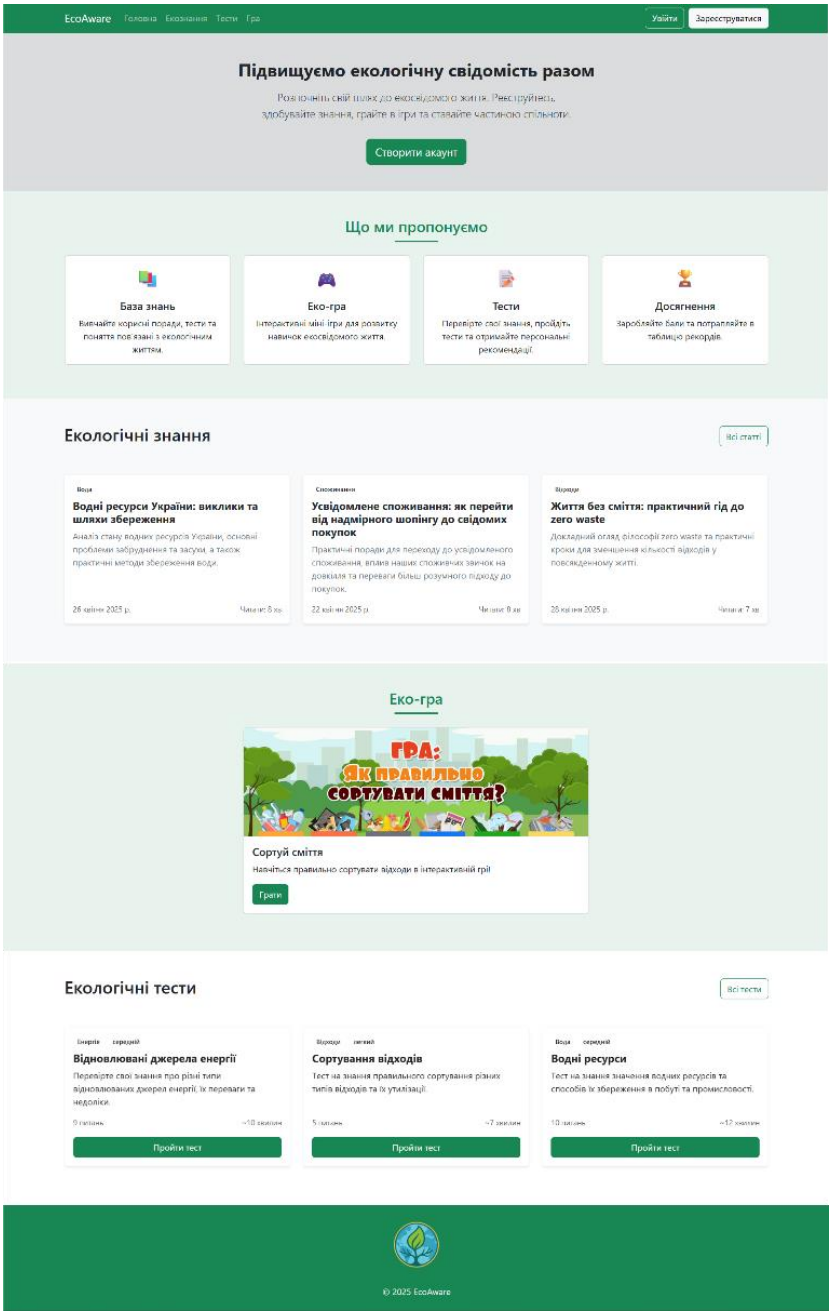


Рис. 2.4 Прототип головної сторінки сайта

Головна сторінка EcoAware, яка зображена на рисунку 2.4, містить основну навігацію з розділами "Екознання", "Тести", "Гра" та блок авторизації. Також тут

розміщені превью останніх екологічних статей, візуальний блок гри "Сортуй сміття" та каталог доступних тестів з кнопками швидкого доступу.

Структура сторінки забезпечує комплексний огляд всіх можливостей сайта з прямими посиланнями для негайного залучення користувачів.

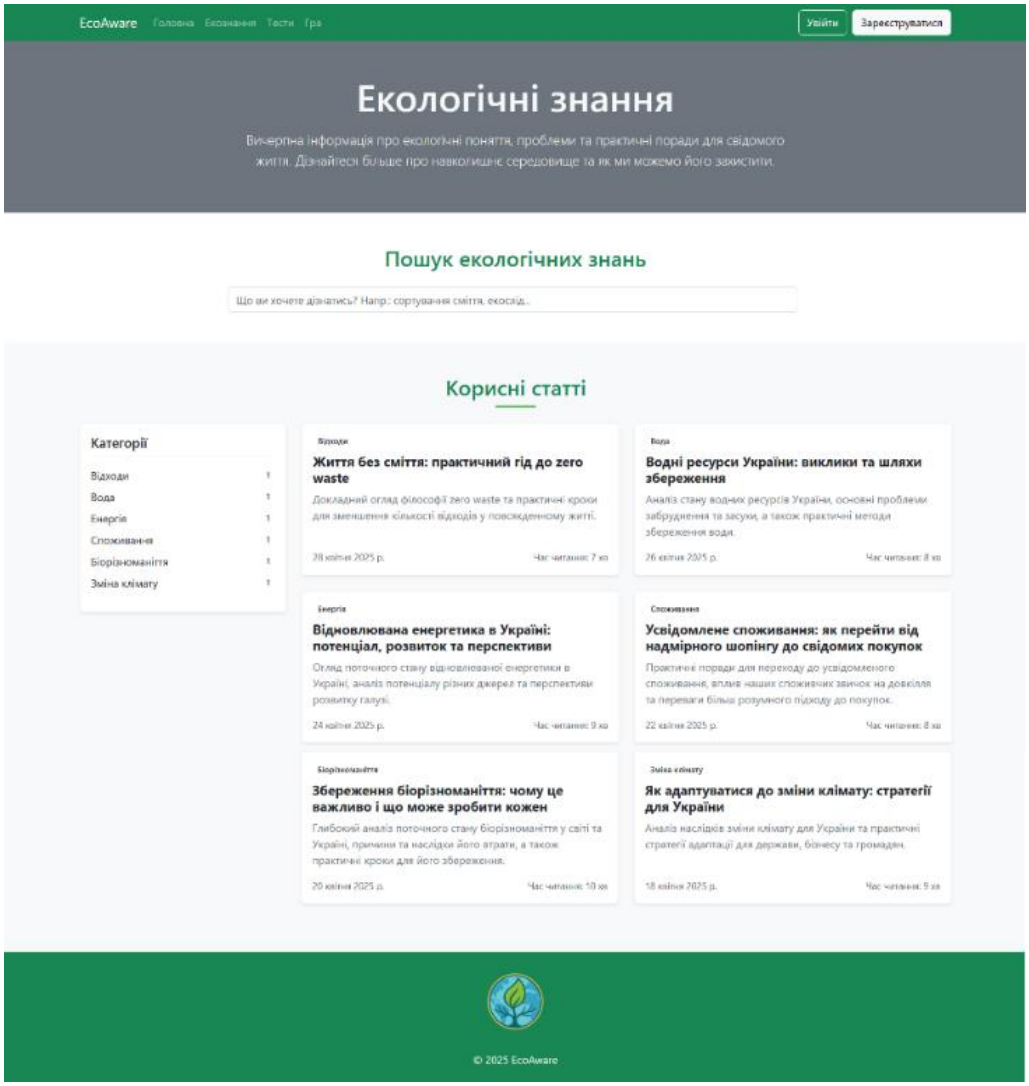


Рис. 2.5 Прототип сторінки з екологічними статтями

Сторінка "Екологічні знання", яка зображена на рисунку 2.5, містить заголовок з описом розділу та пошукове поле з підказкою для швидкого знаходження потрібних матеріалів. Ліворуч розміщено блок категорій з фільтрами. Основну частину займає сітка статей у форматі карток з заголовками, короткими описами, датами публікації та часом читання.

Структура сторінки забезпечує зручну навігацію по освітньому контенту з можливістю категоризації та швидкого пошуку релевантних матеріалів.

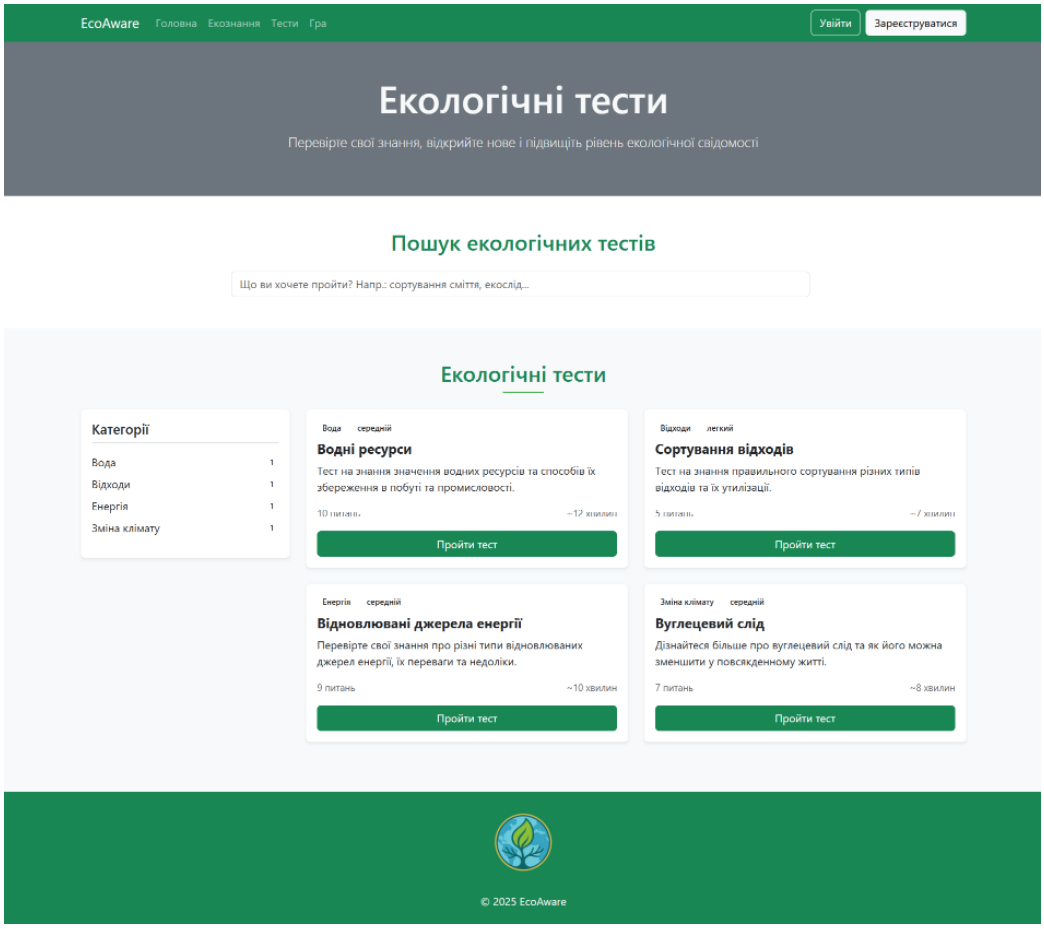


Рис. 2.6 Прототип сторінки з тестами

Сторінка "Екологічні тести", яка зображена на рисунку 2.6, містить заголовок з мотивуючим описом для перевірки знань та пошукове поле для швидкого знаходження потрібних тестів. Ліворуч розміщено блок категорій з фільтрами за тематиками.

Основну частину займає сітка тестів у форматі карток з назвами, короткими описами, позначками рівня складності, кількістю питань та орієнтовним часом проходження. Кожна картка містить кнопку "Пройти тест" для негайного початку тестування.

Структура сторінки забезпечує зручний вибір тестів різної складності з чіткою інформацією про тривалість та тематику для ефективного планування навчального процесу.

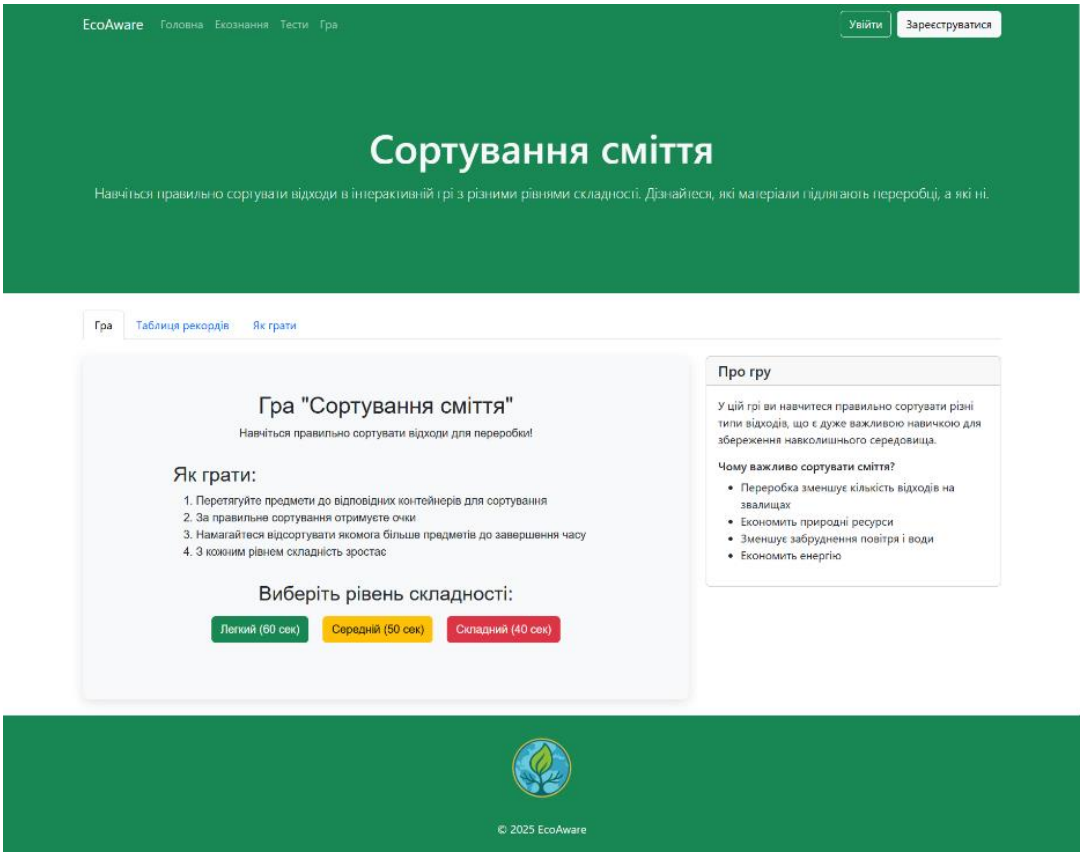


Рис. 2.7 Прототип сторінки з еко-грою

Сторінка гри "Сортування сміття", яка зображена на рисунку 2.7, містить заголовок з описом навчального процесу та навігаційні вкладки "Гра", "Таблиця рекордів" та "Як грати". Центральну частину займає блок з правилами гри, де пояснюється механіка сортування відходів та система нарахування очок.

Ліворуч розміщено детальні інструкції з чотирма кроками гри та вибір рівня складності. Праворуч знаходиться інформаційний блок "Про гру" з поясненням важливості сортування сміття та його екологічних переваг.

Структура сторінки забезпечує повне розуміння ігрового процесу перед початком гри з можливістю вибору рівня складності відповідно до навичок користувача.

