МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТЕХНОЛОГІЙ

ННІ/факультет	Інформаційних технологій
Кафедра	Інформатики і прикладного програмного забезпечення
Спеціальність	Інженерія програмного забезпечення
Форма навчання	Денна

КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА

Татару Євгеній Павлович
(прізвище, ім'я, по батькові здобувача)

на тему **Розробка програмного забезпечення догляду за рослинами**
(повна назва теми)

за матеріалами праць провідних спеціалістів з розробки ПЗ та проектування БД
(повна назва бази дослідження)

науковий керівник **к.е.н., доцент** _____ **Лисенко В.С**
(наук. ступінь, вчене звання) *(підпис)* *(прізвище, ініціали)*

Робота допущена до захисту в ЕК
Протокол засідання кафедри
від 11.06.2025 № 12
Завідувач кафедри _____
(підпис)
д.т.н., професор Зеленський О.С.
Наук. ступень, вчене звання *Ініціали, прізвище*

Кривий Ріг – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТЕХНОЛОГІЙ

ННІ/факультет	Інформаційних технологій
Кафедра	Інформатики і прикладного програмного забезпечення
Спеціальність	Інженерія програмного забезпечення
Форма навчання	Денна

«ЗАТВЕРДЖУЮ»
Завідувач кафедри _____ Зеленський О.С.
(підпис) (Прізвище, ініціали)
«11» червня 2025 року

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ

1. Тема роботи Розробка програмного забезпечення догляду за рослинами

Керівник роботи _к.е.н, доцент_Лисенко В.С_____
затверджені наказом закладу вищої освіти від «04» квітня 2025 р. № 222-ст

2. Строк подання здобувачем роботи до «09» червня 2025 р.

3. Зміст кваліфікаційної роботи, об’єкт, предмет та мета дослідження:

Розділ 1. Постановка задачі

Розділ 2. Проєктування задачі

Розділ 3. Проєктування бази даних для проєкту

Розділ 4. Розробка програмного забезпечення

Об’єкт дослідження: програмне забезпечення, орієнтоване на підтримку і розвиток навичок догляду за кімнатними рослинами

Предмет дослідження: структура, логіка й інтерфейс вебзастосунку, а також способи подання інформації, що дозволяють полегшити засвоєння знань користувачем

Мета кваліфікаційної роботи: створити програмне забезпечення, що допомагає користувачам краще орієнтуватися в догляді за кімнатними рослинами

5. Дата видачі завдання «04» квітня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів МДР	Строк виконання етапів роботи	Відмітка керівника про виконання етапів (дата, підпис)
1	Підготовка розділу 1	04.04.2025-13.04.2025	
2	Підготовка розділу 2	14.04.2025-26.04.2025	
3.	Підготовка розділу 3	27.04.2025-15.05.2025	
4	Підготовка розділу 4	16.05.2025-08.06.2025	
5	Реєстрація завершеної кваліфікаційної роботи	09.06.2025	Реєстраційний №___ «09»червня 2025 р.
6	Отримання відгуку від наукового керівника	10.06.2025	
7	Подання кваліфікаційної роботи на перегляд завідувачу кафедри	11.06.2025	
8	Отримання зовнішньої рецензії	12.06.2025	
9	Попередній захист кваліфікаційної роботи на кафедрі	13.06.2025	
10	Підготовка до захисту в ЕК	16.06.2025-21.06.2025	

Завдання підготував науковий керівник _____ ___Лисенко В.С____

(підпис) (прізвище та ініціали)

Завдання одержав _____ __Татару Є.П._____

(підпис) (прізвище та ініціали)

ЗГОДА здобувача(чки) вищої освіти

Державного університету економіки і технологій про перевірку кваліфікаційної роботи на прояви академічного плагіату та розміщення в Репозитарії Університету

Я, _____Татару Євгеній Павлович_____, підтримую політику Державного університету економіки і технологій з академічної доброчесності і відкритого доступу.

Засвідчую, що кваліфікаційна бакалаврська робота Розробка програмного забезпечення замовлень клієнта у ресторані виконана самостійно та не містить академічного плагіату. Я не надавав(ла) і не одержував(ла) недозволену допомогу під час підготовки цієї роботи. Робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

Із чинним Положенням про запобігання та виявлення академічного плагіату в роботах здобувачів вищої освіти Державного університету економіки і технологій ознайомлений(а). Чітко усвідомлюю, що в разі виявлення у кваліфікаційній роботі порушення норм академічної доброчесності робота не допускається до захисту або оцінюється незадовільно.

Також я поінформований(на), що відповідно до «Положення про Репозитарій (електронну базу даних) Державного університету економіки і технологій» зазначена робота буде розміщена в Електронному архіві Університету (Репозитарії ДУЕТ). З умовами такого розміщення ознайомлений(на).

Дата підпис ініціали, прізвище (власноруч)

АНОТАЦІЯ

на бакалаврську кваліфікаційну роботу

«Розробка програмного забезпечення догляду за рослинами»

Татару Євгеній Павлович

Кваліфікаційна бакалаврська робота на здобуття освітньо-кваліфікаційного рівня бакалавра зі спеціальності 121 «Інженерія програмного забезпечення» – Державний університет економіки і технологій – Кривий Ріг, 2025.

У бакалаврській роботі розроблено два продукти: веб-застосунок та настільну навчальну гру. Веб-застосунок створено з використанням React і GraphQL — він дає змогу переглядати товари, читати публікації та оформлювати замовлення. Інтерфейс адаптивний, побудований за принципом SPA, що забезпечує швидку і зручну взаємодію.

Навчальна гра реалізована на C# (Windows Forms) і моделює догляд за кімнатними рослинами. Користувач переміщує рослину по кімнаті з різними зонами освітлення. Алгоритм обчислює відповідність між потребами рослини у світлі та умовами навколо, показуючи результат через візуальний індикатор.

Ключові слова: ОСВІТНЯ ГРА, ВЕБ-ЗАСТОСУНОК, REACT, GRAPHQL, ІНТЕРАКТИВНА СИМУЛЯЦІЯ, ДОГЛЯД ЗА РОСЛИНАМИ.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД(база даних)	Впорядкований набір логічно взаємопов’язаних даних, що використовуються спільно та призначені для задоволення інформаційних потреб користувачів.
СУБД	Система управління базами даних.
ПЗ	Програмне забезпечення.
ADO .NET	ActiveX Data Object для .NET – технологія доступу і управління базами даних для платформи .NET

ЗМІСТ

ВСТУП

У сучасному світі все більше людей цікавляться кімнатними рослинами як елементом естетики, засобом покращення мікроклімату приміщень та способом відпочинку. Люди прагнуть створити затишок у своїх оселях, урізноманітнити інтер'єр, додати живого акценту у просторі. Проте сам процес вирощування квітів вимагає певного розуміння їхніх потреб. Не завжди вдається самостійно визначити, де краще поставити вазон, скільки світла йому потрібно чи як часто варто поливати.

У багатьох виникає потреба у простому і зручному джерелі знань, де можна знайти відповіді на типові питання по догляду за кімнатними рослинами. Багатьом хочеться не просто читати поради, а спробувати все на практиці. Так простіше зрозуміти, чому одній рослині добре біля вікна, а іншій — у затінку. Це не завжди очевидно з текстів, тому з'являється потреба у такому форматі, де можна самостійно ознайомитися з умовами і побачити результат. Ідея поєднати корисну інформацію, невеликий магазин і навчальний компонент стала основою цієї роботи.

Мета дипломного проєкту — створити програмне забезпечення, що допомагає користувачам краще орієнтуватися в догляді за кімнатними рослинами. Воно складається з трьох основних частин: сайт-блог із порадами, модуль з можливістю замовлення базових товарів і інтерактивна гра, в якій користувач має змогу експериментально визначати комфортне для рослини місце в кімнаті залежно від рівня освітлення.

Завдання, які ставилися під час розробки:

1. зібрати базову інформацію про найбільш популярні кімнатні рослини та умови їх утримання;
2. створити структуру блогу та реалізувати просту навігацію;
3. реалізувати модуль онлайн-магазину з базовим функціоналом;

4. розробити ігровий компонент, що моделює кімнату з кількома зонами освітлення та дозволяє перевірити, наскільки обране місце підходить конкретній рослині;
5. забезпечити простоту використання ресурсу на різних пристроях.

Актуальність роботи пояснюється поширеністю інтересу до теми кімнатних рослин, особливо в умовах міського способу життя. Багато людей працюють вдома або проводять чимало часу у квартирі й прагнуть зробити простір навколо себе більш приємним. При цьому не всі готові витрачати час на глибоке вивчення ботаніки, тому потрібні зручні інструменти, які дозволяють вчитися поступово, у зручному форматі.

Об'єктом дослідження є програмне забезпечення, орієнтоване на підтримку і розвиток навичок догляду за кімнатними рослинами.

Предметом дослідження виступає структура, логіка й інтерфейс веб-застосунку, а також способи подання інформації, що дозволяють полегшити засвоєння знань користувачем.

У роботі поєднуються елементи веб-розробки, веб-дизайну та основ графічного дизайну, оскільки однією з цілей є створення інтерактивної гри для розстановки рослин, які варто намалювати за допомогою засобів OpenGL.

РОЗДІЛ І
ПОСТАНОВКА ЗАДАЧІ

1.1. Характеристика задачі

У роботі розглядається завдання створення програмного забезпечення, яке допомагає краще зрозуміти основи догляду за кімнатними рослинами. Ідея полягає в поєднанні двох платформ: веб-сайту та десктопної гри. Вони мають спільну тему, орієнтовані на одну й ту ж аудиторію, але виконують різні функції. Сайт — це інформаційне середовище, де можна дізнатися більше про світ рослин і знайти потрібне приладдя. Застосунок — це інструмент, у якому користувач навчається на практиці, спостерігаючи за реакцією рослини на зміну середовища.

Блогова частина містить публікації з описами різних видів рослин, порадами щодо їх розміщення, поливу, пересадки та інших нюансів. Матеріали написані простою мовою, розраховані на початківців. Важливо було подати інформацію у зручному для сприйняття вигляді, без перевантаження термінами. Магазин дозволяє користувачу одразу придбати базові товари, які найчастіше потрібні для догляду: горщики, ґрунт, добрива.

В інтерактивній частині, невеликій програмі-грі, користувач потрапляє в умовну кімнату з трьома зонами освітлення. У кожну з них можна поставити рослину й одразу побачити, наскільки їй підходить це місце. Візуалізація зроблена так, щоб було зрозуміло: світла забагато, замало чи достатньо. Користувач має змогу експериментувати, змінювати положення і вчитися на практиці, а не просто за текстом.

Усі ці частини працюють разом: блог дає загальну інформацію, гра дозволяє її перевірити, а магазин — одразу діяти, якщо щось потрібно докупити. Завданням було зробити сайт простим, логічним, таким, щоб ним хотілося користуватись і повертатись до нього. Технічно проєкт охоплює

веброзробку, роботу з інтерфейсами, базами даних і реалізацію логіки гри у браузері.

Проєкт розраховано на людей без спеціальної освіти у сфері ботаніки. Це можуть бути ті, хто лише нещодавно почав займатися кімнатними рослинами або планує їх завести. Також ресурс буде корисним для школярів, які цікавляться природознавством, або для дорослих, які хочуть краще облаштувати свій дім. Головна вимога — мінімальні цифрові навички та бажання вчитись.

Отже, програмне забезпечення розраховане на людей, які не мають глибоких знань у сфері ботаніки, але прагнуть навчитися доглядати за кімнатними рослинами. Це можуть бути:

- молоді люди, які щойно переїхали у власне житло;
- батьки, які хочуть залучити дитину до природничої тематики;
- літні люди, яким цікаво займатися рослинами як хобі;
- школярі, які можуть використовувати гру як навчальний інструмент.

Загальна ціль проєкту — зробити взаємодію з рослинами цікавою, а знання — доступними. Сайт і десктопна гра ніяк не пов’язані, але логічно доповнюють одне одного. У блозі згадується можливість встановити гру, є окрема сторінка з описом і посиланням для завантаження. У грі в свою чергу містяться короткі поради з сайту у вигляді підказок. Завдяки цьому користувач бачить проєкт як єдине ціле, навіть якщо частини реалізовані окремо.

Гра розробляється як окремий застосунок для Windows. Вона не потребує інсталяції складного програмного середовища, працює автономно й не вимагає постійного підключення до Інтернету. Основна увага — простота у використанні, зрозумілий інтерфейс, стабільна робота на середньостатистичному ноутбуці та слабких відеокартах.

1.2. Огляд існуючих блогів

Перед розробкою власного сайту було проаналізовано кілька популярних ресурсів, присвячених темі догляду за кімнатними рослинами. Мета — зрозуміти, як подається інформація, які формати використовуються, наскільки зручно користуватись ресурсом, і що можна зробити краще або інакше.

Один із найвідоміших англомовних ресурсів є «The Sill» (Рис. 1.1.). Він поєднує блог, магазин і онлайн-школу. У статтях прості поради, велика кількість фото, відповіді на типові питання. Проте багато матеріалів орієнтовані на покупців із США, тому частина назв рослин або брендів може бути незнайомою для українського користувача.