3 ВИБІР ЗАСОБІВ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ ВЕБСАЙТА

Для реалізації вебсайту EcoAware було обрано сучасний технологічний стек, що забезпечує високу продуктивність, масштабованість та зручність розробки. Вибір засобів програмної реалізації базувався на аналізі функціональних вимог проекту, необхідності забезпечення інтерактивності інтерфейсу та потребі в надійному зберіганні користувацьких даних.

3.1 JavaScript

JavaScript - це високорівнева, інтерпретована мова програмування, яка спочатку була розроблена для веббраузерів, але зараз широко використовується як для клієнтської, так і для серверної розробки. JavaScript є динамічною мовою з гнучкою типізацією та підтримкою об'єктно-орієнтованого, функціонального та процедурного стилів програмування [8].

JavaScript обрано як основну мову програмування для розробки як клієнтської, так і серверної частини вебсайту EcoAware. Використання єдиної мови програмування для всього стеку технологій значно спрощує процес розробки, підтримки та масштабування додатку, дозволяючи розробникам ефективно працювати з різними частинами системи без необхідності перемикання між різними мовами.

JavaScript ES6+ надає сучасні можливості для написання чистого та ефективного коду. Підтримка стрілочних функцій дозволяє створювати більш лаконічний синтаксис для функцій та автоматично прив'язувати контекст this. Деструктуризація спрощує роботу з об'єктами та масивами, дозволяючи витягувати потрібні значення в одному рядку коду. Конструкції async/await забезпечують зручну роботу з асинхронним кодом, роблячи його більш читабельним та схожим на синхронний код.

Модульна система ES6 дозволяє створювати структурований та легко підтримуваний код через систему імпорту та експорту модулів. Це особливо важливо для проекту EcoAware, де різні функціональності (автентифікація, управління контентом, тестування, ігри) організовані в окремі модулі з чітко визначеними інтерфейсами взаємодії.

Асинхронна природа JavaScript ідеально підходить для обробки користувацьких взаємодій в реальному часі, що критично важливо для інтерактивних елементів вебсайту. Під час проходження екологічних тестів користувачі можуть отримувати миттєвий зворотний зв'язок, а в екологічній грі - динамічне оновлення результатів без затримок інтерфейсу.

Підтримка Promises та Event Loop в JavaScript забезпечує ефективну обробку множинних асинхронних операцій, таких як одночасні запити до бази даних для завантаження екологічних статей, збереження результатів тестів та оновлення ігрової статистики. Це гарантує швидкодію та відгукливість додатку навіть при високому навантаженні.

3.2 Фронтенд технології

3.2.1 React

React - це JavaScript бібліотека з відкритим вихідним кодом для створення користувацьких інтерфейсів, розроблена компанією Facebook (Meta). React базується на компонентному підході, де інтерфейс розбивається на незалежні, повторно використовувані частини, що можуть мати власний стан та логіку [9]. Основною особливістю React є використання віртуального DOM для оптимізації продуктивності.

React (версія 19.1.0) обрано як основну бібліотеку для створення користувацького інтерфейсу вебсайту EcoAware з кількох критично важливих причин. Компонентний підхід React дозволяє створювати повторно використовувані елементи інтерфейсу, що особливо важливо для освітньої платформи з різними типами контенту. Наприклад, картки екологічних статей,

елементи тестових питань та ігрові компоненти можуть бути реалізовані як незалежні компоненти з власною логікою та стилізацією.

Віртуальний DOM React є однією з ключових переваг для проекту EcoAware. Коли користувач взаємодіє з інтерактивними елементами (відповідає на питання тесту, грає в екологічну гру, фільтрує статті), React порівнює поточний стан віртуального DOM з попереднім станом та оновлює лише ті частини реального DOM, які справді змінилися. Це мінімізує кількість операцій з DOM та забезпечує швидке оновлення інтерфейсу, що критично важливо для інтерактивних освітніх додатків.

Hooks API React революціонізував підхід до управління станом та життєвим циклом компонентів. У контексті EcoAware useState hook використовується для управління локальним станом компонентів, таких як поточне питання в тесті, відповіді користувача або прогрес у грі. useEffect hook дозволяє виконувати побічні ефекти, такі як завантаження екологічних статей з сервера при монтуванні компонента або збереження результатів тесту при їх зміні.

Кастомні hooks дозволяють інкапсулювати складну логіку та повторно використовувати її в різних компонентах. Для EcoAware можуть бути створені спеціалізовані hooks, такі як useAuth для управління автентифікацією користувача, useQuiz для логіки тестування або useGameState для управління ігровим процесом.

Context API React забезпечує глобальне управління станом для спільних даних без необхідності передавати props через множинні рівні компонентів (prop drilling). У вебсайті EcoAware контекст використовується для зберігання інформації про авторизованого користувача, його прогресу в навчанні та налаштування інтерфейсу, що дозволяє будь-якому компоненту в дереві отримати доступ до цих даних.

React Developer Tools надають потужні можливості для відлагодження та оптимізації додатку під час розробки. Можливість інспектувати компоненти, їх

стан, props та виконувати профілювання продуктивності є критично важливою для забезпечення швидкодії освітньої платформи.

Екосистема React включає величезну кількість додаткових бібліотек та інструментів, що дозволяють легко інтегрувати додаткову функціональність. Для EcoAware це означає можливість легко додавати нові типи інтерактивного контенту, системи аналітики користувацької поведінки або інтеграцію з зовнішніми освітніми ресурсами.

3.2.2 React Router DOM

React Router DOM - це декларативна бібліотека маршрутизації для React додатків, яка дозволяє створювати Single Page Applications (SPA) з множинними "сторінками" без перезавантаження браузера. Бібліотека забезпечує синхронізацію URL з інтерфейсом додатку та управління історією браузера [10].

React Router DOM (версія 6.30.0) забезпечує клієнтську маршрутизацію для створення плавного користувацького досвіду в EcoAware. Користувачі можуть навігувати між розділами "Екознання", "Тести", "Гра" та особистим кабінетом без перезавантаження сторінки, що значно покращує швидкодію та зменшує споживання трафіку.

Декларативний підхід до маршрутизації спрощує управління навігацією та забезпечує підтримку історії браузера, дозволяючи користувачам використовувати кнопки "назад" та "вперед" для навігації по освітньому контенту.

3.2.3 React Bootstrap

React Bootstrap - це повна реімплементація Bootstrap компонентів для React, яка замінює Bootstrap JavaScript на React компоненти без залежностей від jQuery. Кожен компонент побудований з урахуванням доступності та найкращих практик React [11].

React Bootstrap (версія 2.10.9) надає готові React-компоненти, базовані на Bootstrap фреймворку, що прискорює розробку інтерфейсу EcoAware та забезпечує консистентний дизайн. Компоненти повністю інтегровані з React

екосистемою та підтримують всі переваги React, включаючи керування станом та обробку подій.

3.3 Бекенд технології

3.3.1 Express.js

Express.js - це мінімалістичний та гнучкий веб-фреймворк для Node.js, який надає надійний набір функцій для веб- та мобільних додатків. Express є де-факто стандартним сервер-фреймворком для Node.js та відомий своєю простотою та продуктивністю [12].

Express.js (версія 4.18.2) використовується як веб-фреймворк для створення серверної частини EcoAware. Middleware архітектура Express дозволяє легко інтегрувати автентифікацію, валідацію даних, логування та обробку помилок. Система маршрутизації забезпечує організацію API endpoints для управління екологічними статтями, тестами та ігровими даними.

3.3.2 Mongoose

Mongoose - це Object Document Mapper (ODM) для MongoDB та Node.js, який надає схему-базований підхід для моделювання даних додатку. Mongoose включає вбудовану типізацію, валідацію, побудову запитів, middleware та інші функції для спрощення роботи з MongoDB [13].

Mongoose (версія 7.5.0) використовується для роботи з MongoDB в EcoAware, надаючи структуровані схеми для користувачів, статей, тестів та ігрових даних. Валідація Mongoose забезпечує цілісність даних, а middleware функції дозволяють автоматично обробляти дані перед збереженням.

3.3.3 JSON Web Tokens

JSON Web Token (JWT) - це відкритий стандарт (RFC 7519) для безпечної передачі інформації між сторонами у вигляді JSON об'єкта. JWT використовується для автентифікації та авторизації, дозволяючи перевірити цілісність та автентичність переданої інформації [14].

JWT використовується для автентифікації користувачів EcoAware, забезпечуючи stateless автентифікацію без необхідності зберігання сесій на сервері. Це дозволяє масштабувати додаток та забезпечує безпечний доступ до персоналізованих функцій.

3.3.4 Node.js

Node.js - це відкрите JavaScript runtime середовище, побудоване на потужному V8 JavaScript engine від Google Chrome. Node.js дозволяє виконувати JavaScript код на сервері, поза межами веб-браузера, що революціонізувало підхід до серверної розробки. Платформа використовує подієво-орієнтовану, неблокуючу I/O модель, що робить її надзвичайно ефективною для створення масштабованих мережевих додатків [15].

Архітектура Node.js базується на концепції Event Loop - механізмі, який дозволяє обробляти тисячі одночасних з'єднань без створення окремих потоків для кожного запиту[16]. Замість блокування виконання програми під час очікування результатів I/O операцій (таких як читання файлів або запити до бази даних), Node.js продовжує обробляти інші завдання, а коли I/O операція завершується, відповідний callback викликається асинхронно.

Для проекту EcoAware Node.js обрано як серверну платформу з кількох критично важливих причин. По-перше, асинхронна природа Node.js ідеально підходить для обробки множинних одночасних запитів від користувачів, які можуть одночасно читати екологічні статті, проходити тести та грати в ігри. Event-driven архітектура забезпечує високу продуктивність навіть при значному навантаженні, що важливо для освітньої платформи, яка може обслуговувати багато користувачів одночасно.

По-друге, використання JavaScript як на клієнті, так і на сервері, значно спрощує розробку та підтримку коду. Розробники можуть використовувати одні й ті ж структури даних, бібліотеки валідації та логіку як для фронтенду, так і для бекенду EcoAware. Це особливо важливо для функцій валідації даних тестів, де одна й та ж логіка перевірки відповідей може використовуватися як на клієнті для миттєвого зворотного зв'язку, так і на сервері для остаточної верифікації.

NPM (Node Package Manager) екосистема надає доступ до понад мільйона готових пакетів, що значно прискорює розробку EcoAware. Для проекту можуть бути легко інтегровані пакети для роботи з зображеннями в статтях, генерації PDF звітів з результатами тестів, відправки email сповіщень користувачам, або інтеграції з зовнішніми API для отримання актуальних екологічних даних.

Модульна система Node.js дозволяє організувати код EcoAware у логічні блоки: окремі модулі для автентифікації користувачів, управління екологічними статтями, обробки тестових результатів та ігрової логіки. Це забезпечує чистоту архітектури та легкість підтримки коду.

Вбудована підтримка JSON в Node.js ідеально інтегрується з MongoDB та React, створюючи seamless потік даних від бази даних через API до користувацького інтерфейсу. Екологічні статті, тестові питання та ігрові дані можуть легко передаватися між різними рівнями додатку без складних перетворень формату.

Node.js також надає відмінні можливості для real-time комунікації через WebSockets, що може бути використано в EcoAware для створення інтерактивних ігрових елементів, live чатів під час проходження тестів, або real-time оновлення лідерборду екологічної гри.

3.3.5 MongoDB

MongoDB - це документо-орієнтована NoSQL система управління базами даних, яка зберігає дані у BSON (Binary JSON) форматі [17, 18]. MongoDB надає високу продуктивність, доступність та легке масштабування.

MongoDB забезпечує гнучке зберігання різноманітних типів даних EcoAware без жорсткої схеми, що дозволяє легко адаптувати структуру даних при розвитку проекту.

3.4 Інструменти розробки

3.4.1 Visual Studio Code

Visual Studio Code (VS Code) - це безкоштовний редактор вихідного коду, розроблений Microsoft для Windows, Linux та macOS. VS Code підтримує

налагодження, вбудований Git контроль, підсвічування синтаксису, інтелектуальне автодоповнення коду, рефакторинг та вбудований термінал. Редактор має розширену систему плагінів, що дозволяє налаштувати його під конкретні потреби розробки [19].

Visual Studio Code обрано як основне середовище розробки для проекту EcoAware завдяки його відмінній підтримці JavaScript та React екосистеми. Вбудований IntelliSense надає розумне автодоповнення коду, що значно прискорює написання компонентів React та серверної логіки Node.js.

Інтегрована підтримка Git дозволяє ефективно управляти версіями коду EcoAware, відстежувати зміни в різних компонентах проекту та здійснювати commit операції безпосередньо з редактора. Вбудований термінал забезпечує зручний доступ до npm команд для запуску сервера розробки, встановлення залежностей та виконання тестів.

Система розширень VS Code дозволяє інтегрувати спеціалізовані інструменти для React розробки, такі як ES7+ React/Redux/React-Native snippets для швидкого створення компонентів, Prettier для автоматичного форматування коду та ESLint для перевірки якості коду. Розширення для роботи з MongoDB надає зручний інтерфейс для перегляду та редагування даних безпосередньо з редактора.

3.4.2 Figma

Figma - це хмарний інструмент для дизайну інтерфейсів та прототипування, який дозволяє командам співпрацювати в реальному часі над створенням дизайнів. Figma підтримує векторну графіку та надає інструменти для створення інтерактивних прототипів [20].

Figma використовувалася для створення всіх макетів та прототипів інтерфейсу EcoAware, дозволивши спроектувати зручний та привабливий дизайн для ефективної екологічної освіти.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБСАЙТУ

4.1 Структура проєкту

Організація структури проєкту є фундаментальним аспектом розробки будь-якого вебзастосунку. Чітка та логічна структура не лише полегшає навігацію кодовою базою та розуміння принципів її роботи, але і сприяє підтримці та масштабуванню застосунку. Представлений вебсайт EcoAware розроблено за архітектурою full-stack, де клієнтська частина реалізована за допомогою бібліотеки React, а серверна частина – на базі середовища виконання Node.js з використанням фреймворку Express. Для зберігання та керування даними використовується NoSQL база даних MongoDB

Основна структура проєкту EcoAware розділена на дві головні частини: клієнтську (client) та серверну (backend), кожна з яких має свою внутрішню організацію, оптимізовану для відповідних технологій та завдань екологічної освітньої платформи.

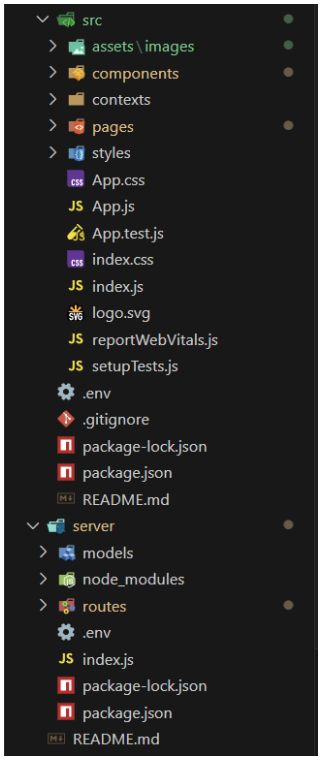


Рис. 4.1 Структура проекту EcoAware

4.2 Розробка серверної частини

Серверна частина вебсайту EcoAware реалізована на базі Node.js з використанням фреймворку Express.js та побудована за принципами RESTful API архітектури. Бекенд забезпечує надійну обробку запитів від клієнтської частини, управління даними в MongoDB та реалізацію бізнес-логіки для всіх функціональних модулів екологічної платформи.

Головний файл index.js служить точкою входу для серверного додатку та виконує ключові функції ініціалізації системи. Спочатку підключаються всі необхідні модулі: Express для створення вебсервера, CORS для налаштування міждоменних запитів, Mongoose для роботи з MongoDB, cookie-parser для обробки cookies та jsonwebtoken для автентифікації користувачів.

```
const express = require('express');
const cors = require('cors');
const mongoose = require('mongoose');
const cookieParser = require('cookie-parser');
const jwt = require('jsonwebtoken');

require('dotenv').config();
require('./models/User');
require('./models/Article');
require('./models/Test');
require('./models/ViewedArticle');
require('./models/CompletedTest');

const app = express();
const PORT = process.env.PORT || 5000;

// Налаштування CORS для доступу з frontend
app.use(cors({
  origin: process.env.CLIENT_URL || 'http://localhost:3000',
  credentials: true,
  methods: ['GET', 'POST', 'PUT', 'DELETE', 'OPTIONS'],
  allowedHeaders: ['Content-Type', 'Authorization', 'Accept', 'X-User-ID']
}));

app.use(express.json());
app.use(cookieParser());

// Middleware для декодування JWT токена та створення req.user
app.use((req, res, next) => {
  try {
    // Перевірка наявності токена в cookies
    const token = req.cookies.token;

    if (token) {
      // Декодування токена
      const decoded = jwt.verify(token, process.env.JWT_SECRET || 'ecoaware_secure_jwt_token_2024');

      // Встановлення об'єкта користувача в запит
      req.user = decoded;

    } } catch (error) {
    console.error('Помилка при декодуванні токена:', error);
  }

  // Продовження обробки запиту
  next();
});

// MongoDB Connection
mongoose.connect(process.env.MONGODB_URI || 'mongodb://localhost:27017/ecoaware', {
  useNewUrlParser: true,
  useUnifiedTopology: true,
})
.catch(err => console.error('MongoDB connection error:', err));
```

Рис. 4.2 Ініціалізація серверної частини та конфігурація middleware

Файл index.js є точкою входу серверного додатку. Імпортуються основні модулі: Express, CORS, Mongoose, cookie-parser, jsonwebtoken, dotenv та моделі бази даних.

Створюється Express додаток з портом 5000. Налаштовується CORS для взаємодії з фронтендом, включаючи підтримку credentials та дозволені HTTP методи (GET, POST, PUT, DELETE, OPTIONS).

Підключаються middleware: JSON parser для обробки запитів, cookie-parser для роботи з cookies, та JWT middleware для автоматичної перевірки токенів з cookies і встановлення req.user.

Підключення до MongoDB через Mongoose з використанням змінних середовища для URL бази даних та JWT секретного ключа.

4.2.1 Система автентифікації та авторизації

Модуль автентифікації реалізований в файлі auth.js та забезпечує повний цикл управління користувачами.

Процес реєстрації, який зображений на рисунку 4.3, включає валідацію даних через express-validator: ім'я користувача (3-30 символів, унікальність), email (формат та унікальність, конвертація в нижній регістр), пароль (мінімум 6 символів). При успішній валідації створюється користувач з автоматичним хешуванням пароля через bcrypt в pre-save hook.

Після збереження генерується JWT токен з ідентифікатором, ім'ям та роллю користувача, термін дії 7 днів. Токен встановлюється в HTTP-only cookie з налаштуваннями безпеки: httpOnly, secure для продакшену, maxAge та sameSite: 'strict' для захисту від CSRF.

```
router.post('/register', [
  // Валідація даних
  body('username')
    .trim()
    .isLength({ min: 3, max: 30 })
    .withMessage('Ім\'я користувача повинно містити від 3 до 30 символів')
    .custom(async value => {
      const user = await User.findOne({ username: value });
      if (user) {
        throw new Error('Користувач з таким ім\'ям вже існує');
      }
      return true;
    }),
  body('email')
    .trim()
    .isEmail()
    .withMessage('Введіть коректну електронну адресу')
    .custom(async value => {
      const user = await User.findOne({ email: value });
      if (user) {
        throw new Error('Користувач з такою електронною адресою вже існує');
      }
      return true;
    }),
  body('password')
    .isLength({ min: 6 })
    .withMessage('Пароль повинен містити мінімум 6 символів'),
  body('confirmPassword')
    .custom((value, { req }) => {
      if (value !== req.body.password) {
        throw new Error('Паролі не співпадають');
      }
      return true;
    })
], async (req, res) => {
  // Перевірка результатів валідації
  const errors = validationResult(req);
  if (!errors.isEmpty()) {
    return res.status(400).json({
      success: false,
      errors: errors.array()
    });
  }

  try {
    const { username, email, password } = req.body;

    // Створення нового користувача (без firstName і lastName)
    const newUser = new User({
      username,
```

Рис. 4.3 Реалізація маршруту реєстрації користувачів з валідацією даних

Авторизація, яка зображена на рисунку 4.4, валідує обов'язкові поля (ім'я/email та пароль) з підтримкою входу через MongoDB оператор $or як за ім'ям користувача, так і за електронною поштою.

Пошук користувача виконується за співпадінням введених даних з полями username або email в базі даних. При відсутності користувача повертається загальне повідомлення про неправильні облікові дані без вказівки конкретної причини для безпеки.

Перевірка пароля здійснюється через метод comparePassword моделі User, який використовує bcrypt для порівняння введеного пароля з хешованим значенням в базі даних.

При успішній автентифікації оновлюється поле lastLogin користувача для відстеження активності. Створюється новий JWT токен з актуальною інформацією та встановлюється в cookie.

Відповідь включає статус успіху, повідомлення та дані користувача для оновлення стану фронтенду.

```
router.post('/login', [
  // Валідація даних
  body('username').trim().not().isEmpty().withMessage('Введіть ім\'я користувача'),
  body('password').not().isEmpty().withMessage('Введіть пароль')
], async (req, res) => {
  // Перевірка результатів валідації
  const errors = validationResult(req);
  if (!errors.isEmpty()) {
    return res.status(400).json({
      success: false,
      errors: errors.array()
    });
  }

  try {
    const { username, password } = req.body;

    // Пошук користувача в базі даних
    const user = await User.findOne({
      $or: [{ username }, { email: username }]
    });

    if (!user) {
      return res.status(401).json({
        success: false,
        message: 'Неправильний логін або пароль'
      });
    }

    // Перевірка пароля
    const isMatch = await user.comparePassword(password);

    if (!isMatch) {
      return res.status(401).json({
        success: false,
        message: 'Неправильний логін або пароль'
      });
    }

    // Оновлення часу останнього входу
    user.lastLogin = Date.now();
    await user.save();

    // Створення JWT токена
    const token = jwt.sign(
      { id: user._id, username: user.username, role: user.role },
      process.env.JWT_SECRET || 'ecoaware_secure_jwt_token_2024',
      { expiresIn: '7d' }
    );
```

Рис. 4.4 Реалізація маршруту авторизації користувачів

4.2.2 Управління екологічним контентом

Маршрути для роботи з екологічними статтями реалізовані в файлі articles.js та забезпечують повний CRUD функціонал для освітнього контенту. Система управління статтями є одним з ключових компонентів платформи EcoAware, оскільки саме через неї користувачі отримують доступ до структурованих екологічних знань.

```
router.get('/', async (req, res) => {
  try {
    const articles = await Article.find().sort({ date: -1 });
    res.json(articles);
  } catch (err) {
    console.error('Error fetching articles:', err);
    res.status(500).json({ message: 'Server error' });
  }
});

// Important: Define the /viewed endpoint BEFORE /:id to prevent route conflicts
router.get('/viewed', requireAuth, async (req, res) => {
  try {

    // Check if ViewedArticle model is registered
    if (!mongoose.modelNames().includes('ViewedArticle')) {
      console.error('ViewedArticle model is not registered!');
      return res.status(500).json({
        success: false,
        message: 'Server error: Model not registered',
        error: 'ViewedArticle model is not registered'
      });
    }

    // Get viewed articles for the user, sorted by last viewed date
    const viewedArticles = await ViewedArticle.find({ userId: req.user.id })
      .sort({ lastViewedAt: -1 });

    res.status(200).json({
      success: true,
      articles: viewedArticles
    });
  } catch (error) {
    console.error('Error fetching viewed articles:', error);
    res.status(500).json({
      success: false,
      message: 'Server error while fetching viewed articles',
      error: error.toString(),
      stack: error.stack
    });
  }
});

// Get article by ID
router.get('/:id', async (req, res) => {
  try {
    const article = await Article.findById(req.params.id);

    if (!article) {
      return res.status(404).json({ message: 'Article not found' });
    }
```

Рис. 4.5 Маршрути для роботи з екологічними статтями

4.2.3 Система тестування знань

Функціональність тестування реалізована в маршрутах tests.js та забезпечує інтерактивну перевірку екологічних знань користувачів. Модель Test містить питання з варіантами відповідей, пояснення та налаштування складності тестів.

```
116  router.get('/:id', async (req, res) => {
117    try {
118      const test = await Test.findById(req.params.id);
119
120      if (!test) {
121        return res.status(404).json({ message: 'Test not found' });
122      }
123
124      res.json(test);
125    } catch (err) {
126      console.error('Error fetching test:', err);
127      res.status(500).json({ message: 'Server error' });
128    }
129  });
130
131  // Start test - remove correct answers for security
132  router.get('/:id/start', async (req, res) => {
133    try {
134      const test = await Test.findById(req.params.id);
135
136      if (!test) {
137        return res.status(404).json({ message: 'Test not found' });
138      }
139
140      // Create a copy of the test without correct answers
141      const testForUser = JSON.parse(JSON.stringify(test));
142
143      // Remove isCorrect flag from each option to prevent cheating
144      testForUser.questions.forEach(question => {
145        question.options.forEach(option => {
146          delete option.isCorrect;
147        });
148      });
149
150      res.json(testForUser);
151    } catch (err) {
152      console.error('Error preparing test:', err);
153      res.status(500).json({ message: 'Server error' });
154    }
155  });
156
157  // Check test answers
158  router.post('/:id/check', async (req, res) => {
159    try {
160      const { answers } = req.body;
161      const test = await Test.findById(req.params.id);
162
163      if (!test) {
164        return res.status(404).json({ message: 'Test not found' });
165      }
```

Рис. 4.6 Маршрути системи тестування знань

4.2.4 Ігровий модуль

Маршрути для екологічної гри реалізовані в файлі games.js та забезпечують функціонал гейміфікації навчального процесу. Основна гра "Сортування сміття" дозволяє користувачам практикувати навички правильного розділення відходів.

Маршрут для збереження результатів гри "Сортування сміття" перевіряє авторизацію користувача - анонімні користувачі не можуть зберігати результати. Вхідні дані включають рівень складності, очки, правильні/неправильні відповіді, загальну кількість спроб та час гри.

Ідентифікатор користувача конвертується в ObjectId для MongoDB. Числові поля обробляються через parseInt() зі значеннями за замовчуванням через оператор ||. Тип гри встановлюється як 'trash-sorting', точність обчислюється як відсоток правильних відповідей.

Створюється новий екземпляр GameResult та зберігається в базі даних з поверненням статусу 201 при успішному збереженні.

```
// Маршрут для збереження результатів гри "Сортування сміття"
router.post('/trash-sorting/results', async (req, res) => {
  try {
    const { level, score, correct, incorrect, total, playTime } = req.body;

    // Перевірка чи користувач авторизований
    if (!req.user) {
      return res.status(401).json({
        success: false,
        message: 'Для збереження результатів необхідно авторизуватися'
      });
    }

    // Отримання ID користувача з req.user (встановлюється middleware авторизації)
    const userId = req.user.id;

    // Створення ObjectId для userId
    let userObjectId;
    try {
      userObjectId = new ObjectId(userId);
    } catch (objIdError) {
      console.error('Помилка при створенні ObjectId:', objIdError);
      return res.status(400).json({
        success: false,
        message: 'Неправильний формат ID користувача'
      });
    }

    // Підготовка даних з правильними типами
    const gameData = {
      gameType: 'trash-sorting',
      userId: userObjectId,
      level: parseInt(level) || 1,
      score: parseInt(score) || 0,
      correct: parseInt(correct) || 0,
      incorrect: parseInt(incorrect) || 0,
      total: parseInt(total) || 0,
      playTime: parseInt(playTime) || 0
    };

    // Обчислюємо accuracy (якщо не буде обчислено в моделі)
    if (gameData.total > 0) {
      gameData.accuracy = Math.round((gameData.correct / gameData.total) * 100);
    } else {
      gameData.accuracy = 0;
    }

    // Створення нового запису результату гри
    const gameResult = new GameResult(gameData);

    // Зберігаємо результат
```

Рис. 4.7 Маршрут збереження результатів екологічної гри

4.3 Розробка клієнтської частини

Клієнтська частина вебсайту EcoAware розроблена як Single Page Application (SPA) на базі React.js та реалізує сучасний, інтуїтивний користувацький інтерфейс для ефективного взаємодії з екологічним контентом. Фронтенд забезпечує повну функціональність освітньої платформи через компонентну архітектуру та централізоване управління станом.

Головний компонент App.js, який зображений на рисунку 4.8, визначає структуру всього додатку та налаштовує клієнтську маршрутизацію через React Router DOM. Компонент обгортає весь додаток в AuthProvider для забезпечення глобального управління автентифікацією та BrowserRouter для SPA навігації.

```
import { BrowserRouter as Router, Routes, Route, Navigate } from 'react-router-dom';
import NavBar from './components/Navbar';
import Home from './pages/Home';
import EcoKnowledge from './pages/EcoKnowledge';
import ArticleDetail from './pages/ArticleDetail';
import Tests from './pages/Tests';
import TestDetail from './pages/TestDetail';
import TrashSortingGamePage from './pages/TrashSortingGame';
import Login from './pages/Login';
import Register from './pages/Register';
import Profile from './pages/Profile';
import Footer from './components/Footer';

// Компонент контексту для авторизації
import { AuthProvider, useAuth } from './contexts/AuthContext';

// Захищений маршрут
Windsurf: Refactor | Explain | ×
const ProtectedRoute = ({ children }) => {
  const { isAuthenticated, loading } = useAuth();

  if (loading) {
    // Показуємо компонент завантаження, поки перевіряємо статус авторизації
    return <div className="text-center py-5">Завантаження...</div>;
  }

  if (!isAuthenticated) {
    // Перенаправлення на сторінку входу з шляхом повернення
    const currentPath = window.location.pathname;
    return <Navigate to={`/login?redirect=${currentPath}`} replace />;
  }

  return children;
};

Windsurf: Refactor | Explain | Generate JSDoc | ×
const AppRoutes = () => {
  return (
    <Routes>
      <Route path="/" element={<Home />} />
      <Route path="/knowledge" element={<EcoKnowledge />} />
      <Route path="/knowledge/article/:id" element={<ArticleDetail />} />
      <Route path="/tests" element={<Tests />} />
      <Route path="/tests/:id" element={<TestDetail />} />
      <Route path="/games/trash-sorting" element={<TrashSortingGamePage />} />
      <Route path="/login" element={<Login />} />
      <Route path="/register" element={<Register />} />

      {/* Захищений маршрут до профілю */}
      <Route
        path="/profile"
```

Рис. 4.8 Налаштування маршрутизації у App.js

Система маршрутизації організована через компонент AppRoutes, який визначає всі доступні шляхи додатку: головна сторінка, розділи екологічних знань, тестів, ігор та особистого кабінету. Реалізовано захищений маршрут ProtectedRoute, який перевіряє статус автентифікації користувача та перенаправляє неавторизованих користувачів на сторінку входу з збереженням оригінального шляху для подальшого повернення.