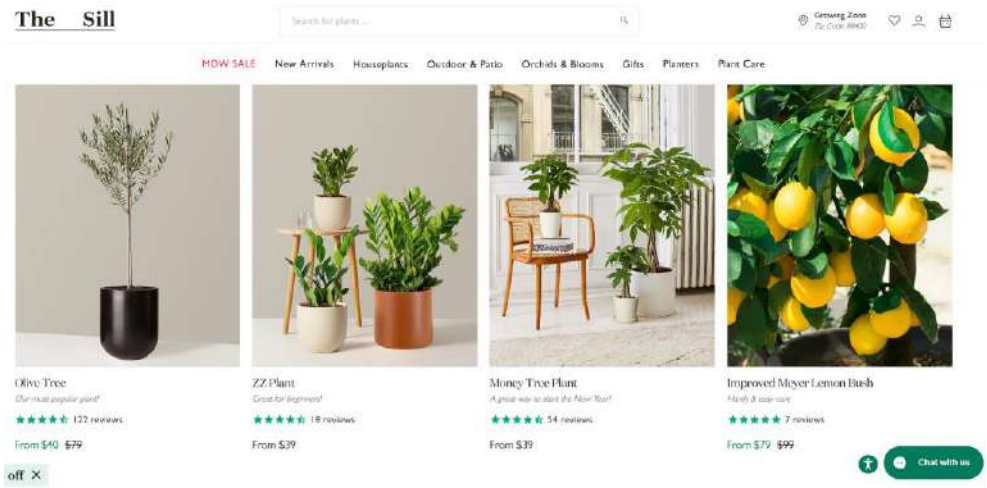

Рис. 1.1. Головна сторінка сайту «The Sill»

Україномовний сайт «Рости квітка» спеціалізується на квітах, які можна вирощувати вдома (Рис. 1.2.). Блог поділений за категоріями, є словник термінів, окремий розділ з питаннями та відповідями. Візуально сайт простий, місцями навіть застарілий. Інформація переважно текстова, з невеликою кількістю зображень.

Рис. 1.2. Сторінка зі статтями на сайті «Рости квітка»

Ще один приклад сучасного блогу — «Houseplant Central» (Рис. 1.3.). Добре структурований, статті мають прості заголовки й короткі підрозділи, що зручно для читання з телефону. Проте немає української мови, а візуальна частина побудована шаблонними засобами.

Рис. 1.3. Інтерфейс блогу «Houseplant Central»

Після огляду стало зрозуміло, що більшість блогів:

- орієнтовані або на зовнішній ринок, або на більш досвідчених користувачів;
- рідко пропонують інтерактив — усе зводиться до читання;
- часто не мають простого дизайну, зрозумілого для новачків;

- назви, сезонність, побутові звички не адаптовані під українське середовище.

У власному проєкті ці моменти враховано: контент адаптований під локальні умови, подається у зручному форматі, з можливістю одразу закріпити знання у практичній грі.

1.3. Проблематика предметної області

Популярність кімнатних рослин постійно зростає. Люди прагнуть зробити свої помешкання теплішими, живішими, природнішими. Але бажання мати рослину — це лише перший крок. Далі починаються труднощі. Одні — практичні, інші — пов’язані зі звичками, а деякі — з інформаційною плутаниною.

Навіть не дуже вибагливі рослини потребують уваги. Найпоширеніші проблеми виникають через:

- неправильне розміщення в кімнаті — надто темно або занадто сонячно;
- нерозуміння, скільки води потрібно;
- помилки з температурою або провітрюванням;
- добрива, що застосовуються «на око».

Усі ці фактори — дрібниці на перший погляд. Але саме з них інтерес до сайту та бажання передивлятися там статті. Якщо людині складно розібратися на сайті, вона просто його лишає та шукає інший.

Доступ до інформації є, але вона не завжди допомагає. Чому?

1. Надто загальні або, навпаки, надто спеціалізовані статті.
2. Різні джерела суперечать одне одному.
3. Поради не адаптовані до конкретного середовища.
4. Багато тексту — мало конкретики.

Людина витрачає більше часу на те, щоб розібратись, ніж на сам догляд.

Ще один аспект — психологія. Багато хто сприймає вазон як «живу відповідальність» — особливо ті, хто не мав досвіду догляду навіть за

домашніми тваринами. Навіть проста дія — переставити рослину в інше місце — викликає сумніви: «а раптом їй стане гірше». Страх зробити неправильно паралізує бажання діяти.

Брак середовища для безпечної проби та тесту розстановки — ще одна невидима, але важлива проблема. Було б значно легше прийняти рішення, якби спершу можна було побачити модельну ситуацію.

Догляд за кімнатними рослинами часто сприймається як щось суто інстинктивне або таке, що не потребує спеціальних знань. Проте насправді це процес, що вимагає системного підходу та оволодіння конкретними практичними навичками. Наприклад:

- уміння оцінити освітленість приміщення;
- читання сигналів рослини;
- розуміння сезонності росту;
- базове знання про субстрати.

Це знання, яке потрібно десь здобути. У школі цього не вчать. На YouTube — забагато відео з різною якістю. У підсумку — новачок просто не знає, з чого почати.

Одна з ключових проблем полягає в тому, що жоден з популярних сайтів не поєднує в собі:

- читання статей;
- можливість візуального прикладу;
- умовно-безпечне середовище для спроб;
- прямий зв’язок із магазином.

Усе це розкидано по різних платформах. Частину знань користувач бере з Instagram, частину з форуму, частину — від знайомих, і все одно результат часто незрозумілий. Аби цей процес став цілісним, потрібен єдиний простір — ідеально, якщо він дає змогу не лише читати, а й діяти.

1.4. Вимоги до програми

З огляду на поставлену мету та враховуючи попередній аналіз існуючих рішень, сформульовано перелік вимог до програмного забезпечення, що складається з двох основних компонентів: інформаційного сайту-блогу та настільного застосунку з елементами гри.

Загальні вимоги:

1. Система має бути зрозумілою для користувачів без спеціальної підготовки.
2. Інтерфейс повинен підтримувати українську мову як основну.
3. Контент повинен адаптуватися під різні типи користувачів: новачки, аматори, підлітки.

Функціональні вимоги до сайту-блогу:

1. Надання користувачу структурованих текстових матеріалів про догляд за кімнатними рослинами.
2. Розподіл матеріалів за категоріями: освітлення, полив, температура, пересадка, типи рослин тощо.
3. Пошук по сайту за ключовими словами.
4. Інтеграція невеликого інтернет-магазину з товарами для догляду.
5. Форма для додавання нових статей.

Функціональні вимоги до настільної гри-застосунку:

1. Візуалізація кімнати з різними умовами освітлення (три зони, три промені).
2. Наявність кількох типів рослин, кожна з яких має свої оптимальні параметри розміщення.
3. Можливість користувачу змінювати положення рослини між зонами.
4. Підрахунок ступеня відповідності умов потребам рослини у відсотках.
5. Можливість запуску на більшості сучасних комп’ютерів без встановлення додаткового програмного забезпечення.

Висновки до розділу 1

У першому розділі було розглянуто загальні аспекти догляду за кімнатними рослинами, а також проведено огляд існуючих блогів і платформ, присвячених цій темі. Аналіз показав, що доступна інформація часто буває недостатньо адаптованою до локальних умов та не завжди зрозумілою для початківців.

Виявлено кілька основних проблем, які ускладнюють процес навчання та догляду за рослинами: відсутність інтерактивних інструментів для практичного застосування знань, надмірна інформаційна складність, психологічні бар’єри та недостатній рівень локалізації контенту.

На основі проведеного аналізу сформульовано вимоги до програмного забезпечення, що включає сайт-блог та настільний застосунок у вигляді інтерактивної гри. Таке рішення покликане зробити процес пізнання та догляду за рослинами доступнішим, більш наочним і приємним для користувача.

Загалом, дослідження обґрунтувало необхідність створення комплексного інструменту, який поєднуватиме інформаційний, освітній і практичний аспекти, що ляжуть в основу подальшої розробки.

РОЗДІЛ ІІ
ПРОЄКТУВАННЯ ЗАДАЧІ

2.1. Проєктування гри

Алгоритм функціонування гри розпочинатиметься з вибору користувачем одного з рівнів, кожен із яких визначатиме певний тип рослини із унікальними параметрами потреби у світлі. Після визначення рівня будуть встановлені початкові умови: задаються фізичні характеристики рослини та параметри зон освітлення, розташованих у просторі кімнати.

Простір кімнати розбиватиметься на кілька зон із чіткими межами, кожна з яких характеризуватиметься числовим показником інтенсивності світла. Ці зони виступатимуть як геометричні області, координати яких будуть задані у двовимірній системі, що дозволить визначати положення об’єктів у просторі.

Рослина рухатиметься по ігровому полю, її позиція постійно оновлюватиметься. Переміщення рослини відстежуватиметься за допомогою безперервного оновлення координат. Для визначення світлових умов у місці розташування рослини обчислюватиметься евклідова відстань між центром рослини та центрами кожної із зон освітлення. Зона з мінімальною відстанню визначатиметься як найближча, і її параметр інтенсивності світла вважатиметься значенням, що характеризує умови для рослини.

Далі відбуватиметься порівняння потреб рослини у світлі з фактичним рівнем інтенсивності у визначеній зоні. Результатом такого порівняння стане числовий показник відповідності, який інтерпретуватиметься як індекс якості розміщення рослини. Цей індекс слугуватиме основою для формування зворотного зв’язку у вигляді візуального індикатора.

Якщо індекс якості свідчитиме, що рослина перебуватиме поза межами сприятливих умов освітлення, індикатор відсотку буде горіти червоним кольором. Це означатиме що існує потреба у переміщенні рослини ближче до

тієї зони, параметри освітлення якої найкраще відповідатимуть потребам даного виду.

Алгоритм також передбачатиме логіку переходу між рівнями, де умови складності поступово зростатимуть, а вимоги до користувача — ускладнюватимуться. Кожен наступний рівень задаватиме нові параметри рослин і нову умову для постановку рослини у відповідну зону освітлення, що вимагатиме від користувача адаптації та застосування набутого досвіду.

Положення рослини на ігровому полі змінюватиметься в реальному часі, відображаючись у вигляді координат, які оновлюватимуться під час її переміщення користувачем за допомогою миші. Одночасно на екрані відбуватиметься рендеринг ігрового поля, що включатиме графічне зображення кімнати з різними зонами освітлення, а також візуалізацію самої рослини у вибраному місці. Крім того, інтерфейс відображатиме кольоровий індикатор, який демонструватиме рівень освітленості у поточній позиції рослини.

У верхній частині екрану розміщуватимуться кнопки управління, зокрема «Меню» для повернення до вибору рівня та «Наступний рівень» для переходу до більш складних завдань. Ці елементи керування завжди будуть активними, забезпечуючи зручну навігацію і можливість швидко змінювати параметри гри. Весь інтерфейс формуватиметься та оновлюватиметься в режимі реального часу, щоб користувач завжди мав актуальну інформацію і міг оперативно реагувати на зміни положення рослини та її взаємодію із зонами освітлення.

Алгоритм роботи гри досить зрозумілий і логічний (Рис. 2.1.).

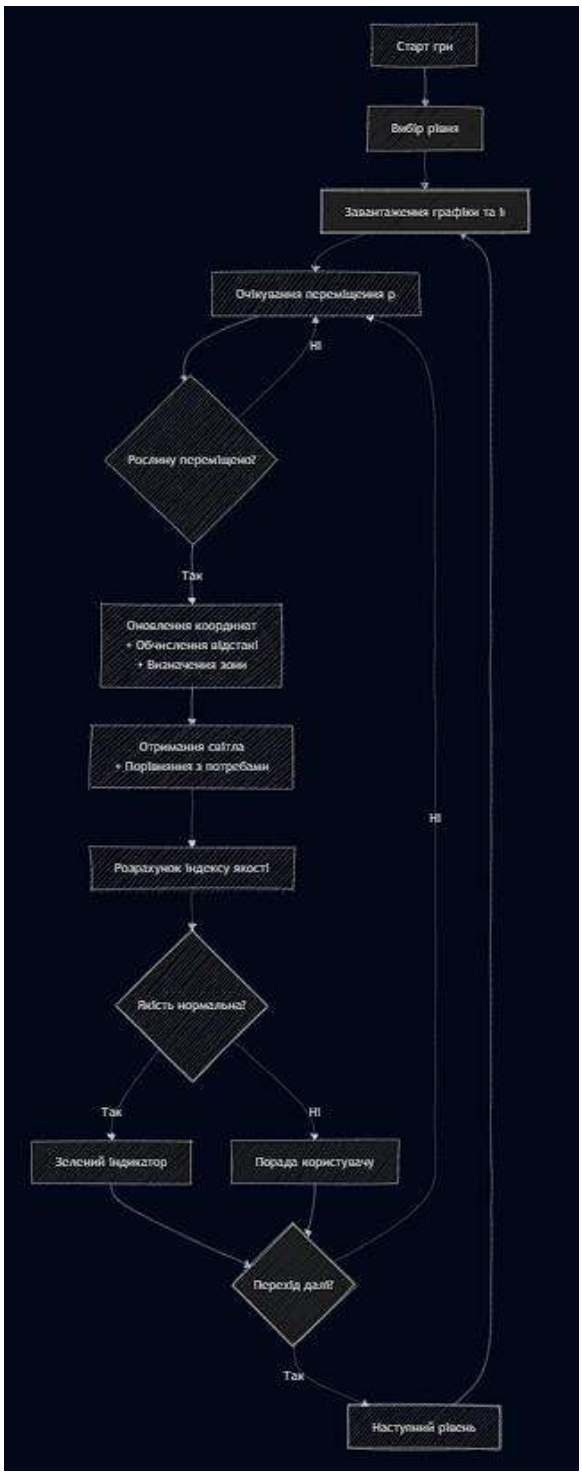

Рис. 2.1. Алгоритм квізу

Спочатку гравець вибирає рівень, і це визначає, які рослини і які умови освітлення будуть у грі. Потім рослину можна рухати по ігровому полю, а її положення постійно оновлюється — це дає можливість зрозуміти, де саме вона знаходиться. Щоб визначити, наскільки добре рослина освітлена, програма порівнює її позицію з зонами світла, які теж задані в грі. Зона з

найближчими до потрібного джерела світла координатами вважається тією, що впливає на рослину, і її інтенсивність світла враховується для оцінки.

Далі відбувається порівняння того, скільки світла треба рослині, і скільки її фактично отримує. Виходить певний показник, який показує, наскільки правильно вона розміщена. Якщо все погано, гра підкаже, куди її краще пересунути. Так гравець вчиться знаходити оптимальні умови. Кнопки для повернення в меню чи переходу до наступного рівня виконують алгоритм руху по програмі.