Глобальна навігація забезпечується компонентом NavBar, який адаптується залежно від статусу авторизації користувача. Для авторизованих користувачів відображається dropdown меню з доступом до профілю та функцією виходу, для неавторизованих - кнопки входу та реєстрації.

Автентифікація реалізована через React Context API в файлі AuthContext.js, який зображений на рисунку 4.9, забезпечує централізоване управління станом користувача по всьому додатку. Контекст надає функції для реєстрації, входу, виходу та перевірки статусу автентифікації.

```
const Register = () => {
  const [formData, setFormData] = useState({
    username: '',
    email: '',
    password: '',
    confirmPassword: ''
  });
  const [loading, setLoading] = useState(false);
  const [errors, setErrors] = useState([]);

  const navigate = useNavigate();
  const location = useLocation();
  const { register } = useAuth();

  // Отримання параметра redirect з URL, якщо він є
  const redirectPath = new URLSearchParams(location.search).get('redirect') || '/';

  Windsurf: Refactor | Explain | Generate JSDoc | ×
  const handleChange = (e) => {
    const { name, value } = e.target;
    setFormData(prev => ({ ...prev, [name]: value }));
  };

  Windsurf: Refactor | Explain | Generate JSDoc | ×
  const handleSubmit = async (e) => {
    e.preventDefault();
    setLoading(true);
    setErrors([]);

    // Клієнтська валідація
    const validationErrors = [];
    if (formData.password !== formData.confirmPassword) {
      validationErrors.push({ msg: 'Паролі не співпадають' });
    }

    if (validationErrors.length > 0) {
      setErrors(validationErrors);
      setLoading(false);
      return;
    }

    try {
      // Використання функції register з контексту авторизації
      await register(formData);

      // Перенаправлення на сторінку з редіректом або головну
      navigate(redirectPath);
    } catch (err) {
      console.error('Помилка реєстрації:', err);
      setErrors([{ msg: err.message || 'Помилка реєстрації. Спробуйте ще раз.' }]);
    } finally {
      setLoading(false);
```

Рис. 4.9 Компонент реєстрації користувача

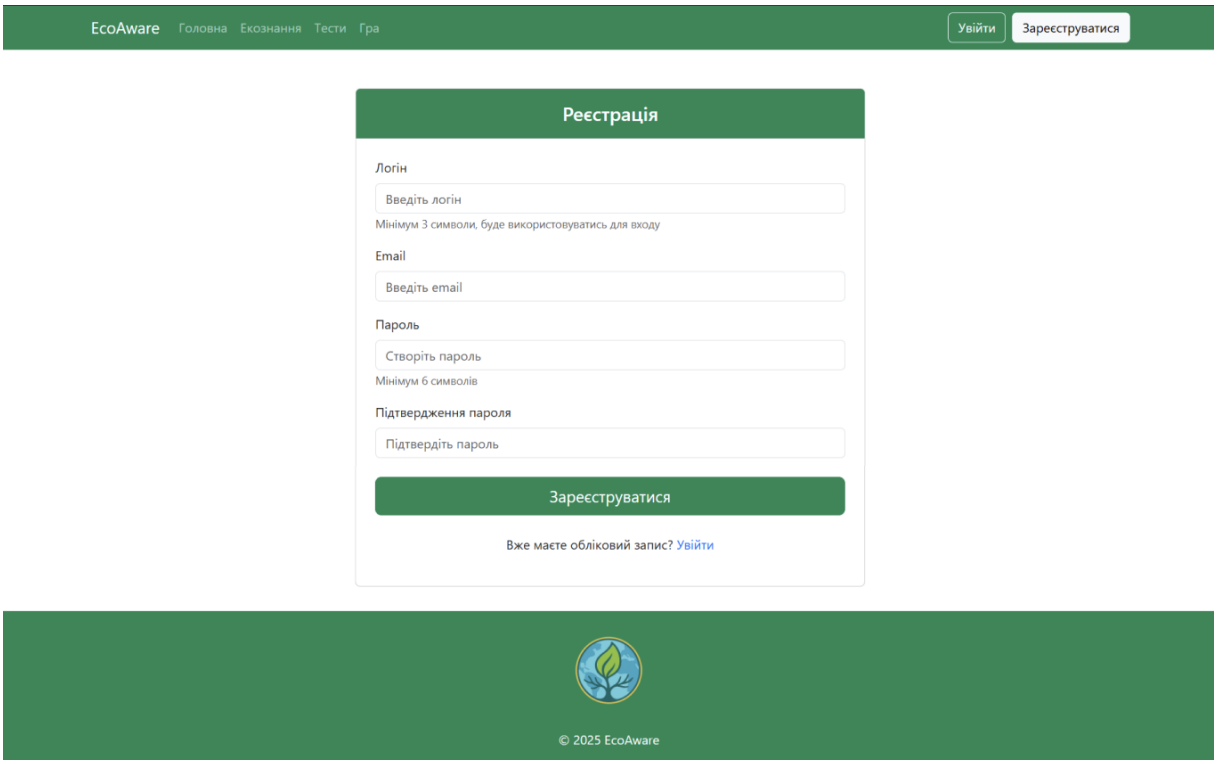


Рис. 4.10 Сторінка реєстрації користувача

Якщо користувач вже має обліковий запис, він може увійти на сайт через сторінку авторизації, яка зображена на рисунку 4.11. Подібно до сторінки реєстрації, яка зображена на рисунку 4.10, тут є форма для введення облікових даних. Після успішного входу користувач перенаправляється на головну сторінку або на попередньо збережений шлях.

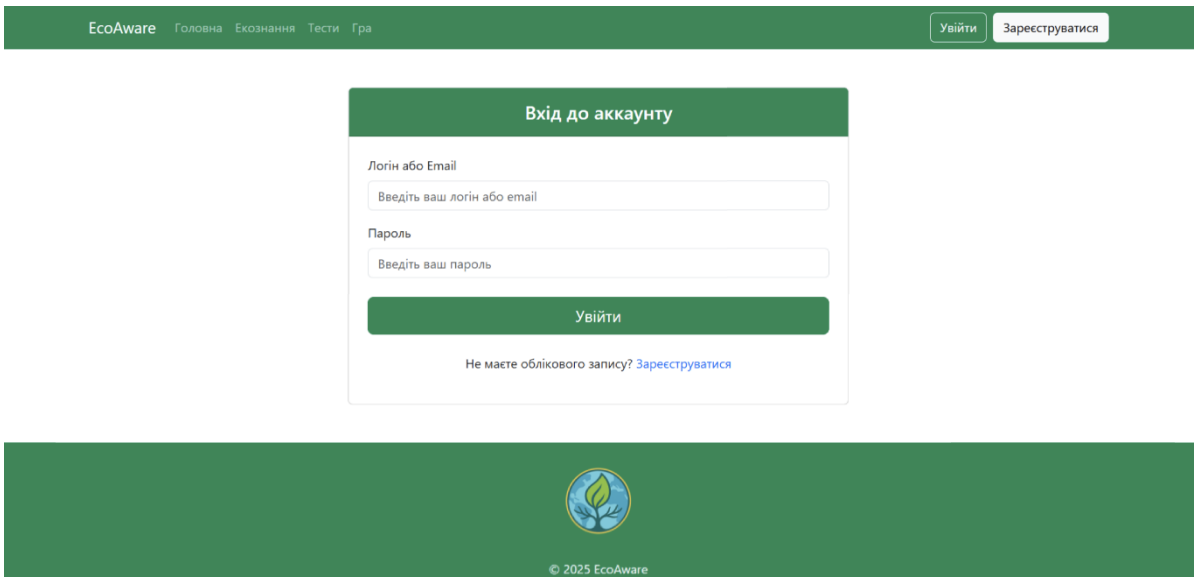


Рис. 4.11 Сторінка авторизації користувача

Після успішної автентифікації або при відвідуванні сайту користувач потрапляє на головну сторінку сайта, яка зображена на рисунку 4.12. Ця сторінка, представлена компонентом Home.js, є своєрідним центром контенту. Тут відображаються основні розділи платформи: екологічні знання, тести та ігри.

Home.js відображає контент у вигляді секцій, забезпечуючи зручний перегляд. Кожна секція містить прев'ю відповідного контенту. З цієї сторінки користувач може перейти до детального перегляду конкретних статей, тестів або ігор, натиснувши на відповідні картки.

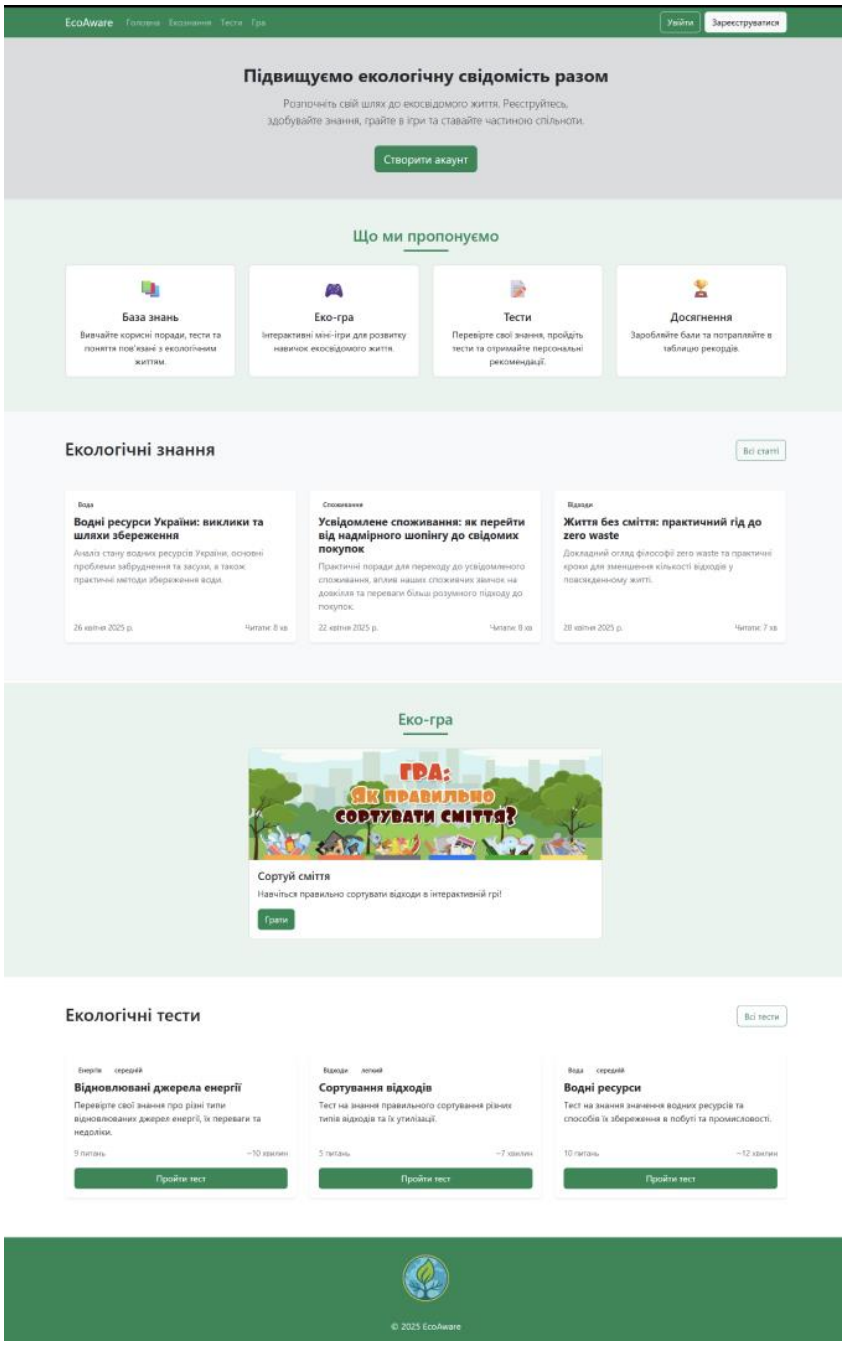


Рис. 4.12 - Головна сторінка

Сторінка "Екознання",яка зображена на рисунку 4.13, надає користувачам доступ до колекції екологічних статей. Компонент EcoKnowledge.js, який зображений на рисунку 4.14 забезпечує перегляд каталогу статей з можливостями пошуку та фільтрації за категоріями.

Рис. 4.13 Сторінка з екологічними статтями

```
Windsurf: Refactor | Explain | Generate JSDoc | X
const EcoKnowledgePage = () => {
  const [articles, setArticles] = useState([]);
  const [categories, setCategories] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);
  const [searchTerm, setSearchTerm] = useState('');
  const [selectedCategory, setSelectedCategory] = useState(null);

  useEffect(() => {
    Windsurf: Refactor | Explain | Generate JSDoc | X
    const fetchArticles = async () => {
      try {
        setLoading(true);

        const articlesResponse = await fetch('http://localhost:5000/api/articles');

        if (!articlesResponse.ok) {
          throw new Error('Failed to fetch articles');
        }

        const articlesData = await articlesResponse.json();
        setArticles(articlesData);

        const categoryMap = articlesData.reduce((acc, article) => {
          const category = article.category;
          acc[category] = (acc[category] || 0) + 1;
          return acc;
        }, {});

        const categoryArray = Object.entries(categoryMap).map(([name, count]) => ({
          name,
          count
        }));

        setCategories(categoryArray);
      } catch (err) {
        setError(err.message);
      } finally {
        setLoading(false);
      }
    };

    fetchArticles();
  }, []);
```

Рис. 4.14 Компонент відображення екологічних статей

Коли користувач вибирає конкретну статтю на сторінці "Екознання", відкривається сторінка детального перегляду, яка зображена на рисунку 4.15. Компонент ArticleDetail.js відповідає за відображення повного змісту статті, включаючи заголовок, текст з markdown форматуванням, інформацію про категорію, дату публікації та час читання.

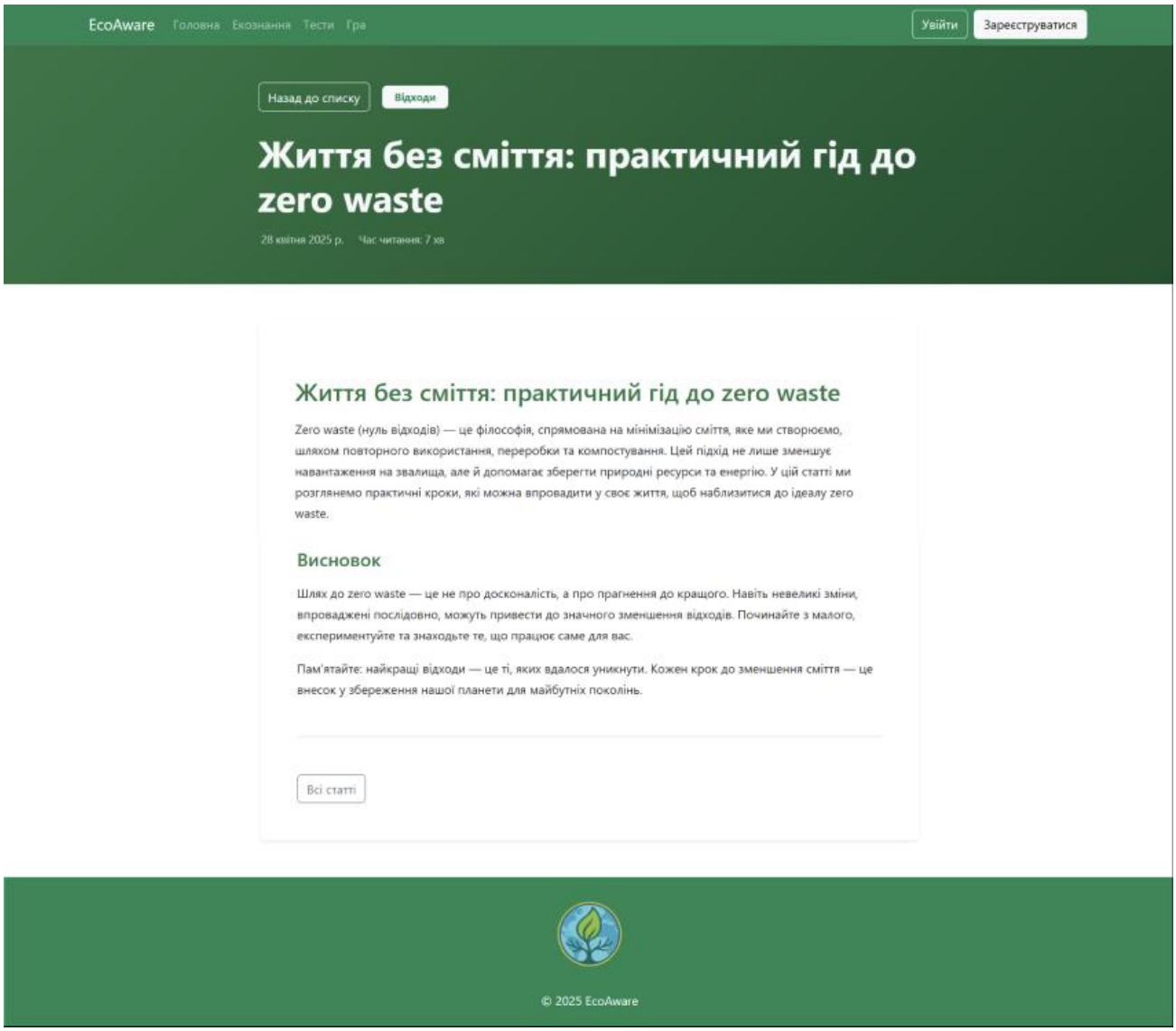


Рис. 4.15 Перегляд обраної статті

Сторінка тестів, яка зображена на рисунку 4.16, надає користувачам можливість перевірити свої екологічні знання. Подібно до сторінки статей, тут реалізовано пошук та фільтрацію за категоріями.

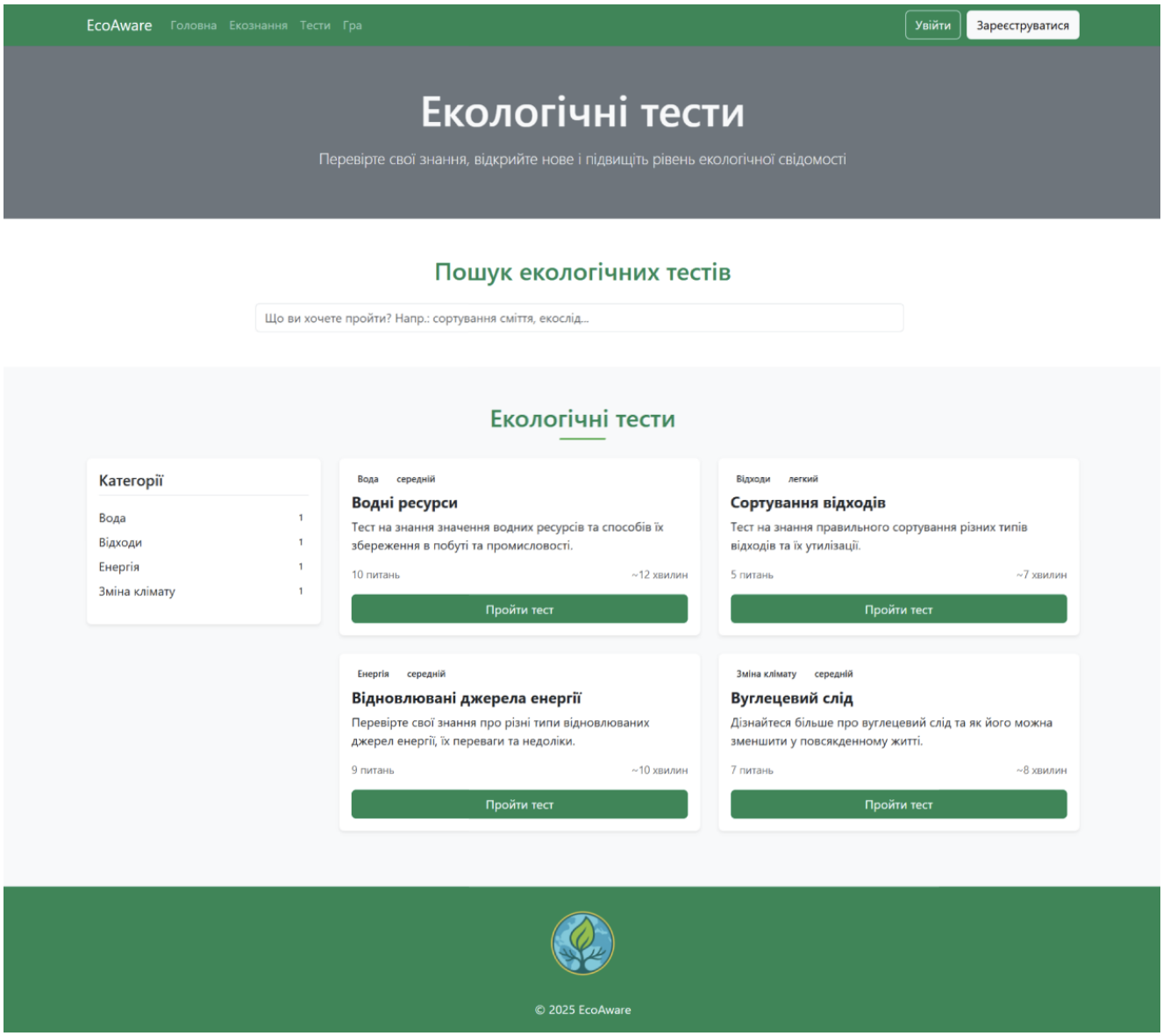


Рис. 4.16 Сторінка екологічних тестів

Коли користувач вибирає конкретний тест, відкривається сторінка проходження тесту. Компонент TestDetail.js, який зображений на рисунку 4.17, реалізує повний цикл тестування через систему станів.

```
const TestDetail = () => {
  const { id } = useParams();
  const navigate = useNavigate();

  const [test, setTest] = useState(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);
  const [currentStep, setCurrentStep] = useState('intro'); // intro, test, results

  const [currentQuestion, setCurrentQuestion] = useState(0);
  const [userAnswers, setUserAnswers] = useState([]);
  const [selectedOptions, setSelectedOptions] = useState([]);
  const [results, setResults] = useState(null);

  // Завантаження даних тесту
  useEffect(() => {
    Windsurf: Refactor | Explain | Generate JSDoc | X
    const fetchTest = async () => {
      try {
        setLoading(true);

        // Спочатку отримуємо загальну інформацію про тест
        const response = await fetch(`http://localhost:5000/api/tests/${id}`);

        if (!response.ok) {
          throw new Error('Не вдалося завантажити тест');
        }

        const data = await response.json();
        setTest(data);

        // Ініціалізуємо масив для відповідей користувача
        setUserAnswers(Array(data.questions.length).fill(null).map(() => ({
          questionId: '',
          selectedOptions: []
        })));

      } catch (err) {
        setError(err.message);
      } finally {
        setLoading(false);
      }
    };

    fetchTest();
  }, [id]);

  // Функція для обробки початку тесту
  Windsurf: Refactor | Explain | X
  const handleStartTest = async () => {
```

Рис. 4.17 Процес проходження тесту з прогрес-баром

Після завершення тесту відображаються детальні результати, які можна побачити на рисунку 4.18, з аналізом кожного питання та можливістю повторного проходження.

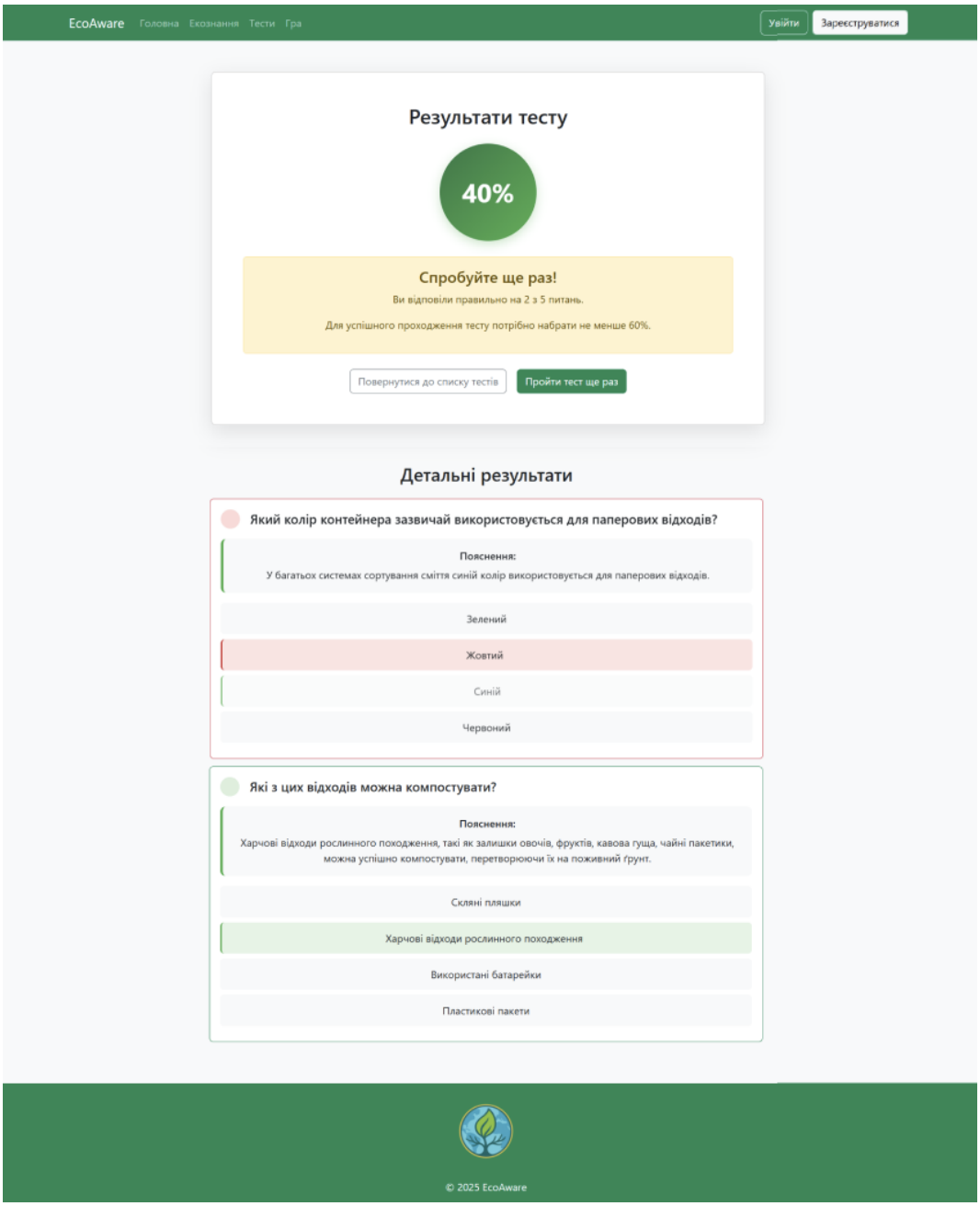


Рис. 4.18 Результати тесту з детальним аналізом

Сторінка гри "Сортування сміття", яка зображена на рисунку 4.19, організована через систему табів, що включає саму гру, таблицю лідерів та правила гри.

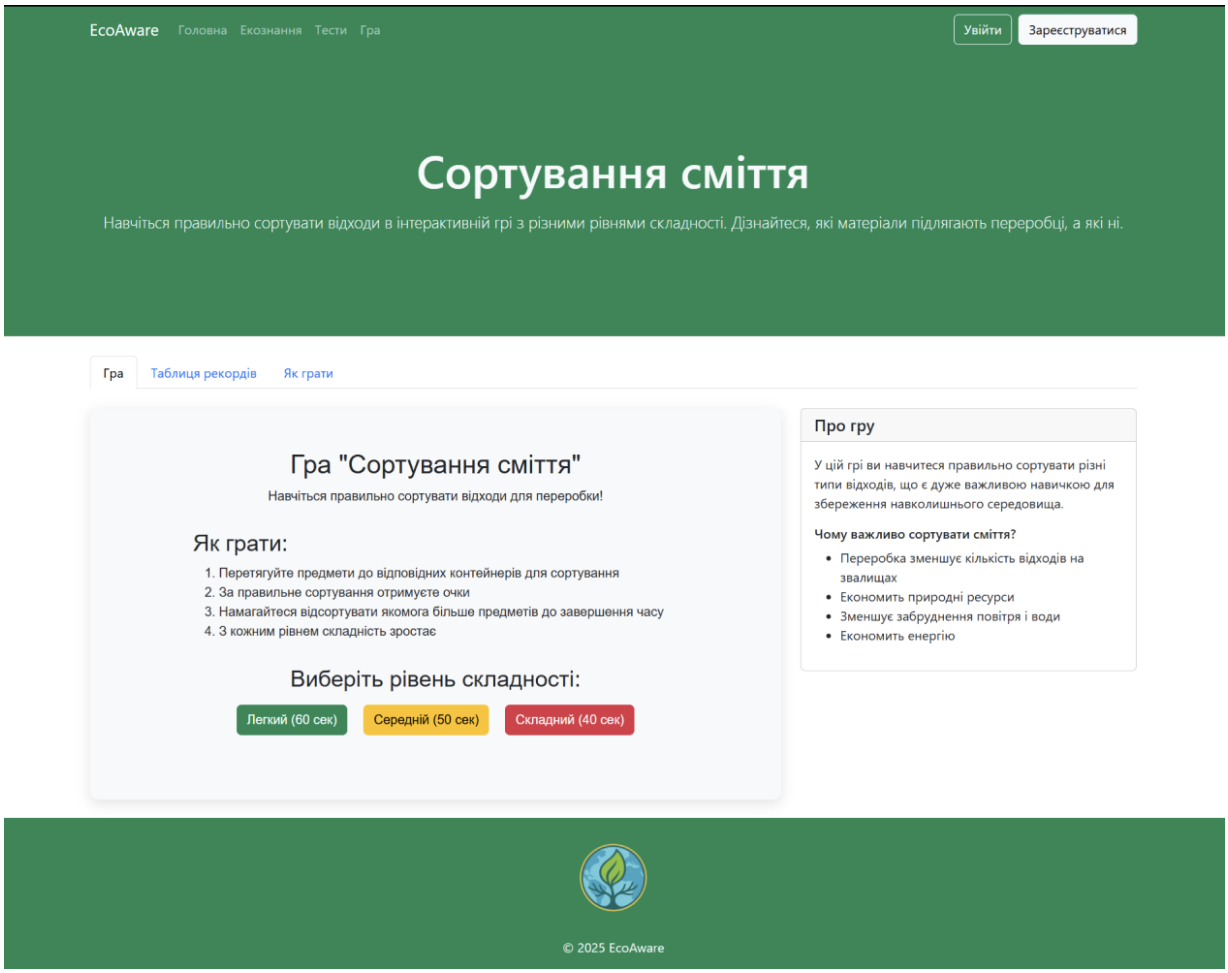


Рис. 4.19 Сторінка екологічної гри з поясненнями

Основний ігровий компонент TrashSortingGame, який зображений на рисунку 4.20, реалізує інтерактивну механіку drag-and-drop для сортування різних типів відходів у відповідні контейнери. Гра включає систему рівнів складності (легкий, середній, складний), що відрізняються швидкістю появи предметів та часом на прийняття рішення.