В цілому, алгоритм не дуже складний, але його вистачить, щоб гра була цікавою і зрозумілою, а також щоб гравець міг поступово вчитися краще розуміти, як правильно розставляти рослини в кімнаті з різним освітленням. В майбутньому можна додати нові рівні, та розділи з поливанням рослин та підбором потрібного добрива.

2.2. Проєктування веб-сайту

У межах розробки інформаційної системи планується створення веб-застосунку, який стане зручною платформою як для ознайомлення з продукцією, так і для здійснення онлайн-замовлень та перегляду тематичних публікацій. Уся логіка клієнтської частини буде проєктуватись із урахуванням простоти використання, зрозумілості та гнучкої адаптації до потреб різних категорій користувачів. Із самого початку закладається підхід, при якому всі ключові взаємодії будуть розподілені між окремими сторінками, що дозволить уникати перевантаження інтерфейсу та водночас забезпечить повноту функціоналу. Уся система проєктується як односторінковий застосунок (SPA), що забезпечуватиме динамічне оновлення вмісту без повного перезавантаження сторінки.

Після переходу на сайт користувач потраплятиме на головну сторінку, яка виконуватиме функцію вітрини — тут відображатиметься коротка презентація бренду, заклик до дії та можливість перейти далі. Основною

точкою навігації буде загальне меню, яке міститиме посилання на основні розділи: «Магазин», «Блог», «Про нас» і додаткові функціональні компоненти, як-от перемикач теми та індикатор кошика. Уся логіка переходів між сторінками реалізовуватиметься з використанням маршрутизатора, що дозволить користувачу переміщуватись між розділами без перезавантаження.

Структура сайту передбачає централізоване керування даними та станом через React-хуки, без використання глобального сховища чи Context API. Це дозволить зберегти простоту та прозорість архітектури, а також забезпечити легке масштабування у майбутньому. Наприклад, логіка кошика зберігатиметься у локальному стані, але буде доступна в потрібних компонентах завдяки кастомізованим хукам [Додаток Е].

У межах магазину користувач зможе переглядати перелік доступних товарів, кожен з яких буде відображено у вигляді окремої картки. Картки міститимуть зображення, назву, опис та ціну, а також кнопку для додавання товару до кошика. У верхній частині сторінки користувач матиме доступ до індикатора кошика, натискання на який відкриватиме модальне вікно із переліком усіх доданих позицій. Тут можна буде змінювати кількість товарів або видаляти їх. Оформлення замовлення передбачатиме відкриття форми, де користувач введе необхідні контактні дані. Після підтвердження замовлення буде сформовано запит на сервер, який або збереже його до бази, або поверне повідомлення про помилку.

Цей сценарій описано у вигляді діаграми, що моделює типовий ланцюг дій від моменту додавання товару до підтвердження замовлення (Рис. 2.2). Таке структурування забезпечує логічну завершеність кожної дії та дозволяє користувачу відчувати контроль над процесом, що є важливим фактором довіри до веб-застосунку.

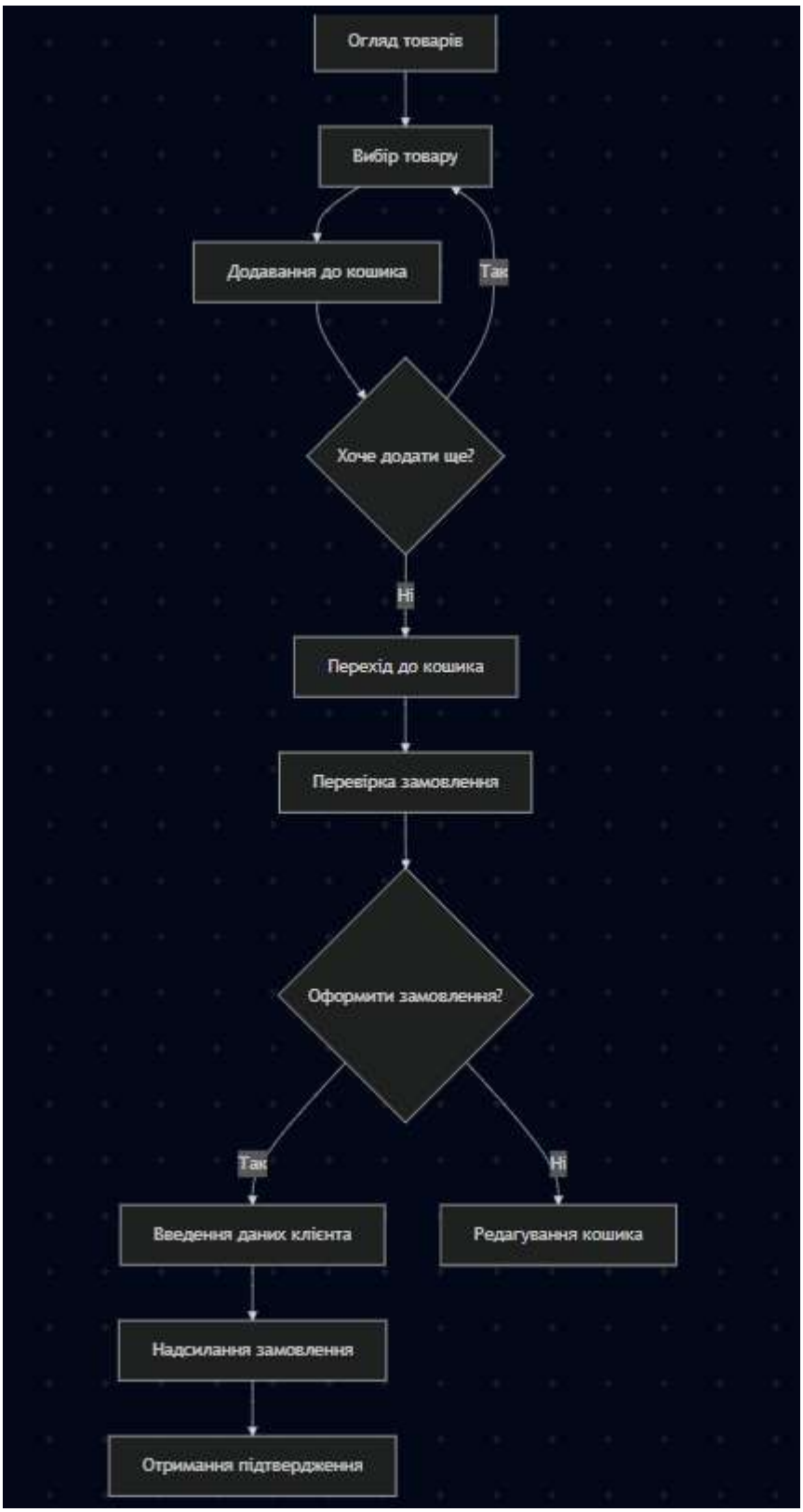

Рис. 2.2. Діаграма підтвердження замовлення

Окрім торгового функціоналу, на сайті буде реалізовано розділ блогу. Тут користувач зможе переглядати публікації, які розміщуватимуться у

вигляді карток із заголовком, фрагментом тексту та кнопкою «Докладніше». Кожна картка відображатиме короткий зміст публікації, а натискання на неї здійснюватиме перехід на сторінку повного перегляду. Контент буде завантажуватись динамічно через запити до сервера, що дозволить оновлювати записи без перезавантаження сторінки. У перспективі планується додати фільтрацію публікацій за ключовими словами та категоріями.

Крім цього, існуватиме окрема сторінка для створення публікацій. Вона вже реалізована і наразі є загальнодоступною, тобто будь-хто з користувачів зможе додати новий запис до блогу. Проте в майбутньому заплановано додати механізм авторизації, що обмежить цю можливість лише для зареєстрованих користувачів або адміністраторів. Відповідна база даних для авторизації вже створена, що спрощуватиме наступні кроки у розвитку функціоналу.

Проєктована логіка роботи сайту враховує загальну послідовність дій користувача: від ознайомлення з вмістом — до активної взаємодії, зокрема оформлення замовлення та публікаційного контенту. Цей підхід буде реалізовано у вигляді інтуїтивного та функціонального інтерфейсу, що оновлюється в реальному часі, забезпечуючи плавність взаємодії. Повна схема взаємодії користувача із загальними розділами сайту наведена у вигляді діаграми (Рис. 2.3).

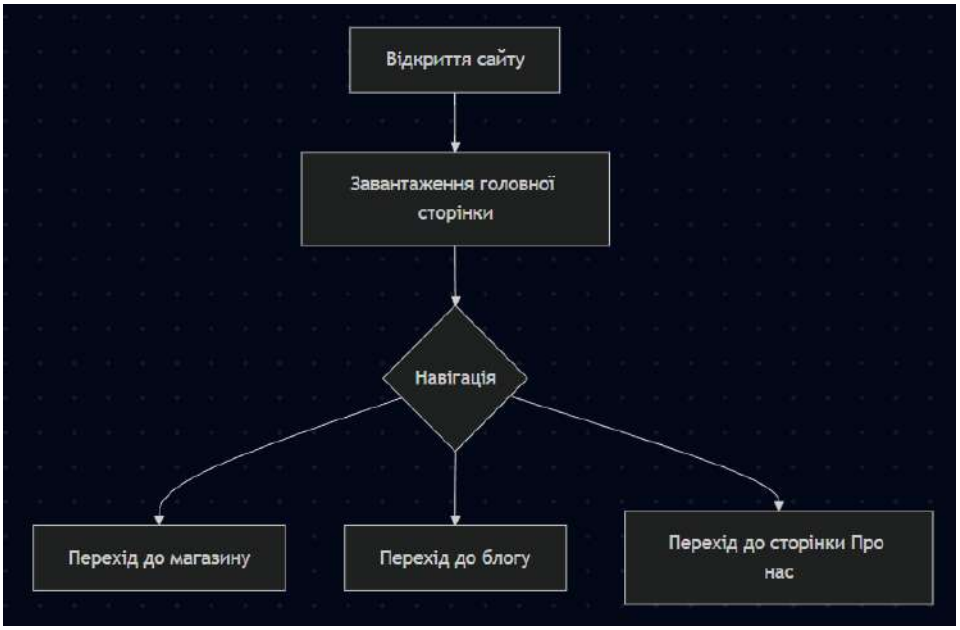

Рис. 2.3. Діаграма взаємодії з розділами сайту

Таким чином, усе функціонування веб-сайту спроєктоване з урахуванням практичності, масштабованості та майбутнього розширення. Система буде гнучкою до змін, а інтерфейс — адаптованим для швидкого та зручного використання.

Висновки до розділу 2

У межах даного розділу було розглянуто алгоритм функціонування навчальної гри, орієнтованої на формування у користувача базових навичок догляду за кімнатними рослинами, зокрема щодо оптимального вибору освітлення. Розроблена логіка гри передбачає послідовне проходження рівнів зі зростанням складності, інтерактивне оновлення координат рослини в режимі реального часу та надання візуального зворотного зв’язку у вигляді кольорових індикаторів.

Запропонований підхід поєднує в собі геометричне моделювання простору кімнати, розрахунок відстаней до зон освітлення, а також систематичне порівняння поточних умов із потребами конкретного типу рослини. Це дозволяє не лише перетворити процес навчання у гру, а й реалізувати інтуїтивно зрозумілий механізм, що стимулює гравця до вдосконалення прийняття рішень на основі зібраних даних.

Паралельно із функціональною частиною гри, у розділі було охарактеризовано структуру майбутнього веб-застосунку, розробленого на основі сучасних фронтенд-технологій . Основна увага приділялася створенню зручного інтерфейсу користувача, організації навігації за допомогою маршрутизатора та реалізації функцій магазину й блогу в межах односторінкового застосунку. Локальне керування станом, модульна архітектура та використання кастомних React-хуків забезпечують як простоту розробки, так і гнучкість у подальшому розвитку системи.

РОЗДІЛ ІІІ
ПРОЄКТУВАННЯ БАЗИ ДАНИХ ДЛЯ ПРОЄКТУ

3.1. Структура бази даних

Проєктування бази даних є необхідною передумовою для побудови стабільної та зрозумілої інформаційної системи, яка забезпечує збереження й організацію даних, взаємодію між компонентами застосунку та послідовну логіку обробки інформації. У межах системи для веб-сайту "Зелена Нота", що поєднує функціональність онлайн-магазину та блогу, база даних виконує роль внутрішнього механізму, який дозволяє реалізувати численні процеси — від збереження текстового контенту до обробки замовлень клієнтів. Вибір системи управління базами даних у таких проєктах безпосередньо впливає на архітектуру програмного забезпечення, а також визначає можливості підтримки, переносу та масштабування проєкту в межах його цільового середовища.

Для цієї системи використано SQLite — вбудовану реляційну систему управління базами даних, яка зберігає всі дані в одному файлі та не передбачає окремого серверного процесу. Така модель зберігання є зручною для застосунків, що працюють у межах обмеженого навантаження, а також у випадках, коли перевагою є простота розгортання та автономність. SQLite забезпечує підтримку основного підмножини стандарту SQL [12], дозволяючи виконувати запити з використанням агрегатних функцій, умов, сортувань і зв’язків між таблицями. Це дає змогу розробникам реалізувати необхідну бізнес-логіку, не вдаючись до зовнішніх залежностей або складної конфігурації.

Порівняно з серверними системами, такими як PostgreSQL чи MySQL, SQLite функціонує без постійного фоново запущеного процесу та не вимагає окремих механізмів для мережевої взаємодії. Це зменшує складність підтримки й дозволяє уникнути потреби в адмініструванні прав доступу,

сертифікатів або балансування навантаження. Натомість, у проєктах з високою кількістю одночасних звернень до бази або потребою в гнучкому керуванні ресурсами така система може мати певні обмеження, зокрема щодо паралельної обробки записів. Тому її використання найбільш виправдане у структурах, орієнтованих на читання, з контрольованою частотою записів та оновлень.