Система підрахунку очок нараховує бали за правильне сортування та відслідковує помилки. Таймер відображає залишковий час, а прогрес-бар показує кількість відсортованих предметів. При завершенні гри користувач отримує детальну статистику з можливістю збереження результатів у профілі.

```jsx
const TrashSortingGamePage = () => {
  const [activeTab, setActiveTab] = useState('game');

  return (
    <>
      {/* Hero Banner */}
      <div className="bg-success hero-section">
        <Container className="py-5 text-center text-light">
          <h1 className="display-4 mb-3">Сортування сміття</h1>
          <p className="lead mx-auto">
            Навчіться правильно сортувати відходи в інтерактивній грі з різними рівнями складності.
            Дізнайтеся, які матеріали підлягають переробці, а які ні.
          </p>
        </Container>
      </div>

      {/* Tabs Navigation */}
      <Container className="mt-4">
        <Tabs
          activeKey={activeTab}
          onSelect={(k) => setActiveTab(k)}
          className="mb-4"
        >
          <Tab eventKey="game" title="Гра">
            <Row>
              <Col lg={8} className="mb-4">
                <TrashSortingGame />
              </Col>

              <Col lg={4}>
                <div className="game-info-sidebar">
                  <Card className="mb-4">
                    <Card.Header as="h5">Про гру</Card.Header>
                    <Card.Body>
                      <p>
                        У цій грі ви навчитеся правильно сортувати різні типи відходів,
                        що є дуже важливою навичкою для збереження навколишнього середовища.
                      </p>
                      <h6>Чому важливо сортувати сміття?</h6>
                      <ul>
                        <li>Переробка зменшує кількість відходів на звалищах</li>
                        <li>Економить природні ресурси</li>
                        <li>Зменшує забруднення повітря і води</li>
                        <li>Економить енергію</li>
                      </ul>
                    </Card.Body>
                  </Card>

                </div>
```

Рис. 4.20 Організація ігрового модуля

Особистий профіль користувача, який зображений на рисунку 4.21, є централізованим місцем для кожного користувача вебсайта, де вони можуть керувати своєю присутністю на платформі. Ця сторінка надає користувачам можливість переглядати статистику прочитаних статей, результати пройдених тестів та досягнення в іграх.

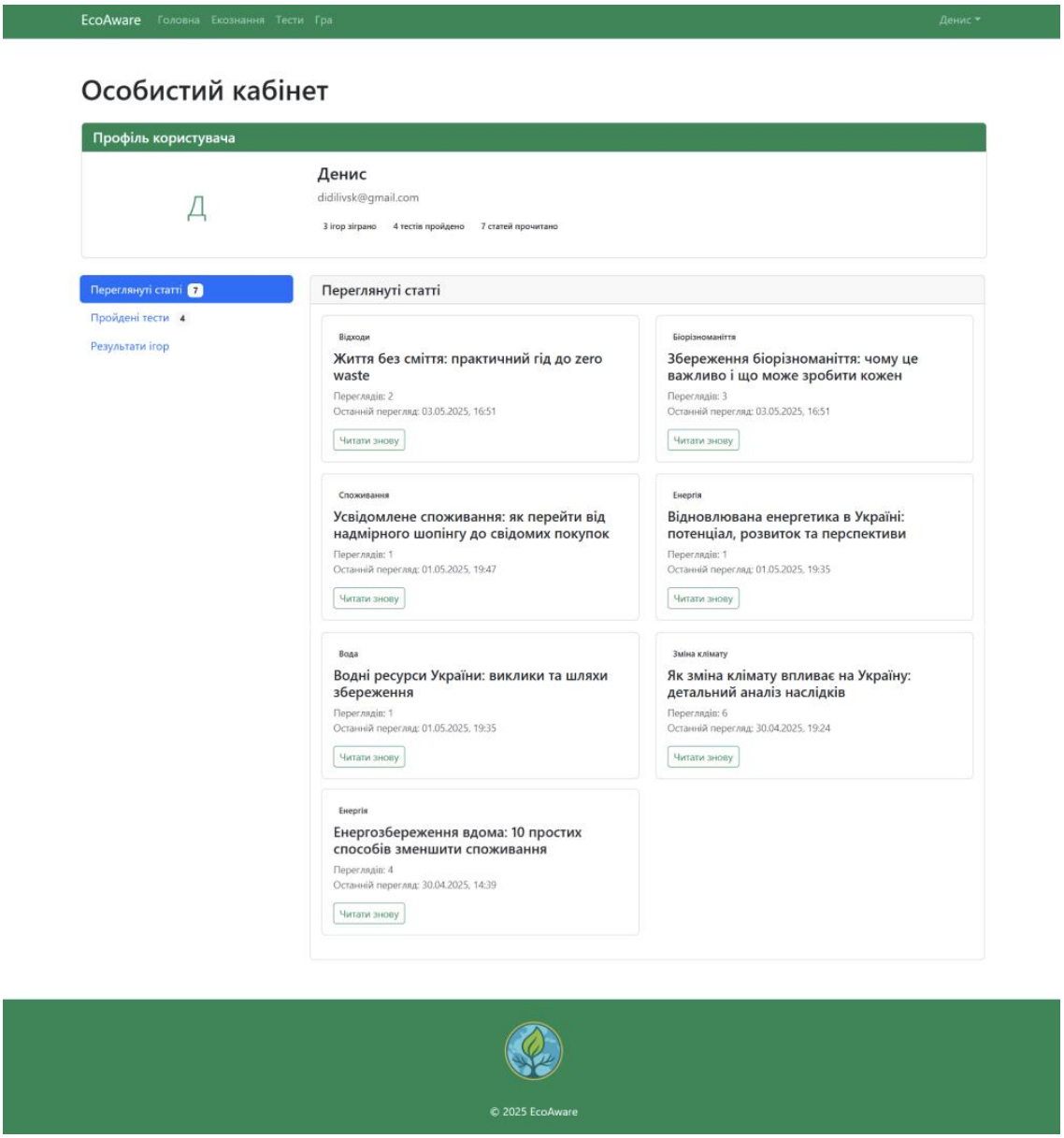


Рис. 4.21 Особистий кабінет з загальною інформацією користувача

Клієнтська частина EcoAware успішно реалізує всі функціональні вимоги екологічної освітньої платформи, забезпечуючи інтуїтивний, швидкий та доступний користувацький досвід для ефективного підвищення екологічної свідомості через інтерактивні статті, тести та ігри з персоналізованим відстеженням прогресу користувачів.

4.4 Тестування вебсайта

Тестування вебсайту EcoAware було проведено комплексно з метою забезпечення надійності, функціональності та зручності використання освітньої екологічної платформи. Процес тестування включав функціональне тестування користувацького інтерфейсу та продуктивності для гарантування якісного досвіду користувачів.

Таблиця 4.1

Тест-кейси для функціонального тестування

№ тест-кейсу	OK/NOK	Дії	Очікуваний результат
1	OK	1. Відкрити головну сторінку вебсайту EcoAware.	1. Відображається головна сторінка з розділами екологічних знань, тестів та ігор.
2	OK	1. Натиснути кнопку "Зареєструватися". 2. Заповнити форму реєстрації валідними даними. 3. Натиснути "Зареєструватися".	1. Відкривається форма реєстрації. 2. Користувач успішно реєструється та автоматично авторизується.
3	OK	1. Натиснути кнопку "Увійти". 2. Ввести правильні логін та пароль. 3. Натиснути "Увійти".	1. Відкривається форма авторизації. 2. Користувач успішно авторизується та перенаправляється на головну сторінку.
4	OK	1. Перейти до розділу "Екознання". 2. Ввести ключове слово у поле пошуку.	1. Відображається каталог екологічних статей. 2. Список статей фільтрується відповідно до пошукового запиту.

Продовження таблиці 4.1

Тест-кейси для функціонального тестування

№ тест-кейсу	OK/NOK	Дії	Очікуваний результат
5	OK	1. На сторінці "Екознання" обрати категорію зі списку. 2. Натиснути на обрану категорію.	1. Відображаються лише статті обраної категорії з оновленням лічильника результатів.
6	OK	1. Натиснути на картку статті в каталозі. 2. Переглянути повний зміст статті.	1. Відкривається детальна сторінка статті з повним текстом та метаданими. 2. Для авторизованих користувачів фіксується перегляд у статистиці.
7	OK	1. Перейти до розділу "Тести". 2. Обрати тест для проходження. 3. Натиснути "Пройти тест".	1. Відображається каталог доступних тестів з інформацією про складність. 2. Відкривається вступна сторінка обраного тесту.
8	OK	1. На вступній сторінці тесту натиснути "Почати тест". 2. Відповісти на всі питання. 3. Натиснути "Завершити тест".	1. Починається процес тестування з відображенням прогрес-бару. 2. Відображаються детальні результати з аналізом відповідей.
9	OK	1. Перейти до розділу "Гра". 2. Обрати рівень складності. 3. Розпочати гру "Сортування сміття".	1. Відкривається ігровий інтерфейс з інструкціями. 2. Запускається інтерактивна гра з системою очок та таймером.
10	OK	1. Під час гри перетягнути предмет до правильного контейнера. 2. Завершити гру.	1. Нараховуються очки за правильне сортування. 2. Відображаються результати гри та статистика.

ВИСНОВКИ

В результаті аналізу існуючих вебресурсів з екологічної тематики виявлено ключовий недолік — відсутність інтерактивних елементів для активного засвоєння знань. Більшість проаналізованих платформ, таких як "Екологія Право Людина", "Гагарка" та "Всесвіт", обмежуються лише інформаційним контентом без можливостей для практичного застосування отриманих знань. Це знижує ефективність навчального процесу та не сприяє формуванню стійких екологічних навичок у користувачів.

На основі аналізу існуючих аналогів сформульовано та реалізовано комплекс функціональних і нефункціональних вимог до вебсайту, що покращує користувацький досвід. Функціональні вимоги включають систему автентифікації, можливості перегляду та пошуку екологічного контенту, проходження інтерактивних тестів, участь у навчальних іграх та ведення персональної статистики. Нефункціональні вимоги забезпечують безпеку даних через JWT-шифрування, кросбраузерну сумісність та оптимальну швидкодію завантаження сторінок.

Спроектована та реалізована структура сайту включає три ключові розділи, а саме екознання, еко-гра, тести, що формують цілісну екосистему навчання з високим рівнем інтерактивності. Розділ "Екознання" надає доступ до структурованої бази екологічних статей з можливостями пошуку та фільтрації за категоріями. Модуль "Тести" реалізує систему перевірки знань з детальним аналізом результатів та поясненнями до кожного питання. Ігровий компонент "Сортування сміття" забезпечує практичне закріплення навичок екологічно відповідальної поведінки через інтерактивну механіку drag-and-drop з різними рівнями складності.

Розроблений функціонал реєстрації користувачів, проходження тестів, еко-гри та перегляду статистики дозволяє персоналізувати досвід кожного користувача та відстежувати його прогрес. Система автентифікації забезпечує

безпечний доступ до персональних функцій через JWT-токени з HTTP-only cookies. Особистий кабінет надає комплексну аналітику навчальної активності, включаючи історію прочитаних статей, результати пройдених тестів з кольоровим кодуванням успішності та детальну ігрову статистику за рівнями складності.

Проведено тестування вебсайту з метою відповідності функціональним та нефункціональним вимогам. Додатково проведено тестування продуктивності, сумісності з різними браузерами, адаптивності дизайну та безпеки системи.

Отже, розроблений вебсайт EcoAware успішно вирішує виявлену проблему відсутності інтерактивних методів екологічної освіти, надаючи користувачам комплексну платформу для підвищення екологічної свідомості. Реалізована архітектура забезпечує масштабованість системи та можливість додавання нових освітніх модулів у майбутньому. Інтеграція теоретичних знань з практичними навичками через ігрові елементи та систему тестування створює ефективне середовище для формування екологічно відповідальної поведінки серед широкої аудиторії користувачів.

ПЕРЕЛІК ПОСИЛАНЬ

1. Дідилівський Д.В., Яскевич В.О. Дослідження розробки інформаційного web-сайту для підвищення екологічної свідомості громадян. Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в ІКТ», Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. – К.: ДУІКТ, 2025. С. 133-135.

2. Дідилівський Д.В., Яскевич В.О. Використання протоколу OAuth 2.0 для захисту даних користувачів у вебзастосунках. VI Науково-технічна конференція «Сучасний стан та перспективи розвитку IoT», 15 квітня 2025 р., Київ, Державний університет інформаційно-комунікаційних технологій. (подано до друку).

3. Томчук М., Томчук С. РОЗВИТОК ЕКОЛОГІЧНОЇ СВІДОМОСТІ СТУДЕНТІВ ІНФОРМАЦІЙНИМИ ЗАСОБАМИ. *Науковий вісник Вінницької академії безперервної освіти. Серія «Педагогіка. Психологія».* 2022. С. 64.

4. Куць Н. Екологічна свідомість українців & довкілля: аналітичний документ, 2020. С. 31.

5. Екологія Право Людина. URL: https://epl.org.ua/.

6. Гагарка. URL: https://gagarka.com/.

7. Всесвіт. URL: https://vse-svit.com.ua/.

8. JavaScript MDN. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/.

9. Quick Start – React. URL: https://react.dev/learn.

10. React Router Official Documentation. URL: https://reactrouter.com/.

11. React Bootstrap. URL: https://react-bootstrap.netlify.app/.

12. Express - Node.js web application framework. URL: https://expressjs.com/.

13. Mongoose ODM v8.15.0. URL: https://mongoosejs.com/.

14. JSON Web Tokens. URL: https://jwt.io/introduction/.

15. Node.js — Introduction to Node.js. URL: https://nodejs.org/en/learn/getting-started/introduction-to-nodejs/.

16. Node.js Tutorial URL: https://www.geeksforgeeks.org/nodejs/.

17. MongoDB Documentation. URL: https://www.mongodb.com/docs/.

18. MongoDB Tutorial URL: https://www.w3schools.com/mongodb/index.php

19. Documentation for Visual Studio Code. URL: https://code.visualstudio.com/docs.

20. Figma: The Collaborative Interface Design Tool. URL: https://www.figma.com/design/

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка інформаційного web-сайту для підвищення екологічної свідомості громадян

Виконав студент 4 курсу
Групи ПД-41
Дідилівський Денис Віталійович
Керівник роботи
к.т.н, доц, доцент кафедри ІПЗ Яскевич Владислав Олександрович

Київ – 2025

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи –** підвищення екологічної свідомості громадян за рахунок використання інтерактивних методів навчання, реалізованих у вигляді інформаційного вебсайта.
- **Об’єкт дослідження –** процес підвищення екологічної свідомості громадян за допомогою вебсайта.
- **Предмет дослідження –** вебсайт EcoAware як засіб інтерактивного підвищення екологічної обізнаності населення.

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Проаналізувати наявні веб-ресурси, спрямовані на підвищення екологічної обізнаності.
2. Визначити функціональні та нефункціональні вимоги до інформаційного вебсайту.
3. Розробити структуру сайту з урахуванням розділів: екознання, еко-гра, тести.
4. Реалізувати функціонал реєстрації користувачів, проходження тестів, еко-гри і перегляд статистики.
5. Провести тестування вебсайту.

3

АНАЛІЗ АНАЛОГІВ

Характеристика	Екологія Право Людина	Гагарка	Всесвіт	EcoAware
Екологічні статі	+	+	+	+
Екологічні тести	-	-	-	+
Інтерактивна гра	-	-	-	+
Персональна статистика	-	-	-	+
Пошук і фільтрація по категоріям	+	+	+	+

4

ВИМОГИ ДО ВЕБСАЙТУ

Функціональні вимоги:

1. Реєстрація та авторизація користувачів.
2. Можливість перегляду, пошуку та фільтрації екологічних статей
3. Можливість проходження екологічних тестів з отриманням результатів.
4. Можливість участі у екологічній грі з системою рейтингу
5. Можливість переглядати персональну статистику прочитаних матеріалів, пройдених тестів та ігрового прогресу.

Нефункціональні вимоги:

1. Зберігання персональних даних користувачів в зашифрованому вигляді за допомогою JWT.
2. Вебсайт має працювати в браузерах Chrome, Edge, Opera.
3. Завантаження сторінки сайту до двох секунд.

5

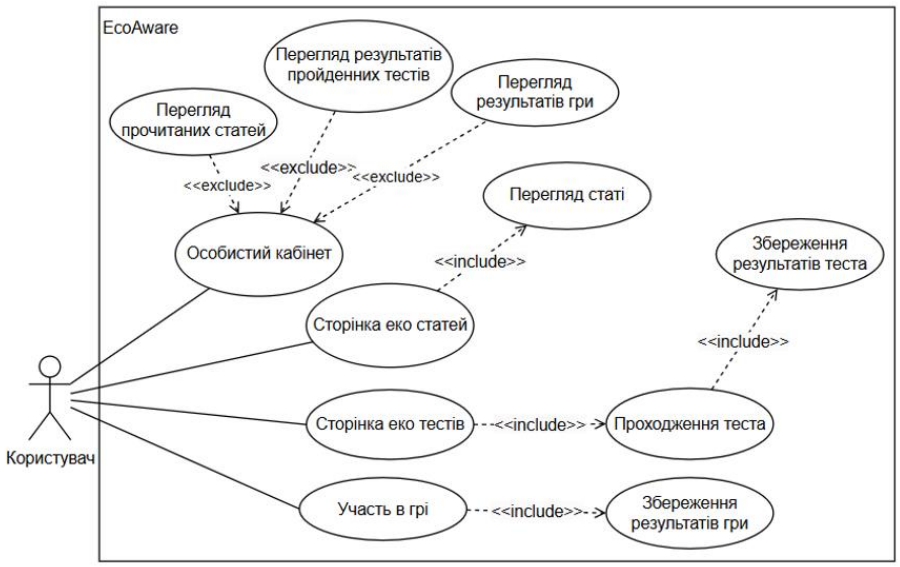
ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

Bootstrap

JSON Web Tokens

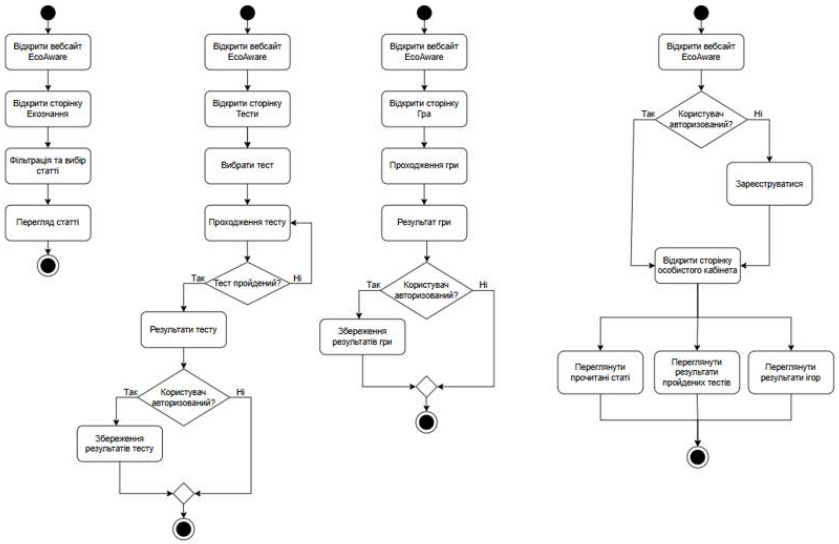
6

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ



7

ДІАГРАМИ ДІЯЛЬНОСТІ

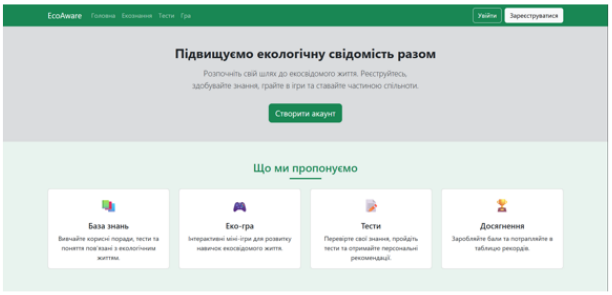


8

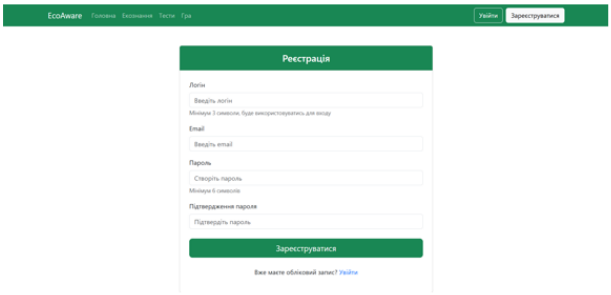
МАПА ВЕБСАЙТА

9

ЕКРАННІ ФОРМИ



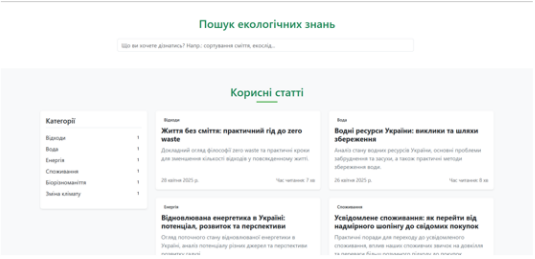
Головна сторінка сайта



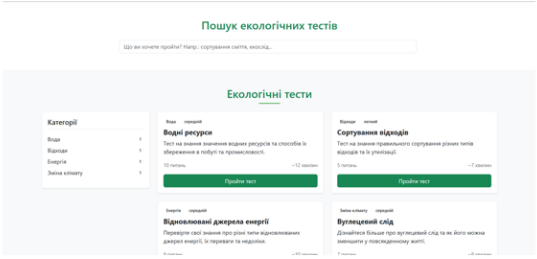
Реєстрація користувача

10

ЕКРАННІ ФОРМИ



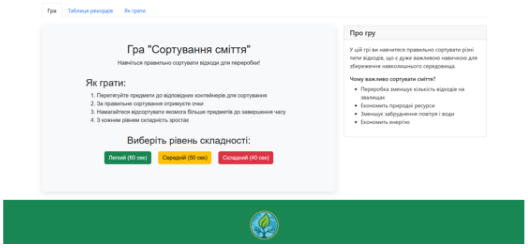
Сторінка зі статтями



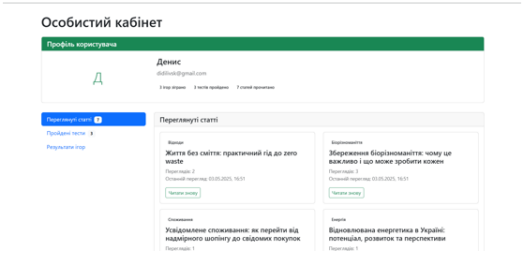
Сторінка з тестами

11

ЕКРАННІ ФОРМИ



Сторінка з грою



Особистий кабінет користувача

12

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Дідилівський Д.В., Замрій І.В. Дослідження розробки інформаційного web-сайту для підвищення екологічної свідомості громадян. Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в ІКТ», Збірник тез. 24.04.2025, ДУІКТ, м.Київ. К.: ДУІКТ, 2025. С. 133-135.
2. Дідилівський Д.В., Казначеєва А.В. Використання протоколу OAuth 2.0 для захисту даних користувачів у вебзастосунках. VI Науково-технічна конференція «Сучасний стан та перспективи розвитку IoT», 15 квітня 2025 р., Київ, Державний університет інформаційно-комунікаційних технологій. (подано до друку).

13

ВИСНОВКИ

1. В результаті аналізу існуючих веб-ресурсів з екологічної тематики виявлено ключовий недолік — відсутність інтерактивних елементів для активного засвоєння знань.
2. На основі аналізу існуючих аналогів сформульовано та реалізовано комплекс функціональних і нефункціональних вимог до веб-сайту, що покращує користувацький досвід.
3. Спроектована та реалізована структура сайту включає три ключові розділи, а саме екознання, еко-гра, тести, що формують цілісну екосистему навчання з високим рівнем інтерактивності.
4. Розроблений функціонал реєстрації користувачів, проходження тестів, еко-гри та перегляду статистики дозволяє персоналізувати досвід кожного користувача та відстежувати його прогрес.
5. Проведено тестування вебсайту з метою відповідності функціональним та нефункціональним вимогам.