Особливістю SQLite є відсутність необхідності у зовнішньому налаштуванні. Усі основні компоненти можуть взаємодіяти із базою одразу після підключення файлу, що особливо зручно в середовищах розробки або при створенні застосунків, які передбачають локальне розміщення даних. Підтримка транзакцій, зовнішніх ключів та базових обмежень забезпечує достатній рівень структурної цілісності. Крім того, ця СУБД сумісна з багатьма мовами програмування й технологіями, включно з JavaScript-орієнтованими фреймворками для клієнтських застосунків, що дає змогу гнучко реалізовувати архітектуру повного циклу.

Ще одним фактором, що визначає доцільність використання SQLite, є її компактність. У разі потреби перенесення системи, оновлення чи створення резервної копії, розробнику достатньо мати лише один файл, який містить усі структурні й інформаційні компоненти бази. Це спрощує процеси супроводу, тестування та відновлення, особливо в ситуаціях, коли не передбачено централізовану IT-інфраструктуру або коли застосунок призначено для розміщення на простому хостингу.

Загальна структура бази спроєктована з дотриманням принципів логічної нормалізації. Дані поділені за тематичними категоріями, з чітко встановленими зв’язками між сутностями, що дозволяє зменшити дублювання та забезпечити контроль за оновленням даних. При цьому структура лишається придатною до розширення — її можна адаптувати під нові розділи або типи інформації без значних змін у базовій логіці.

SQLite у межах цієї системи виконує функцію локального механізму збереження, що підходить для середовищ із невисоким навантаженням,

обмеженою кількістю активних користувачів та фокусом на контентну складову. Її застосування дозволяє зосередитись на реалізації бізнес-функцій, дизайну інтерфейсу й інтеграції з клієнтською частиною без необхідності складної підтримки серверного середовища бази даних. Така модель сприяє гнучкому керуванню вмістом, ефективному використанню ресурсів і придатна для невеликих комерційних або інформаційних онлайн-платформ.

У межах розробки інформаційної системи для майбутнього блогу, використання SQLite також дає змогу спростити процеси взаємодії між клієнтським застосунком і внутрішнім сховищем даних. Оскільки система передбачає як візуалізацію товарів, так і публікацію статей, база даних має забезпечувати логічне розділення цих напрямів та водночас зберігати узгодженість між ними. Наприклад, зв’язок між контентом і адміністративною частиною керується внутрішніми ідентифікаторами, а вся структура таблиць дозволяє системі залишатися керованою навіть при розширенні кількості товарних позицій або збільшенні активності користувачів, які залишають замовлення чи підписуються на розсилку.

Незважаючи на свою вбудовану архітектуру, SQLite дозволяє забезпечити базові механізми контролю даних — зокрема, можна використовувати обмеження цілісності, первинні й зовнішні ключі, а також комбіновані умови, що задаються при створенні таблиць. Це дозволяє обмежити потенційні логічні помилки вже на рівні структури бази, до того як дані потрапляють у застосунок. У середовищах, де можливість людського втручання в структуру або дані обмежена, така перевага дозволяє зменшити кількість помилок у продуктивному середовищі.

Серед додаткових технічних можливостей варто згадати простоту створення копій бази — завдяки файловій структурі достатньо лише дублювати файл SQLite, щоби здійснити резервне копіювання або протестувати зміни у захищеному середовищі. Це особливо зручно в розробницькому циклі, де часто виникає потреба у збереженні попередніх версій або створенні паралельних екземплярів системи.

Таким чином, використання SQLite у системі для "Зеленої Ноти" є виправданим вибором у контексті невеликого онлайн-магазину з контентною складовою. Система отримує базу даних, що не потребує складного обслуговування або адміністрування, водночас залишаючись достатньо гнучкою для реалізації всіх базових бізнес-процесів. Простота роботи з нею, інтегрованість у застосунок і підтримка основних функцій SQL дають змогу реалізувати весь необхідний функціонал без суттєвих обмежень для цільового сценарію використання.

3.2. Розробка бази даних застосунку

Забезпечення контрольованого доступу до адміністративного інтерфейсу веб-сайту передбачає створення окремого механізму зберігання інформації про уповноважених користувачів. Таблиця users (табл. 3.1) виконує цю функцію у системі даних проєкту «Зелена Нота», фіксуючи базову облікову інформацію, яка дає змогу автентифікувати та ідентифікувати осіб, що здійснюють операції з вмістом платформи. Це стосується публікації дописів у блозі, модерації замовлень, редагування товарних позицій і категорій [Додаток К].

Розробка такої таблиці базується на необхідності мати централізований і уніфікований реєстр користувачів із можливістю обмеження або розширення їх доступу до ресурсів системи. Основним ідентифікатором у структурі є поле id, яке автоматично збільшується з кожним новим записом, що забезпечує простий та надійний спосіб відстеження змін. Для унікального входу використовується поле username, а поле password передбачене для збереження зашифрованого значення пароля, що мінімізує ризики доступу до конфіденційних даних у разі несанкціонованого доступу до бази.

Поле email надає додатковий канал зв’язку або ідентифікації, зокрема в контексті можливого відновлення доступу. Текстове поле created_at дає змогу зафіксувати дату створення облікового запису, що корисно для

адміністрування або проведення журналювання змін. Логічне поле is_active дозволяє реалізувати кероване блокування або активацію доступу без необхідності видалення облікових записів, що особливо актуально в умовах тривалого життєвого циклу системи.

У перспективі структура таблиці залишається відкритою до розширення, зокрема шляхом додавання рівнів доступу, ролей або привілеїв, однак поточна реалізація достатньо повно охоплює основні потреби адміністрування ресурсу у його базовій версії.

Таблиця 3.1

Опис таблиці users

Поле	Тип даних	NULL	Опис
id	INTEGER	NOT NULL	Унікальний ідентифікатор користувача (первинний ключ)
username	TEXT	NOT NULL	Унікальне ім’я користувача для входу в систему
password	TEXT	NOT NULL	Хешований пароль для автентифікації
email	TEXT	NULL	Електронна адреса, асоційована з обліковим записом
created_at	TEXT	NULL	Дата та час створення облікового запису
is_active	BOOLEAN	NULL	Статус активності облікового запису (1 — активний, 0 — ні)

Інформаційна складова сайту магазину «Зелена Нота» реалізується через функціональність блогу, що дає змогу публікувати тематичні статті, рекомендації щодо догляду за кімнатними рослинами, новини про надходження товарів і сезонні поради. Такий формат не лише підтримує інтерес цільової аудиторії, але й сприяє просуванню товарів, покращенню SEO-показників та підвищенню рівня довіри до бренду.

Таблиця posts (табл. 3.2) відповідає за зберігання контенту, пов’язаного з кожним блог-дописом. Основна логіка реалізується через типове розділення на заголовок (title) та текстову частину (content), що дозволяє ефективно структурувати дані для виведення на сторінках сайту. Поле image_url забезпечує можливість ілюстративного супроводу допису, підвищуючи візуальну привабливість контенту.

Кожен допис прив’язується до конкретного користувача, який є його автором. Для цього передбачено поле author_id, що виступає зовнішнім ключем і формує зв’язок з таблицею users. Таким чином реалізується базовий механізм відстеження авторства та, за потреби, дозволяє застосовувати фільтрацію або сортування дописів за адміністраторами.

Як і в інших таблицях проєкту, застосовується поле created_at, що фіксує час створення запису. Це дозволяє впорядковувати матеріали у хронологічному порядку, відображати найновіші публікації або формувати історичну аналітику активності.

Загалом, таблиця posts реалізує одну з ключових функцій контентного наповнення сайту, підтримуючи комунікацію з аудиторією та надаючи простір для публічного представлення експертизи магазину.

Таблиця 3.2

Опис таблиці posts

Поле	Тип даних	NULL	Опис
1	2	3	4
id	INTEGER	NOT NULL	Унікальний ідентифікатор допису (первинний ключ)
title	TEXT	NOT NULL	Заголовок допису
content	TEXT	NOT NULL	Основний текст публікації
image_url	TEXT	NULL	Посилання на ілюстрацію або обкладинку статті

Продовження табл. 3.2

1	2	3	4
created_at	TEXT	NULL	Дата та час створення публікації
author_id	INTEGER	NOT NULL	Зовнішній ключ, що вказує на автора публікації (user.id)

Таблиця products (табл. 3.3) є центральною в структурі бази даних магазину, оскільки вона містить інформацію про асортимент товарів — кімнатні рослини в горщиках, які доступні для замовлення. Ця таблиця зберігає базові характеристики кожного продукту, такі як назва, опис, ціна та наявність, що є необхідними для формування каталогу і здійснення комерційних операцій.

Реалізація поля category_id як зовнішнього ключа дозволяє організувати товари за категоріями, що спрощує навігацію по каталогу і фільтрацію асортименту за різними ознаками. Важливим елементом є також поле is_active, яке визначає, чи доступний товар для замовлення, що дає змогу тимчасово приховувати товари без їх видалення з бази.

Включення дати створення (created_at) допомагає відслідковувати появу нових позицій у каталозі, а поле quantity — кількість товару на складі, що використовується для контролю запасів і управління замовленнями.

Загалом, таблиця products формує основу для представлення товарів у магазині та є базою для більшості бізнес-процесів, пов’язаних з продажем.

Таблиця 3.3

Опис таблиці products

Поле	Тип даних	NULL	Опис
id	INTEGER	NOT NULL	Унікальний ідентифікатор товару (первинний ключ)
name	TEXT	NOT NULL	Назва товару
description	TEXT	NULL	Опис товару
price	REAL	NOT NULL	Вартість товару
image_url	TEXT	NULL	Посилання на зображення товару
quantity	INTEGER	NULL	Кількість товару на складі
category_id	INTEGER	NULL	Зовнішній ключ, що вказує на категорію товару
created_at	TEXT	NULL	Дата та час додавання товару до бази
is_active	BOOLEAN	NULL	Статус активності товару (доступний/прихований)

Таблиця product_categories (табл. 3.4) служить для класифікації товарів на логічні групи, що допомагає структурувати каталог та спрощує процес пошуку і фільтрації товарів. Використання окремої таблиці для категорій дозволяє централізовано керувати цими групами, змінювати їхні назви або опис без втручання у дані товарів. Таке розмежування сприяє підтримці цілісності даних і підвищує гнучкість системи.

Крім назви категорії, у таблиці передбачено поле для опису, що дає змогу зберігати додаткову інформацію про кожну групу товарів, наприклад, особливості рослин, умови догляду чи стиль оформлення. Це може використовуватись як у адміністративному інтерфейсі, так і на публічній частині сайту для допомоги користувачам у виборі.

Відмітка часу створення запису (created_at) дозволяє відстежувати історію додавання категорій і підтримувати контроль версій у разі потреби.

Таблиця 3.4

Опис таблиці product_categories

Поле	Тип даних	NULL	Опис
id	INTEGER	NOT NULL	Унікальний ідентифікатор категорії (первинний ключ)
name	TEXT	NOT NULL	Назва категорії
description	TEXT	NULL	Опис категорії
created_at	TEXT	NULL	Дата та час створення запису

Таблиця orders (табл. 3.5) призначена для зберігання інформації про замовлення, зроблені покупцями магазину. Вона містить дані про сумарну вартість замовлення, контактні відомості замовника, статус обробки замовлення, а також час створення запису. Ця таблиця є центральною у процесі управління комерційною діяльністю, оскільки агрегує ключові відомості, необхідні для подальшої обробки, доставки і звітності.

Поля, що відповідають за контактні дані покупця, дозволяють ідентифікувати клієнта та забезпечити комунікацію з ним. Стан замовлення, який зберігається у полі status, дає можливість відстежувати поточний етап виконання, наприклад, «pending» (очікує обробки), «completed» (виконано) або «canceled» (скасовано).

Вказівка часу створення дозволяє аналізувати обсяги продажів за період, а також контролювати оперативність обробки заявок.

Таблиця 3.5

Опис таблиці orders

Поле	Тип даних	NULL	Опис
id	INTEGER	NOT NULL	Унікальний ідентифікатор замовлення
total_price	REAL	NOT NULL	Загальна сума замовлення
order_text	TEXT	NULL	Додаткова інформація або коментар
customer_name	TEXT	NOT NULL	Ім’я замовника
customer_phone	TEXT	NOT NULL	Телефон замовника
customer_address	TEXT	NOT NULL	Адреса доставки
created_at	TEXT	NULL	Дата та час створення замовлення
status	TEXT	NULL	Поточний статус замовлення

Таблиця order_items (табл. 3.6) містить детальну інформацію про товари, включені до конкретного замовлення. Вона забезпечує зв’язок між замовленнями та товарами, фіксуючи кількість кожного найменування та ціну одиниці на момент замовлення. Така структура дозволяє вести облік асортименту, що був придбаний, а також формувати точні звіти щодо продажів.

Ця таблиця є проміжною в реляційній моделі, яка реалізує зв’язок багато-до-багатьох між замовленнями та продуктами. Зовнішні ключі забезпечують цілісність даних, гарантують коректність посилань на існуючі замовлення та товари.

Завдяки поділу інформації на заголовок замовлення і його складові позиції, зберігається гнучкість і спрощується оновлення даних. Наприклад,

зміни в каталозі товарів не впливають на вже зроблені замовлення, адже ціни і кількість зберігаються локально у цій таблиці.

Таблиця 3.6

Опис таблиці orders_items

Поле	Тип даних	NULL	Опис
id	INTEGER	NOT NULL	Унікальний ідентифікатор рядка
order_id	INTEGER	NOT NULL	Ідентифікатор замовлення
product_id	INTEGER	NOT NULL	Ідентифікатор товару
quantity	INTEGER	NOT NULL	Кількість товару в замовленні
unit_price	REAL	NOT NULL	Ціна за одиницю товару

Усі описані таблиці формують узгоджену логічну модель, що дозволяє систематизовано зберігати інформацію про товари, їхні категорії, публікації, користувачів і підписників. Завдяки використанню зовнішніх ключів забезпечується взаємозв’язок між сутностями, що сприяє цілісності даних та спрощує їх обробку. Така структура створює основу для стабільної роботи сайту, підтримує розширення функціоналу та адаптацію до змін у вимогах системи.

Висновки до розділу 3

У цьому розділі описано структуру бази даних, побудовану на основі реляційного підходу. Таблиці та зв’язки між ними сформовані так, щоб забезпечити впорядковане зберігання інформації, уникати дублювання й підтримувати логічну узгодженість. Усі сутності спроєктовані з урахуванням ролей, які вони відіграють у системі, а також принципів нормалізації.

Представлена модель орієнтована на інтеграцію з іншими частинами цифрового середовища, де дані активно змінюються, взаємодіють і відображаються.

РОЗДІЛ IV
РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1. Десктопний застосунок на C#

У рамках розробки інтерактивного навчального застосунку, присвяченого кімнатним рослинам, було створено програму з графічним інтерфейсом на платформі Windows Forms із використанням бібліотеки OpenTK для роботи з OpenGL. Основна мета застосунку полягає у реалізації інтерактивної гри-квізу, яка містить меню вибору рівня [Додаток А] та ігрове поле з графічним представленням кімнати, рослин та зон освітлення [Додаток Б].

Основним вікном програми є клас MainForm, що наслідується від Form Windows Forms. В якості контейнера для графіки використовується OpenGL-контрол GLControl, ініціалізований із версією OpenGL 3.3 і сумісним профілем. Такий вибір забезпечує доступ до сучасних можливостей OpenGL для 2D-візуалізації та дозволяє ефективно керувати рендерингом в межах стандартного віконного застосунку.

У конструкторі головної форми здійснюється налаштування GLControl розташування, розміру, події рендерингу і взаємодії з користувачем та ініціалізація основних компонентів гри: меню (MenuScreen), ігрового поля (GameScreen) і текстового рендерера (TextRenderer).

Залежно від поточного стану гри, що зберігається у змінній currentState, відбувається відображення відповідного екрану — або меню вибору рівня (Рис. 4.1.), або ігрового поля (Рис. 4.2.). Обробники подій миші делегують керування відповідним елементам інтерфейсу, що забезпечує інтерактивність: вибір рівня в меню, перетягування рослини в межах ігрового поля тощо.

Рис. 4.1. Вибір рівня

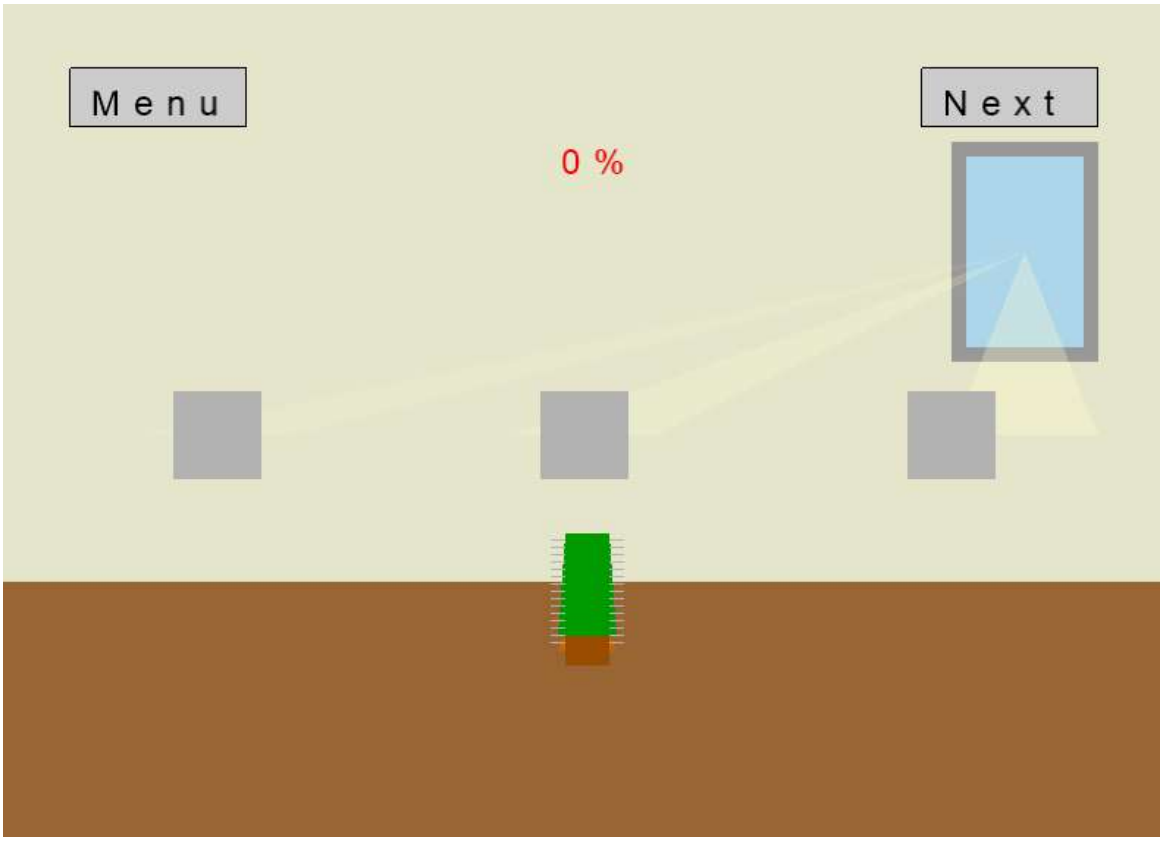

Рис. 4.2. Ігрове поле

Клас GameScreen реалізує основну ігрову логіку. Для кожного рівня гри визначено унікальну рослину з певними параметрами оптимального освітлення. Графічне відображення сцени включає інтер'єр кімнати, вікно з променями світла, зони розміщення рослин, самі рослини та елементи керування. Реалізовано механізм перетягування рослини за допомогою миші, а також оцінку якості освітлення на основі відстані між рослиною та освітлювальною зоною. Результат цієї оцінки відображається текстовими повідомленнями.

Клас MenuScreen відповідає за відображення головного меню гри. Візуально меню виконано із використанням градієнтного фону, декоративних зображень кімнатних рослин по краях екрану та інтерактивних кнопок вибору рівня. Кожна кнопка має власну логіку відображення та обробки подій кліку, що забезпечує інтуїтивний вибір рівня і запуск відповідної частини гри.

Для виведення текстових повідомлень використовується клас TextRenderer, який створює текстуру шрифту з усіма ASCII-символами за допомогою GDI+. Ця текстура надалі використовується для малювання символів OpenGL, що дозволяє ефективно відображати текст поверх графіки з підтримкою кольорів, розмірів і позиціонування.

Кактус — це сукулентна рослина, яка зображена в горщику з характерним масивним зеленим тілом у формі квадрата зі скошеними кутами [2]. Основна частина кактуса має насичений темно-зелений колір, що підкреслює його стійкість та витривалість. Додатково для передачі текстури рослини нанесені численні тонкі сірі лінії — іголки, які рівномірно розташовані вздовж бокових сторін, символізуючи захисний покрив кактуса (Рис. 4.3.).

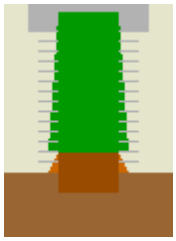

Рис. 4.3. Зображення кактусу

Орхідея відтворена як ніжна рослина в горщику з тонким вертикальним стеблом темно-зеленого кольору. Над стеблом розташовані три яскраві квітки світло-фіолетового та жовтого відтінку, які розташовані по вертикалі з невеликим інтервалом. Квіти мають округлу форму, що характерна для орхідей, і контрастує з тонким стеблом, підкреслюючи елегантність і делікатність цієї рослини (Рис. 4.4.).

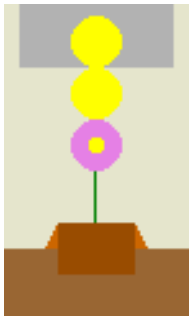

Рис. 4.4. Зображення орхідеї

Фіалка представлена у вигляді компактної рослини в горщику з розташованими навколо центральної точки шістьма листками, які розкидаються у вигляді зірки. Листя мають темно-зелений відтінок із природним вигином. Над листям розміщено чотири невеликі квітки яскраво-фіолетового кольору, розташовані по колу з невеликим нахилом, що створює ефект пишної і густої рослини (Рис. 4.5.).

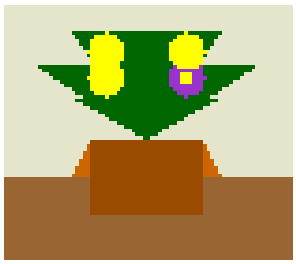

Рис. 4.5. Фіалка

Авокадо зображено як рослину з міцним дерев’янистим стовбуром коричневого кольору, що виходить із горщика. Від стовбура відходить кілька пар широких листків зеленого кольору, розташованих симетрично з обох боків. Листя має трикутну форму та поступово зменшується у розмірі догори, що створює ефект здорової і гармонійної рослини з добре вираженою структурою (Рис. 4.6.).

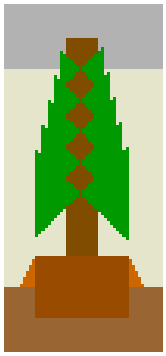

Рис. 4.6. Авокадо

Монстера — це декоративна рослина, яка відображена з горщиком та тонким стеблом зеленого кольору. Від стебла відходять три великі широкі листки, кожен із яких має характерну форму з вирізами по краях, що створює впізнаваний силует монстери. Листя розташовані рівномірно по колу та поєднують насичений зелений колір із природними вигинами, підкреслюючи тривимірність і декоративність рослини (Рис. 4.7.).

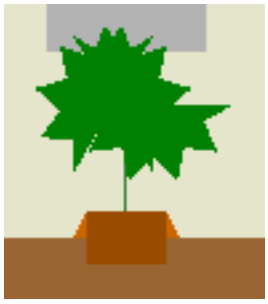

Рис. 4.7. Монстера

Ігрове поле містить три джерела світла — промені, та три спеціальні столи, на які користувач може розміщувати рослину. Кожна зона характеризується різним рівнем освітленості, який визначає, наскільки умови підходять для конкретної рослини. В залежності від положення рослини на цих столах та близькості до променів світла, формується відповідний відсоток освітлення (Рис. 4.8.): 0%, 20%, 40%, 60%, 80% або 100%.

100% — ідеальне освітлення. Рослина розміщена на столі, що безпосередньо знаходиться під променем світла. Колір тексту в індикаторі змінюється на яскраво-зелений, що сигналізує про оптимальні умови росту.

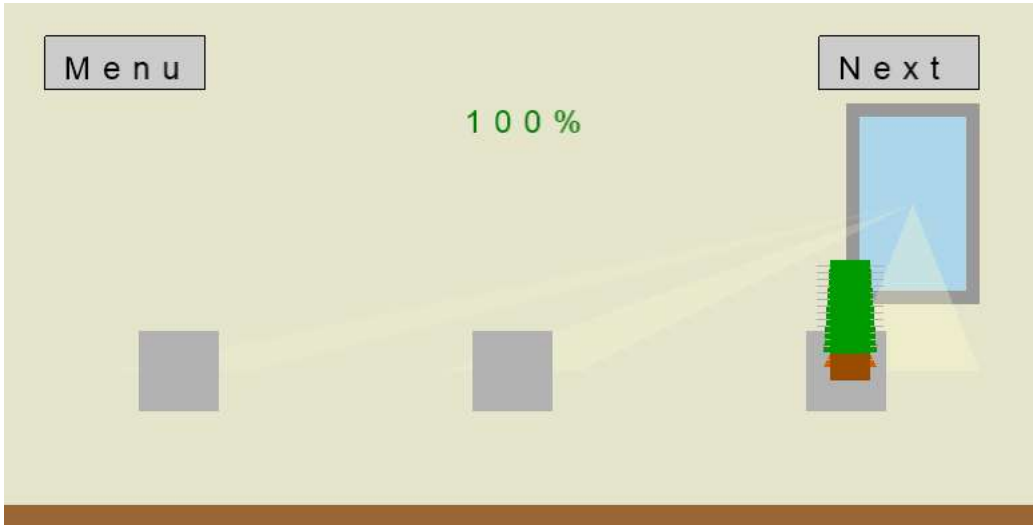

Рис. 4.8. Індекатор рівня освітлення

80% — дуже хороше освітлення. Рослина знаходиться поруч із променем або на столі з хорошим доступом світла. Текст підсвічується світло-зеленим кольором.

60% — прийнятне освітлення, але не ідеальне. Рослина розташована трохи далі від променя, можливо, на столі зі слабшим освітленням. Колір тексту стає жовтим, попереджаючи про можливість покращення.

40% — недостатнє освітлення. Рослина розміщена далеко від променів, або на столі з поганим доступом світла. Текст підсвічується помаранчевим кольором, що натякає на потребу у зміні позиції.

20% — дуже слабке освітлення. Рослина майже позбавлена променів світла, що небажано для її розвитку. Колір тексту червоний, сигналізуючи про небезпеку.

0% — відсутність освітлення. Рослина знаходиться поза зонами світла або на неправильному столі. Текст стає яскраво-червоним, підкреслюючи критичний стан.

Відсоток освітлення автоматично оновлюється у реальному часі при переміщенні рослини, відображається у вигляді текстового повідомлення у вікні гри. Паралельно з цим змінюється колір тексту, що дає гравцю візуальний сигнал про якість освітлення і дозволяє приймати обґрунтовані рішення для досягнення оптимального стану рослини.

4.2. Створення клієнтської частини сайту

Для реалізації клієнтської частини сайту, який поєднує функціональність блогу та інтернет-магазину, було обрано фреймворк React, що дозволяє створювати динамічні односторінкові застосунки. У проєкті також використовуються бібліотеки Joy UI для побудови адаптивного інтерфейсу, та для анімацій – Framer Motion.