14

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

```
const express = require('express');
const cors = require('cors');
const mongoose = require('mongoose');
const cookieParser = require('cookie-parser');
const jwt = require('jsonwebtoken');

require('dotenv').config();
require('./models/User');
require('./models/Article');
require('./models/Test');
require('./models/ViewedArticle');
require('./models/CompletedTest');

const app = express();
const PORT = process.env.PORT || 5000;

// Налаштування CORS для доступу з frontend
app.use(cors({
  origin: process.env.CLIENT_URL || 'http://localhost:3000',
  credentials: true,
  methods: ['GET', 'POST', 'PUT', 'DELETE', 'OPTIONS'],
  allowedHeaders: ['Content-Type', 'Authorization', 'Accept',
'X-User-ID']
}));

app.use(express.json());
app.use(cookieParser());

// Middleware для декодування JWT токена та створення
req.user
app.use((req, res, next) => {
  try {
    // Перевірка наявності токена в cookies
    const token = req.cookies.token;

    if (token) {
      // Декодування токена
      const decoded = jwt.verify(token,
process.env.JWT_SECRET ||
'ecoaware_secure_jwt_token_2024');

      // Встановлення об'єкта користувача в запит
      req.user = decoded;

    } } catch (error) {
    console.error('Помилка при декодуванні токена:', error);
  }

  // Продовження обробки запиту
  next();
});

// MongoDB Connection
mongoose.connect(process.env.MONGODB_URI ||
'mongodb://localhost:27017/ecoaware', {
  useNewUrlParser: true,
  useUnifiedTopology: true,
})
.catch(err => console.error('MongoDB connection error:',
err));

// Import routes
const articlesRoutes = require('./routes/articles');
const testsRoutes = require('./routes/tests');
const gamesRoutes = require('./routes/games');
const authRoutes = require('./routes/auth');

// Routes
app.use('/api/auth', authRoutes);
app.use('/api/articles', articlesRoutes);
app.use('/api/tests', testsRoutes);
app.use('/api/games', gamesRoutes);

// Маршрут для тестування авторизації
app.get('/api/auth-test', (req, res) => {
  if (req.user) {
    res.json({
      message: 'Ви авторизовані',
      userId: req.user.id,
      username: req.user.username
    });
  } else {
    res.status(401).json({
      message: 'Ви не авторизовані'
    });
  }
});

app.listen(PORT, () => {
  console.log(`Server is running on
http://localhost:${PORT}`);
});
// server/routes/articles.js
const express = require('express');
const router = express.Router();
const mongoose = require('mongoose');
const Article = require('../models/Article');
const ViewedArticle = require('../models/ViewedArticle');
const authRouter = require('./auth');

// Get the requireAuth middleware
const requireAuth = authRouter.requireAuth;

// Get all articles
router.get('/', async (req, res) => {
  try {
    const articles = await Article.find().sort({ date: -1 });
    res.json(articles);
  } catch (err) {
    console.error('Error fetching articles:', err);
    res.status(500).json({ message: 'Server error' });
  }
});

// Important: Define the /viewed endpoint BEFORE /:id to
prevent route conflicts
router.get('/viewed', requireAuth, async (req, res) => {
  try {

    // Check if ViewedArticle model is registered
    if (!mongoose.modelNames().includes('ViewedArticle')) {
      console.error('ViewedArticle model is not registered!');
      return res.status(500).json({
        success: false,
```

```
        message: 'Server error: Model not registered',
        error: 'ViewedArticle model is not registered'
      });
    }

    // Get viewed articles for the user, sorted by last viewed
date
    const viewedArticles = await ViewedArticle.find({ userId:
req.user.id })
      .sort({ lastViewedAt: -1 });

    res.status(200).json({
      success: true,
      articles: viewedArticles
    });
  } catch (error) {
    console.error('Error fetching viewed articles:', error);
    res.status(500).json({
      success: false,
      message: 'Server error while fetching viewed articles',
      error: error.toString(),
      stack: error.stack
    });
  }
});

// Get article by ID
router.get('/:id', async (req, res) => {
  try {
    const article = await Article.findById(req.params.id);

    if (!article) {
      return res.status(404).json({ message: 'Article not found'
});
    }

    res.json(article);
  } catch (err) {
    console.error('Error fetching article:', err);
    res.status(500).json({ message: 'Server error' });
  }
});

// Track article view - endpoint for authenticated users
router.post('/track-view', requireAuth, async (req, res) => {
  try {
    const { articleId, title, category } = req.body;
    const userId = req.user.id;

    if (!articleId || !title) {
      return res.status(400).json({
        success: false,
        message: 'Article ID and title are required'
      });
    }

    // Check if the user has already viewed this article
    let viewedArticle = await ViewedArticle.findOne({
      userId,
      articleId
    });

    if (viewedArticle) {
      // If article was already viewed, update it
      viewedArticle.viewCount += 1;
      viewedArticle.lastViewedAt = Date.now();
      await viewedArticle.save();
    } else {
      // If not viewed yet, create new record
      viewedArticle = new ViewedArticle({
        userId,
        articleId,
        title,
        category: category || 'Інше'
      });
      await viewedArticle.save();
    }

    res.status(200).json({
      success: true,
      message: 'Article view tracked successfully'
    });
  } catch (error) {
    console.error('Error tracking article view:', error);
    res.status(500).json({
      success: false,
      message: 'Server error while tracking article view'
    });
  }
});

module.exports = router;

// server/routes/tests.js
const express = require('express');
const router = express.Router();
const mongoose = require('mongoose');
const Test = require('../models/Test');
const CompletedTest = require('../models/CompletedTest');
const authRouter = require('./auth');

// Get the requireAuth middleware
const requireAuth = authRouter.requireAuth;

// IMPORTANT: Define the /completed endpoint BEFORE
other routes to prevent conflicts
router.get('/completed', requireAuth, async (req, res) => {
  try {

    // Check if CompletedTest model is registered
    if (!mongoose.modelNames().includes('CompletedTest')) {
      console.error('CompletedTest model is not registered!');
      return res.status(500).json({
        success: false,
        message: 'Server error: Model not registered',
        error: 'CompletedTest model is not registered'
      });
    }

    // Get completed tests for the user, sorted by completion
date
    const completedTests = await CompletedTest.find({
userId: req.user.id })
      .sort({ completedAt: -1 });

    res.status(200).json({
      success: true,
      tests: completedTests
    });
  } catch (error) {
    console.error('Error fetching completed tests:', error);
    res.status(500).json({
      success: false,
      message: 'Server error while fetching completed tests',
      error: error.toString(),
      stack: error.stack
    });
  }
```

```
});

// Track completed test - endpoint for authenticated users
router.post('/track-completion', requireAuth, async (req, res)
=> {
  try {
    const { testId, title, category, score, correctAnswers,
totalQuestions, timestamp } = req.body;
    const userId = req.user.id;

    if (!testId || !title || score === undefined) {
      return res.status(400).json({
        success: false,
        message: 'Test ID, title, and score are required'
      });
    }

    // Перевірка на існування запису з такими ж
параметрами за останні 10 хвилин
    const tenMinutesAgo = new Date();
    tenMinutesAgo.setMinutes(tenMinutesAgo.getMinutes() -
10);

    const existingRecord = await CompletedTest.findOne({
      userId,
      testId,
      score,
      completedAt: { $gt: tenMinutesAgo }
    });

    if (existingRecord) {
      return res.status(200).json({
        success: true,
        message: 'Тест вже було зареєстровано',
        duplicate: true
      });
    }

    // Створення нового запису
    const completedTest = new CompletedTest({
      userId,
      testId,
      title,
      category: category || 'Інше',
      score,
      correctAnswers: correctAnswers || 0,
      totalQuestions: totalQuestions || 0,
      completedAt: timestamp ? new Date(timestamp) : new
Date()
    });

    await completedTest.save();

    res.status(200).json({
      success: true,
      message: 'Результати тесту успішно збережено'
    });
  } catch (error) {
    console.error('Error tracking test completion:', error);
    res.status(500).json({
      success: false,
      message: 'Server error while tracking test completion'
    });
  }
});

// Get all tests
router.get('/', async (req, res) => {
  try {
    const tests = await Test.find();
    res.json(tests);
  } catch (err) {
    console.error('Error fetching tests:', err);
    res.status(500).json({ message: 'Server error' });
  }
});

// Get test by ID
router.get('/:id', async (req, res) => {
  try {
    const test = await Test.findById(req.params.id);

    if (!test) {
      return res.status(404).json({ message: 'Test not found' });
    }

    res.json(test);
  } catch (err) {
    console.error('Error fetching test:', err);
    res.status(500).json({ message: 'Server error' });
  }
});

// Start test - remove correct answers for security
router.get('/:id/start', async (req, res) => {
  try {
    const test = await Test.findById(req.params.id);

    if (!test) {
      return res.status(404).json({ message: 'Test not found' });
    }

    // Create a copy of the test without correct answers
    const testForUser = JSON.parse(JSON.stringify(test));

    // Remove isCorrect flag from each option to prevent
cheating
    testForUser.questions.forEach(question => {
      question.options.forEach(option => {
        delete option.isCorrect;
      });
    });

    res.json(testForUser);
  } catch (err) {
    console.error('Error preparing test:', err);
    res.status(500).json({ message: 'Server error' });
  }
});

// Check test answers
router.post('/:id/check', async (req, res) => {
  try {
    const { answers } = req.body;
    const test = await Test.findById(req.params.id);

    if (!test) {
      return res.status(404).json({ message: 'Test not found' });
    }

    // Calculate results
    const results = {
      correctAnswers: 0,
      totalQuestions: test.questions.length,
      details: []
    };

    test.questions.forEach((question, index) => {
```

```
      const userAnswer = answers[index];
      const correctOptions = question.options
        .filter(option => option.isCorrect)
        .map(option => option._id.toString());

      // Get user selected options and convert to strings for
comparison
      const userSelectedOptions =
userAnswer.selectedOptions.map(optionId =>
optionId.toString());

      // Check if user's answer is correct (all correct options
selected and no incorrect ones)
      const allCorrectOptionsSelected =
correctOptions.every(optionId =>
        userSelectedOptions.includes(optionId)
      );

      const noIncorrectOptionsSelected =
userSelectedOptions.every(optionId =>
        correctOptions.includes(optionId)
      );

      const isCorrect = allCorrectOptionsSelected &&
noIncorrectOptionsSelected;

      if (isCorrect) {
        results.correctAnswers++;
      }
      // Add detailed result for this question
      results.details.push({
        questionText: question.text,
        correct: isCorrect,
        userSelectedOptions: userSelectedOptions,
        correctOptions: correctOptions,
        explanation: question.explanation || ''
      });
    });
    // Calculate score as percentage
    results.score = ((results.correctAnswers /
results.totalQuestions) * 100).toFixed(2);
    // Determine if test was passed based on passing score
    results.passed = results.score >= test.passingScore;
    res.json(results);
  } catch (err) {
    console.error('Error checking test answers:', err);
    res.status(500).json({ message: 'Server error' });
  }
});
module.exports = router;
// Маршрут для збереження результатів гри "Сортування
сміття"
router.post('/trash-sorting/results', async (req, res) => {
  try {
    const { level, score, correct, incorrect, total, playTime } =
req.body;

    // Перевірка чи користувач авторизований
    if (!req.user) {
      return res.status(401).json({
        success: false,
        message: 'Для збереження результатів необхідно
авторизуватися'
      });
    }

    // Отримання ID користувача з req.user
(встановлюється middleware авторизації)
    const userId = req.user.id;

    // Створення ObjectId для userId
    let userObjectId;
    try {
      userObjectId = new ObjectId(userId);
    } catch (objIdError) {
      console.error('Помилка при створенні ObjectId:',
objIdError);
      return res.status(400).json({
        success: false,
        message: 'Неправильний формат ID користувача'
      });
    }

    // Підготовка даних з правильними типами
    const gameData = {
      gameType: 'trash-sorting',
      userId: userObjectId,
      level: parseInt(level) || 1,
      score: parseInt(score) || 0,
      correct: parseInt(correct) || 0,
      incorrect: parseInt(incorrect) || 0,
      total: parseInt(total) || 0,
      playTime: parseInt(playTime) || 0
    };

    // Обчислюємо accuracy (якщо не буде обчислено в
моделі)
    if (gameData.total > 0) {
      gameData.accuracy = Math.round((gameData.correct /
gameData.total) * 100);
    } else {
      gameData.accuracy = 0;
    }

    // Створення нового запису результату гри
    const gameResult = new GameResult(gameData);

    // Зберігаємо результат
    await gameResult.save();

    // Відправляємо відповідь
    return res.status(201).json({
      success: true,
      message: 'Результати збережено успішно',
      result: gameResult
    });
  } catch (error) {
    console.error('Помилка при збереженні результатів
гри:', error);
    return res.status(500).json({
      success: false,
      message: 'Помилка при збереженні результатів',
      error: error.message
    });
  }
});

// Отримання рейтингу (топ-10 кращих результатів)
router.get('/trash-sorting/leaderboard', async (req, res) => {
  try {
    const { level } = req.query;

    const query = { gameType: 'trash-sorting' };

    // Фільтрація за рівнем, якщо вказано
    if (level && level !== 'all') {
      query.level = parseInt(level);
    }
```

```
    // Отримання кращих результатів
    const leaderboard = await GameResult.find(query)
      .sort({ score: -1 })  // Сортування за спаданням очок
      .limit(10)  // Обмеження до 10 кращих результатів
      .select('score correct incorrect total accuracy playTime
level createdAt')
      .populate('userId', 'username')  // Заповнення поля
користувача, якщо є зв'язок з колекцією користувачів
      .lean();  // Перетворення результатів на прості JS
об'єкти

    return res.json({
      success: true,
      leaderboard
    });
  } catch (error) {
    console.error('Помилка при отриманні рейтингу:', error);
    return res.status(500).json({
      success: false,
      message: 'Помилка при отриманні рейтингу',
      error: error.message
    });
  }
});

// Отримання статистики користувача
router.get('/trash-sorting/stats', async (req, res) => {
  try {

    // Декодування токена вручну, якщо req.user відсутній
але токен є
    if (!req.user && req.cookies.token) {
      try {
        const jwt = require('jsonwebtoken');
        const decoded = jwt.verify(
          req.cookies.token,
          process.env.JWT_SECRET ||
'ecoaware_secure_jwt_token_2024'
        );
        req.user = decoded;
      } catch (tokenError) {
        console.error('Помилка декодування токена в
маршруті stats:', tokenError);
      }
    }

    // Перевірка на наявність авторизації
    if (!req.user) {

      // Відповідаємо з порожньою статистикою замість
помилки 401
      return res.json({
        success: true,
        message: 'Статистика для неавторизованого
користувача',
        stats: {
          totalGames: 0,
          totalScore: 0,
          correctIncorrect: { correct: 0, incorrect: 0, total: 0,
accuracy: 0 },
          levelStats: []
        }
      });
    }

    // Отримання ID користувача з req.user
    const userId = req.user.id;

    // Переконаємося, що модель GameResult доступна
    if (!GameResult || typeof GameResult.find !== 'function') {
      console.error('Модель GameResult недоступна або не
має методу find');
      return res.json({
        success: true,
        message: 'Помилка доступу до моделі даних',
        stats: {
          totalGames: 0,
          totalScore: 0,
          correctIncorrect: { correct: 0, incorrect: 0, total: 0,
accuracy: 0 },
          levelStats: []
        }
      });
    }

    // Створення ObjectId для userId
    let userObjectId;
    try {
      userObjectId = new ObjectId(userId);
    } catch (objIdError) {
      console.error('Помилка при створенні ObjectId:',
objIdError);

      // Спробуємо використати рядкове представлення ID
      try {
        // Рахуємо ігри без перетворення на ObjectId
        const countWithoutObjectId = await
GameResult.countDocuments({
          gameType: 'trash-sorting',
          userId: userId
        });

        if (countWithoutObjectId > 0) {
          // Якщо знайдено ігри, продовжуємо з рядковим ID
          const allResults = await GameResult.find({
            gameType: 'trash-sorting',
            userId: userId
          }).lean();

          // Обчислення статистики
          // ... (решта коду функції)
        }
      } catch (secondaryError) {
        console.error('Помилка при альтернативному
пошуку:', secondaryError);
      }

      // Повертаємо порожню статистику
      return res.json({
        success: true,
        message: 'Проблема з ідентифікатором користувача',
        stats: {
          totalGames: 0,
          totalScore: 0,
          correctIncorrect: { correct: 0, incorrect: 0, total: 0,
accuracy: 0 },
          levelStats: []
        }
      });
    }

    // Загальна статистика
    const totalGames = await GameResult.countDocuments({
      gameType: 'trash-sorting',
      userId: userObjectId
    });
```

```
    // Якщо користувач не грав жодної гри
    if (totalGames === 0) {
      return res.json({
        success: true,
        stats: {
          totalGames: 0,
          totalScore: 0,
          correctIncorrect: { correct: 0, incorrect: 0, total: 0,
accuracy: 0 },
          levelStats: []
        }
      });
    }

    // Отримання всіх результатів для обчислення
статистики вручну
    const allResults = await GameResult.find({
      gameType: 'trash-sorting',
      userId: userObjectId
    }).lean();

    // Обчислення загальної статистики
    let totalScore = 0;
    let totalCorrect = 0;
    let totalIncorrect = 0;
    let totalItems = 0;

    const levelMap = new Map();

    // Обробка результатів
    allResults.forEach(result => {
      // Загальна статистика
      totalScore += result.score || 0;
      totalCorrect += result.correct || 0;
      totalIncorrect += result.incorrect || 0;
      totalItems += result.total || 0;

      // Статистика за рівнями
      const level = result.level || 1;
      if (!levelMap.has(level)) {
        levelMap.set(level, {
          level,
          gamesPlayed: 0,
          totalScore: 0,
          maxScore: 0,
          totalPlayTime: 0
        });
      }

      const levelStat = levelMap.get(level);
      levelStat.gamesPlayed += 1;
      levelStat.totalScore += result.score || 0;
      levelStat.maxScore = Math.max(levelStat.maxScore,
result.score || 0);
      levelStat.totalPlayTime += result.playTime || 0;
    });

    // Перетворення Map на масив
    const levelStats =
Array.from(levelMap.values()).map(levelStat => ({
      level: levelStat.level,
      gamesPlayed: levelStat.gamesPlayed,
      avgScore: Math.round(levelStat.totalScore /
levelStat.gamesPlayed),
      maxScore: levelStat.maxScore,
      totalPlayTime: levelStat.totalPlayTime,
      avgPlayTime: Math.round(levelStat.totalPlayTime /
levelStat.gamesPlayed)
    })).sort((a, b) => a.level - b.level);

    // Формування відповіді
    const stats = {
      totalGames,
      totalScore,
      correctIncorrect: {
        correct: totalCorrect,
        incorrect: totalIncorrect,
        total: totalItems,
        accuracy: totalItems > 0 ? Math.round((totalCorrect /
totalItems) * 100) : 0
      },
      levelStats
    };

    return res.json({ success: true, stats });
  } catch (error) {
    console.error('Помилка при отриманні статистики:',
error);

    // Повертаємо порожню статистику замість помилки
500
    return res.json({
      success: true,
      message: 'Виникла помилка при отриманні
статистики',
      stats: {
        totalGames: 0,
        totalScore: 0,
        correctIncorrect: { correct: 0, incorrect: 0, total: 0,
accuracy: 0 },
        levelStats: []
      }
    });
  }
});

// Отримання рекомендацій на основі результатів гри
router.get('/trash-sorting/recommendations', async (req, res)
=> {
  try {
    // Перевірка чи користувач авторизований
    if (!req.user) {
      return res.status(401).json({
        success: false,
        message: 'Для отримання рекомендацій необхідно
авторизуватися'
      });
    }

    // Отримання ID користувача з req.user
    const userId = req.user.id;

    // Створення ObjectId для userId
    let userObjectId;
    try {
      userObjectId = new ObjectId(userId);
    } catch (objIdError) {
      console.error('Помилка при створенні ObjectId:',
objIdError);
      return res.status(400).json({
        success: false,
        message: 'Неправильний формат ID користувача'
      });
    }

    // Отримання останніх результатів гри
    const latestResults = await GameResult.find({
      gameType: 'trash-sorting',
```

```
      userId: userObjectId
    })
      .sort({ createdAt: -1 })
      .limit(5)
      .lean();

    // Аналіз слабких місць у сортуванні
    const recommendations = {
      improvementAreas: [],
      generalTips: [
        'Пам\'ятайте, що папір і картон завжди підлягають
переробці, якщо не забруднені їжею',
        'Пластикові пляшки слід сплющувати перед
викиданням для економії місця',
        'Органічні відходи можна компостувати у
спеціальних контейнерах',
        'Скляні пляшки різних кольорів слід сортувати
окремо'
      ]
    };

    // Приклад логіки для визначення областей для
покращення
    if (latestResults.length > 0) {
      let totalAccuracy = 0;
      latestResults.forEach(result => {
        const accuracy = result.total > 0 ? (result.correct /
result.total) * 100 : 0;
        totalAccuracy += accuracy;
      });

      const averageAccuracy = totalAccuracy /
latestResults.length;

      if (averageAccuracy < 60) {
        recommendations.improvementAreas.push({
          area: 'Загальні знання про сортування',
          tip: 'Перегляньте основні правила сортування
відходів у вашому місті'
        });
      }
    }

    return res.json({ success: true, recommendations });
  } catch (error) {
    console.error('Помилка при отриманні рекомендацій:',
error);
    return res.status(500).json({
      success: false,
      message: 'Помилка при отриманні рекомендацій',
      error: error.message
    });
  }
});

module.exports = router;
```