Проєкт організований у вигляді стандартної структури React-додатку. Основна логіка розділена за призначенням по папках:

components/ — багаторазові компоненти інтерфейсу, як-от картки товарів, модальні вікна, елементи блогу;

- pages/ — сторінки сайту (головна, блог, магазин, тощо); router/ — маршрутизація застосунку;
- theme/ — кастомізація теми інтерфейсу;
- apollo/ — інтеграція з GraphQL;
- assets/ — графічні ресурси (наприклад, логотип);
- index.js, App.jsx — вхідні точки застосунку.

Початкове завантаження програми здійснюється через файл index.js, який рендерить компонент App. У App.jsx описана основна структура застосунку, зокрема підключення роутера та обгортки для теми.

Файл router/AppRouter.jsx містить визначення маршрутової логіки, реалізованої за допомогою бібліотеки react-router-dom. Основні маршрути включають:

- / — головна сторінка (Home);
- /about — сторінка з описом ідеї блогу та кнопкою для завантаження гри;
- /blog — перегляд публікацій;
- /create-post — сторінка створення нового поста;
- /shop — магазин кімнатних рослин;
- * — обробка невизначених маршрутів (NotFound).

Маршрути огортаються в компонент Layout, який забезпечує єдину структуру для всіх сторінок (хедер, футер, модальні компоненти).

Сайт-блог «Зелена нота» має кілька основних сторінок, кожна з яких виконує конкретну функцію в межах загальної концепції поєднання блогу та онлайн-магазину. Головна сторінка, реалізована у файлі Home/index.jsx, є першим екраном, який бачить користувач після переходу на сайт. Вона містить візуально виразне фонове зображення, слоган проєкту, а також інтерактивну кнопку, що спрямовує відвідувача далі до тематичних розділів (Рис. 4.9).

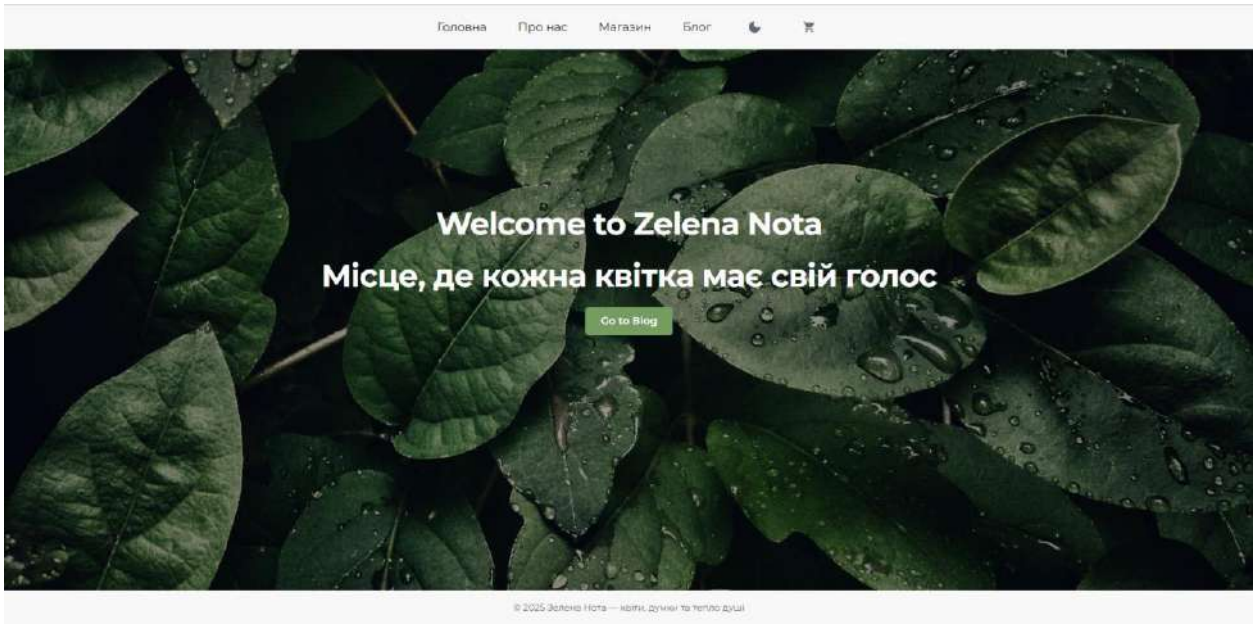

Рис. 4.9. Головна сторінка сайту

Сторінка About (About/index.jsx) присвячена опису проєкту, його ідеї, натхнення та цільової аудиторії. Тут користувач може дізнатись більше про місію «Зеленої ноти» — поєднання любові до рослин, дизайну та цифрової комунікації. Окрім текстового блоку з описом, на цій сторінці реалізовано інтерактивну кнопку, яка дозволяє завантажити окрему гру — навчальний застосунок про кімнатні рослини, створений у попередньому етапі проєкту. Завантаження відбувається через пряме посилання, вбудоване у кнопку, яка виділена візуально і органічно вписана в загальний стиль сторінки. Оформлення головної сторінки побудовано з використанням компонентів бібліотеки Joy UI та елементів анімації за допомогою Framer Motion. Завдяки

цьому елементи з’являються поступово при завантаженні сторінки, створюючи динамічне і привабливе враження. Розміщення секцій чергується — спочатку ліворуч, потім праворуч, потім знову ліворуч і так далі, що надає сторінці ритмічну композицію (Рис. 4.10).

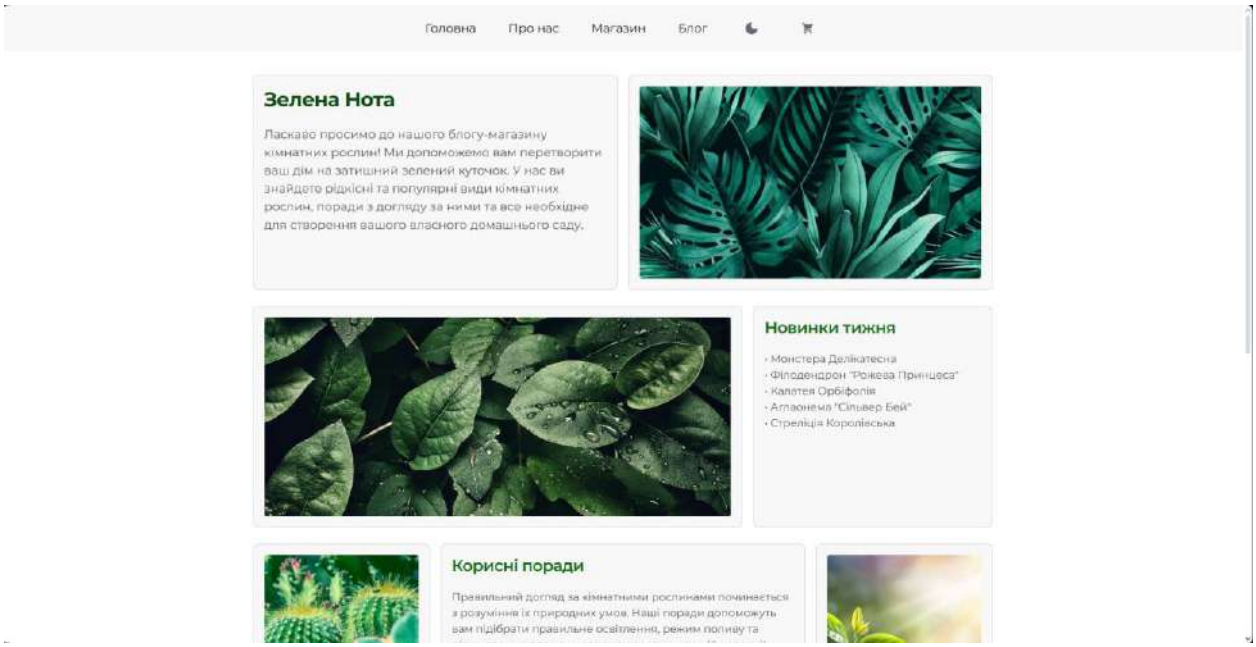

Рис. 4.10. Сторінка сайту About

Сторінка Blog (Blog/index.jsx) призначена для перегляду публікацій, які можуть містити текстовий контент і зображення. Дані для блогу отримуються через GraphQL-запити з серверної частини сайту. Кожен пост виводиться за допомогою компонента PostItem, який відображає заголовок, короткий опис та, за наявності, прев’ю зображення. Таким чином формується стрічка записів, які користувач може швидко переглянути (Рис. 4.11.).

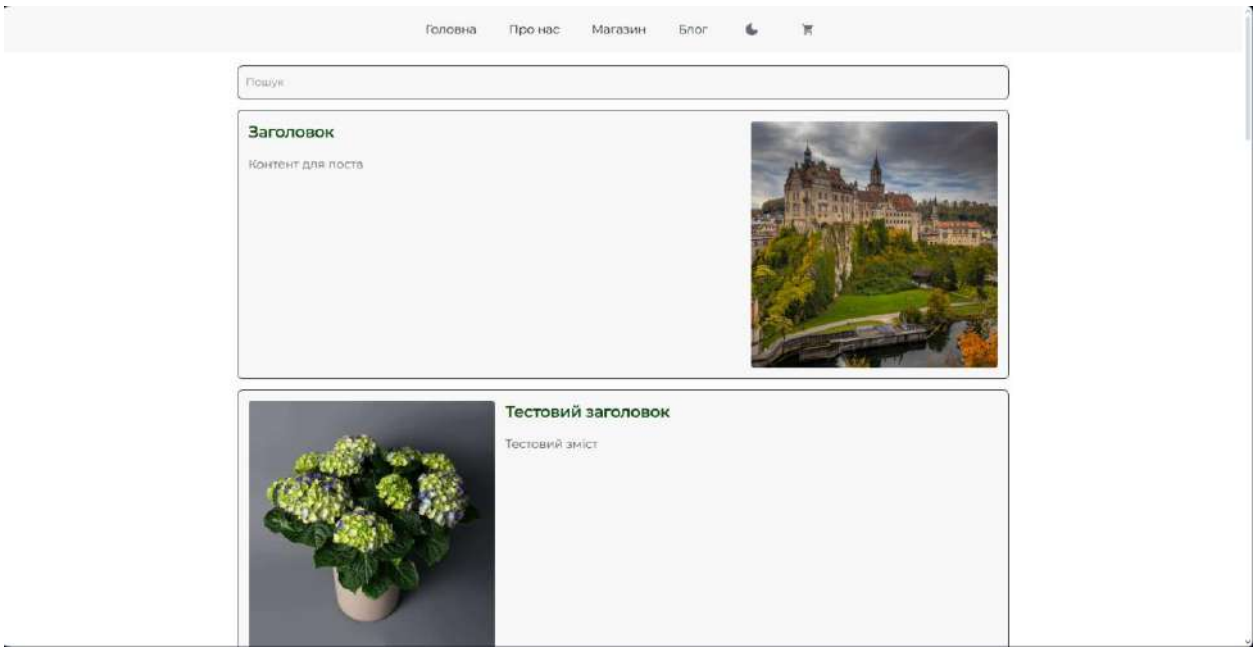

Рис. 4.11. Вигляд поста в блозі

Сторінка CreatePost (CreatePost/index.jsx) дає змогу створювати нові публікації до блогу. Інтерфейс сторінки містить поля для введення заголовка, змісту публікації та необов’язкового посилання на зображення. Після заповнення цих полів користувач натискає кнопку, що викликає GraphQL-запит для створення запису на сервері. Наразі ця функціональність доступна усім користувачам, однак у майбутньому планується реалізувати механізм авторизації, який обмежуватиме доступ до створення постів лише для зареєстрованих та підтверджених користувачів. Відповідна таблиця для користувачів уже створена у базі даних. (Рис. 4.12.). Структура сторінки побудована просто та інтуїтивно, з акцентом на зручність введення інформації. При успішному або неуспішному створенні поста з’являється відповідне повідомлення користувачеві, яке дозволяє чітко зрозуміти, що відбувається з постом після натискання кнопки “Створити пост”. (Рис. 4.13.).

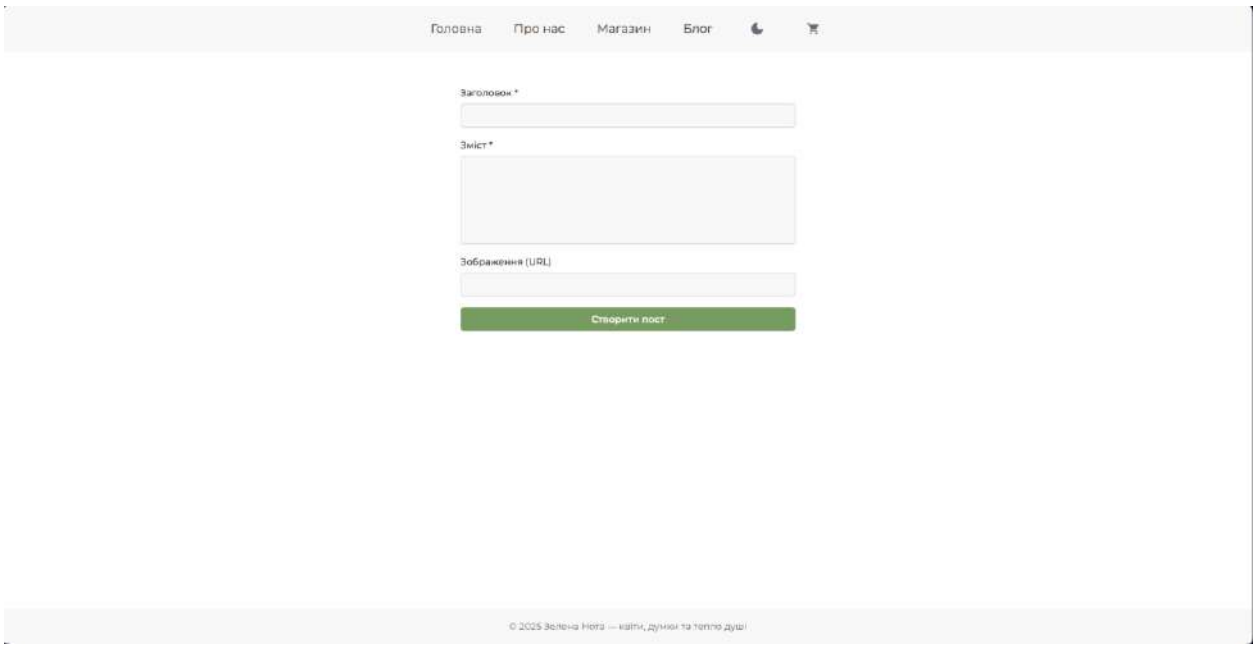

Рис. 4.12. Форма створення публікації

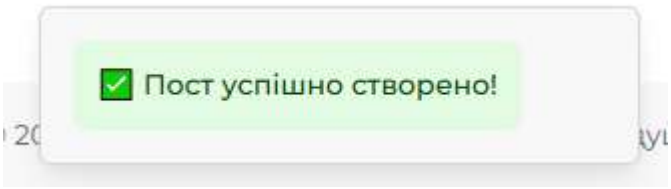

Рис. 4.13. Повідомлення про успішне створення поста

Сторінка Shop (Shop/index.jsx) є розділом електронної комерції, де відвідувач може переглядати кімнатні рослини, додавати їх у кошик та оформляти замовлення. Кожна рослина відображається за допомогою компонента ProductCard, в якому подано назву, фото, короткий опис і ціну. Додавання до кошика відбувається натисканням на кнопку, яка інтегрується з контекстом кошика, реалізованим у CartContext.js. Модальні вікна ShoppingCartModal і CheckoutModal відкриваються при перегляді кошика або оформленні покупки відповідно, надаючи користувачу зручну взаємодію без перенаправлення між сторінками (Рис. 4.14.).

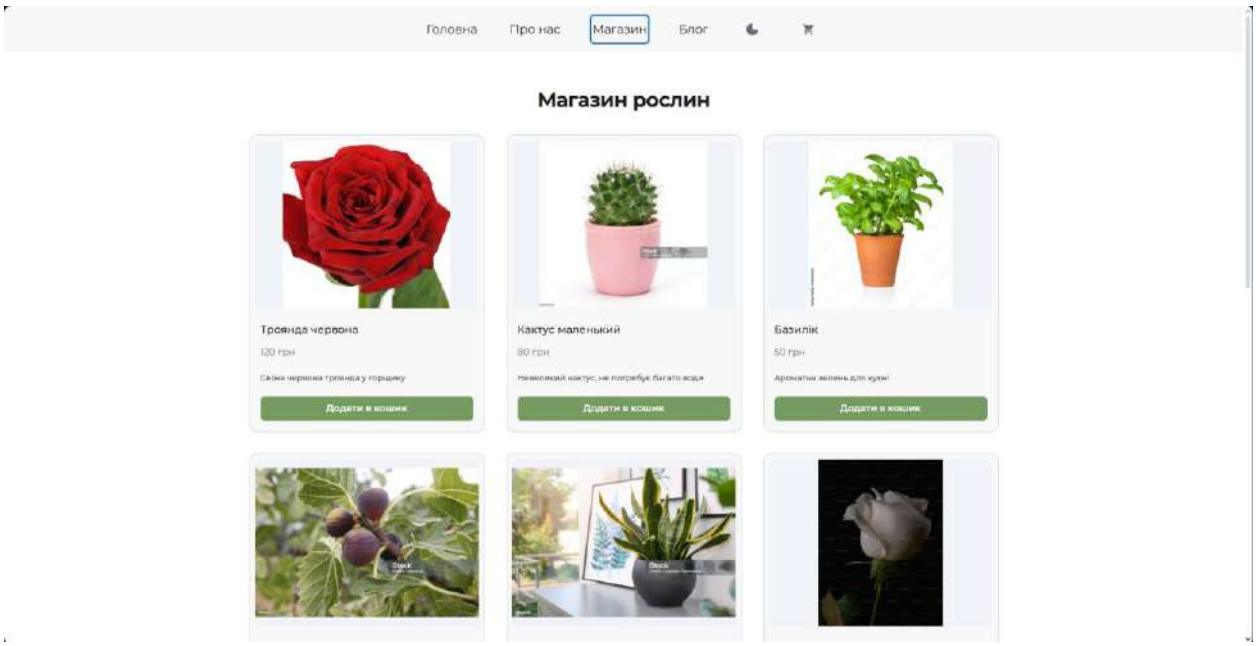

Рис. 4.14. Сторінка магазину з товарами

Крім основних сторінок, у проєкті реалізовано також сторінку обробки помилок — сторінку 404 (NotFound/index.jsx). Вона відображається у випадку, якщо користувач переходить за неіснуючим маршрутом або помилково вводить адресу. Інтерфейс сторінки витримано в загальному стилі сайту — мінімалістичне повідомлення про те, що сторінку не знайдено, а також кнопка для повернення на головну. Це забезпечує більш дружню взаємодію з користувачем, уникаючи стандартного повідомлення браузера про помилку навігації. Сторінка інтегрована через маршрутизатор, що знаходиться у файлі AppRouter.jsx, де вона є fallback-компонентом для всіх невизначених шляхів (Рис. 4.15).

Рис. 4.15. Сторінка помилки 404

Вся маршрутизація в застосунку здійснюється через компонент AppRouter.jsx, який реалізує логіку переходу між сторінками. Для збереження єдиної структури сайту використовується окремий компонент Layout.jsx, в якому відображаються Header [Додаток В] та Footer [Додаток Г], незалежно від вмісту сторінки. Таким чином, заголовок сайту, перемикач теми, іконка кошика та нижній колонтитул завжди залишаються на місці, створюючи консистентне користувацьке враження. Сам роутер вміщує посилання на усі компоненти сайту [Додаток Д].

Навігаційна панель, реалізована у компоненті Header, дозволяє переходити між основними розділами сайту: головною сторінкою, магазином, блогом, формою створення публікацій та сторінкою «Про нас». Крім того, у заголовку розташовано перемикач теми ThemeSwitcher, який дозволяє перемикатися між світлою та темною темами, використовуючи можливості бібліотеки Joy UI [7]. Перемикання теми відбувається миттєво, без перезавантаження сторінки. Зміна теми поширюється на весь інтерфейс завдяки використанню компонента ThemeProvider, який охоплює застосунок зверху (Рис. 4.16).

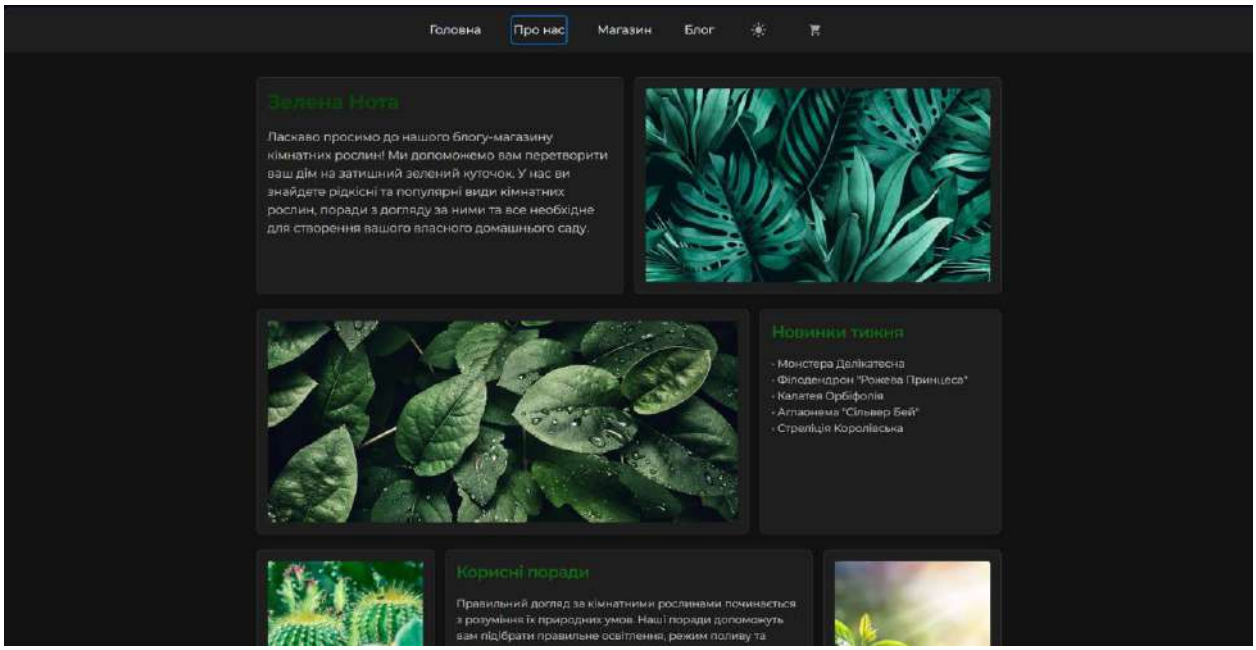

Рис. 4.16. Темна тема сторінки “Про нас”

Компонент Footer відповідає за виведення інформації в нижній частині сторінки. Його вміст може включати посилання, контактну інформацію або авторські права. Він фіксовано присутній на всіх сторінках, будучи частиною компонента Layout.

Візуальний стиль сайту базується виключно на компонентах бібліотеки Joy UI, яка забезпечує елегантний, легкий дизайн і високий рівень доступності. Кожен візуальний елемент — кнопка, форма, картка товару — оформлений відповідно до гайдлайнів цієї бібліотеки, що забезпечує зручність користування та цілісне сприйняття.

Компонентом інтерфейсу, що формує візуальне представлення товарів у магазині - є ProductCard. Він використовується на сторінці Shop для відображення кожної окремої рослини, що доступна для покупки. Картка містить зображення квітки в горщику, назву, короткий опис та ціну. Також реалізовано кнопку додавання до кошика, що одразу викликає відповідну функцію з контексту кошика.

Усі товари, що були додані до кошика, виводяться у вигляді списку за допомогою компонента CartItemCard. Цей компонент відображає назву, кількість і загальну вартість конкретної позиції. Також присутні кнопки збільшення або зменшення кількості та кнопка видалення товару з кошика. Компонент розрахований на використання як у модальному вікні кошика (ShoppingCartModal), так і у вікні оформлення замовлення (CheckoutModal), забезпечуючи повторне використання коду та візуальну узгодженість між різними частинами інтерфейсу.

При натисканні на іконку кошика, що розташована у заголовку сайту, відкривається компонент ShoppingCartModal (Рис. 4.17.). Це модальне вікно з затемненням фону, у якому зібрані всі додані товари, виводиться загальна сума замовлення та присутня кнопка переходу до оформлення покупки. У випадку підтвердження — відкривається CheckoutModal (Рис. 4.18.), у якому користувач вводить дані для замовлення (ім’я, контактну інформацію, тощо) та надсилає їх на сервер. Після цього кошик очищується, і з’являється

повідомлення про успішну покупку. Обидва модалі дотримуються стилістики решти сайту та є повністю адаптивними.

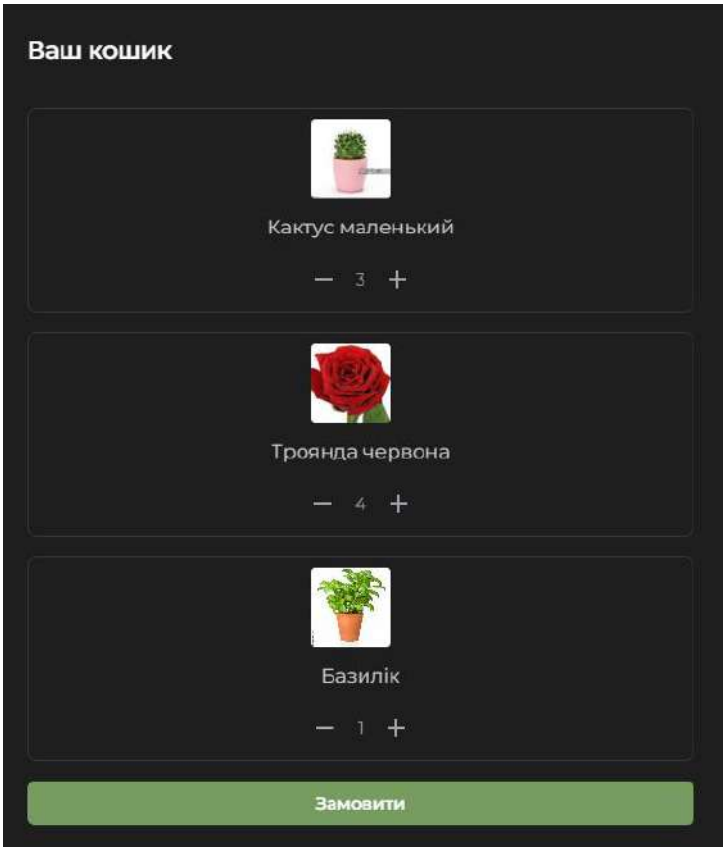

Рис. 4.17. Модальне вікно кошика

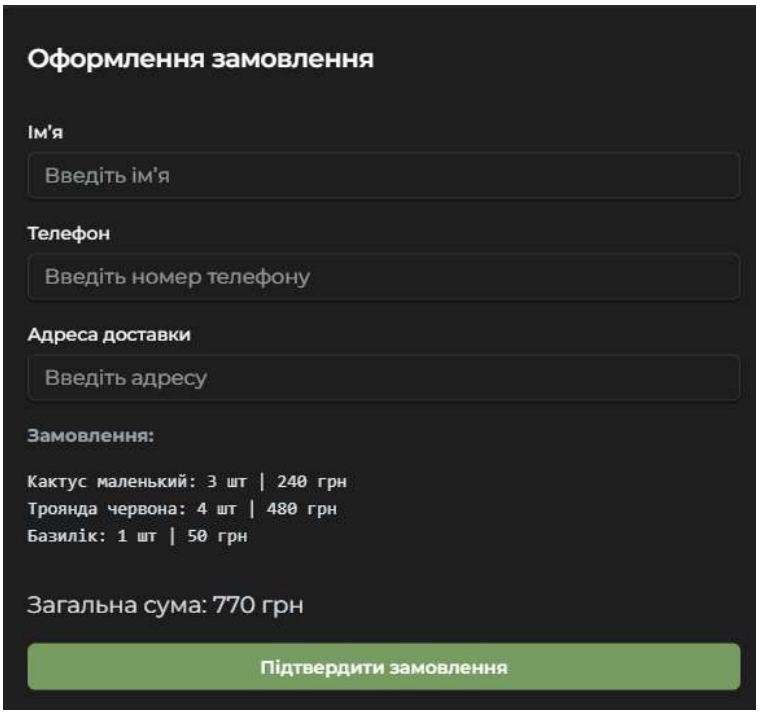

Рис. 4.18. Модальне вікно підтвердження замовлення

На сторінці About додаткову естетичну функцію виконує компонент AboutSwiper. Він реалізований із використанням сторонньої бібліотеки слайдерів і служить для візуального представлення зображень, пов’язаних з тематикою сайту — квітів у горщиках, оформлення інтер’єру, рослин у природному або штучному освітленні. Слайдер автоматично або вручну прокручує серію зображень, кожне з яких оформлене у стилі сайту: з легкими тінями, плавною анімацією та адаптивним відображенням. Компонент AboutSwiper дозволяє підсилити враження користувача від сторінки, зробити сайт більш живим, а також передати атмосферу, яку не завжди можливо відчути лише через текст або кнопки (Рис. 4.19).

Рис. 4.19. Слайдер на сторінці About

Розроблена клієнтська частина сайту-блогу «Зелена нота» поєднує в собі мінімалістичний та функціональний дизайн із зручною логікою навігації. Завдяки використанню бібліотеки Joy UI інтерфейс виглядає сучасно, м’яко та естетично. Компонентний підхід забезпечує гнучкість у повторному використанні логіки, а також дозволяє масштабувати структуру у випадку розширення функціональності. Переходи між сторінками реалізовані без перезавантаження, що забезпечує високу швидкість взаємодії. Інтеграція з GraphQL та використання Apollo забезпечує надійне отримання і публікацію даних, а вся архітектура клієнта побудована так, щоб надалі легко впровадити

додаткові функції — як-от авторизацію, фільтрацію товарів або більш розгорнуті пости. У поєднанні з серверною частиною клієнтська складова створює зручний, адаптивний і привабливий інтерфейс для кінцевого користувача.

4.3. Створення серверної частини сайту

Серверна частина сайту-блогу «Зелена нота» реалізована з використанням Node.js, бібліотеки Apollo Server для побудови GraphQL API [Додаток Л] та бази даних SQLite. Вся логіка зберігання та обробки даних організована у вигляді модульної структури, що дозволяє легко розширювати та підтримувати серверну частину, зберігаючи чистоту коду і його читаність.

Файл src/index.js є точкою входу в застосунок. У ньому створюється сервер Apollo, підключаються визначення типів GraphQL, резолвери та виконується запуск на вказаному порті. Конфігурація середовища, включно з портом або іншими приватними налаштуваннями, зберігається у файлі .env, що дозволяє розділити логіку програми від конфіденційної інформації.

База даних представлена у вигляді файлу database.sqlite, розташованого у папці data. Початкова структура бази описується у файлі schema.sql, який містить SQL-запити для створення таблиць: posts, products, orders, users тощо. Для первинного заповнення бази тестовими даними передбачено файл seedUser.js, який вставляє мінімальний набір записів для роботи функціоналу.

Усі запити обробляються за допомогою GraphQL-схеми, яка розміщена у файлі src/schema/schema.graphql. У ній визначено типи, запити та мутації, які можуть виконуватись з боку клієнта. Ця схема використовується у файлі typeDefs.js, який імпортується під час ініціалізації сервера.

Виконання запитів [Додаток И] реалізовано через резолвери, логіка яких поділена на queries та mutations. У папці src/resolvers/queries розміщені файли products.js, posts.js, orders.js та searchPost.js, кожен із яких відповідає за обробку конкретного типу даних. Наприклад, файл products.js повертає список

квіткових товарів, posts.js — усі публікації блогу, а searchPost.js дозволяє виконувати пошук постів за ключовими словами. Запити до замовлень обробляються окремо у orders.js, де реалізовано логіку отримання історії покупок.

Усі мутації згруповані в src/resolvers/mutations. Тут знаходиться createOrder.js — функція створення нового замовлення, яка записує в таблицю orders деталі замовника та вибрані товари, а також createPost.js, що реалізує логіку створення нової публікації в блозі.

Всі резолвери агрегуються у файлі src/resolvers/index.js, який об'єднує запити та мутації в єдиний об'єкт для передачі до Apollo Server. Така модульна структура дозволяє легко додавати нові типи даних і логіку обробки запитів, не змінюючи основні частини проєкту.

Реалізація серверної частини дозволяє клієнтському застосунку отримувати лише ті дані, які потрібні у конкретний момент, що зменшує навантаження на мережу та підвищує швидкодію. Використання GraphQL спрощує побудову запитів та дозволяє клієнту точно визначати, які поля потрібні. Структура бази даних та архітектура сервера побудовані з урахуванням подальшого розвитку проєкту — наприклад, можливості впровадження ролей користувачів, розширення типів постів, чи додавання нових операцій із товарами.

Висновки до розділу 4

У цьому розділі розглянуто архітектуру та реалізацію трьох основних складових застосунку: десктопної гри, клієнтської та серверної частини веб-сайту «Зелена нота».

Десктопний застосунок створено на Windows Forms з використанням OpenGL для реалізації інтерактивної гри про кімнатні рослини. Було описано ключові екрани, ігрову логіку, а також графічне відображення рослин і освітлення.

Клієнтська частина сайту реалізована на React із Joy UI, з акцентом на модульність, плавні переходи та зручність навігації.

Серверна частина побудована на Node.js із використанням GraphQL та SQLite. Усі основні запити та мутації розділені за призначенням, що забезпечує легку підтримку й розширення.

Загалом, реалізація проєкту демонструє цілісну взаємодію між різними платформами: настільною грою та веб-застосунком, що працюють як частини єдиної системи. Такий підхід дозволяє ефективно масштабувати функціонал і зберігати узгодженість у роботі з даними.

ВИСНОВКИ

В результаті виконання даної кваліфікаційної роботи було досягнуто поставленої мети — створено програмне забезпечення, яке сприяє кращому розумінню процесів догляду за кімнатними рослинами через поєднання інформаційного, комерційного та навчального компонентів. Проєкт охоплює створення веб-сайту з блогом, базовим інтернет-магазином та інтерактивною грою.

В результаті проведеної роботи можна зробити такі висновки:

1. На етапі проєктування веб-застосунку було визначено головну мету — створення інтерактивного ресурсу, що поєднує навчання, інформування та можливість замовлення товарів для догляду за кімнатними рослинами. Для цього було зібрано базу знань про популярні види рослин, реалізовано базовий функціонал онлайн-магазину, створено ігрову складову.

2. Реалізовано інформаційний модуль у вигляді блогу, який містить публікації з порадами щодо догляду за кімнатними рослинами. Забезпечено зручну навігацію між записами, пошук і можливість візуального ознайомлення з матеріалами. Це створює умови для поступового засвоєння знань у легкому форматі.

3. Створено клієнтську частину онлайн-магазину з можливістю перегляду товарів, формування замовлення та надсилання його на сервер. Інтерфейс забезпечує простоту взаємодії, а структура компонентів дозволяє розширювати магазин у майбутньому, додаючи нові категорії чи функціонал.

4. Розроблено ігровий компонент у форматі десктопного застосунку, який дозволяє моделювати розміщення рослин у кімнаті залежно від рівня освітлення. Це дозволяє користувачу на практиці зрозуміти вплив зовнішніх умов на здоров’я рослини, що робить процес навчання більш наочним та ефективним.

5. Уся система створена з урахуванням потреб цільової аудиторії — людей, які цікавляться озелененням простору, але не мають спеціальної освіти.

Завдяки цьому вона є доступною, інтуїтивно зрозумілою та здатною зацікавити як новачків, так і досвідчених користувачів.

Таким чином, бакалаврська робота реалізує поставлену мету і демонструє актуальність поєднання інформаційних технологій з популяризацією догляду за кімнатними рослинами. Створене програмне забезпечення дозволяє не лише отримувати корисні поради, а й навчатися доглядати за рослинами через взаємодію з віртуальним середовищем, що робить його ефективним засобом як пізнання, так і повсякденного використання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Шилдт Г. C# 10 і .NET 6: повний довідник для професіоналів / Г. Шилдт. — Київ : Діалектика, 2022. — 880 с.
2. Troelsen A., Japikse P. Pro C# 10 with .NET 6 / Andrew Troelsen, Philip Japikse. — Apress, 2022. — 1500 с.
3. Opentk.net — The Open Toolkit library [Електронний ресурс]. — Режим доступу: https://opentk.net/
4. Khronos Group. OpenGL 3.3 Specification [Електронний ресурс]. — Режим доступу: https://www.khronos.org/registry/OpenGL/specs/gl/glspec33.core.pdf
5. MSDN Documentation. Windows Forms Overview [Електронний ресурс]. — Режим доступу: https://learn.microsoft.com/en-us/dotnet/desktop/winforms/
6. Шевченко В. Графічне програмування засобами OpenGL. Навчальний посібник. — Київ : НТУУ «КПІ», 2020. — 124 с.
7. Хом’як В. Основи побудови графічних інтерфейсів користувача на C#. — Львів : Видавництво ЛНУ, 2019. — 102 с.
8. McKesson D. OpenGL Insights / D. McKesson. — CRC Press, 2012. — 712с.
9. Бабич А., Кирилюк Т. Алгоритми та структури даних у графіці. — Харків : ХНУРЕ, 2021. — 168с.
10. Node.js Documentation [Електронний ресурс]. — Офіційна документація. https://nodejs.org/en/docs
11. React Documentation – офіційне керівництво по React 18 [Електронний ресурс]. https://react.dev/learn
12. GraphQL Documentation – офіційний сайт [Електронний ресурс]. https://graphql.org/learn/

ДОДАТКИ

Додаток А

Меню вибору рівня

```
namespace PlantQuizGame
{
    public class MenuScreen
    {
        private List<Button> levelButtons = new List<Button>();
        private TextRenderer textRenderer;
        public event EventHandler<int> LevelSelected;

        public MenuScreen(TextRenderer textRenderer)
        {
            this.textRenderer = textRenderer;

            for (int i = 0; i < 5; i++)
            {
                int level = i + 1;
                Button btn = new Button
                {
                    Text = $"Level {level}",
                    X = 300,
                    Y = 150 + i * 80,
                    Width = 200,
                    Height = 60
                };

                btn.Click += (sender, e) =>
LevelSelected?.Invoke(this, level);
                levelButtons.Add(btn);
            }
        }
        public void Render(int width, int height)
        {
            DrawMenuBackground(width, height);

            textRenderer.DrawText("Quiz: Houseplants", width / 2 -
150, 80, Color.DarkGreen);

            foreach (var button in levelButtons)
            {
                DrawAdvancedButton(button, button.IsHovered);
            }
        }

        private void DrawMenuBackground(int width, int height)
        {
            GL.Begin(PrimitiveType.Quads);
            GL.Color4(0.6f, 0.8f, 1.0f, 1.0f);
            GL.Vertex2(0, 0);
            GL.Vertex2(width, 0);
            GL.Color4(0.8f, 0.9f, 1.0f, 1.0f);
            GL.Vertex2(width, height / 2);
            GL.Vertex2(0, height / 2);
```

```
            GL.Color4(0.8f, 0.9f, 1.0f, 1.0f);
            GL.Vertex2(0, height / 2);
            GL.Vertex2(width, height / 2);
            GL.Color4(0.2f, 0.8f, 0.2f, 1.0f);
            GL.Vertex2(width, height);
            GL.Vertex2(0, height);
            GL.End();
            DrawDecorativePlants(width, height);
        }
        private void DrawDecorativePlants(int width, int height)
        {
            DrawSimplePlant(100, height - 150, 0.2f, 0.8f, 0.3f);
            DrawSimplePlant(width - 100, height - 180, 0.3f, 0.7f,
0.2f);
            DrawSimplePlant(50, height - 220, 0.25f, 0.75f, 0.25f);
            DrawSimplePlant(width - 50, height - 200, 0.3f, 0.8f,
0.3f);
        }

        private void DrawSimplePlant(float x, float y, float r, float
g, float b)
        {
            GL.Begin(PrimitiveType.Quads);
            GL.Color4(0.3f, 0.2f, 0.0f, 1.0f);
            GL.Vertex2(x - 5, y + 80);
            GL.Vertex2(x + 5, y + 80);
            GL.Vertex2(x + 5, y);
            GL.Vertex2(x - 5, y);
            GL.End();
            DrawLeaf(x - 15, y + 60, 25, 12, r, g, b, -30);
            DrawLeaf(x + 15, y + 45, 30, 15, r, g, b, 30);
            DrawLeaf(x - 20, y + 30, 35, 18, r, g, b, -45);
            DrawLeaf(x + 18, y + 15, 32, 16, r, g, b, 45);
            DrawFlower(x, y, r, g, b);
        }

        private void DrawLeaf(float x, float y, float width, float
height, float r, float g, float b, float angle)
        {
            GL.PushMatrix();
            GL.Translate(x, y, 0);
            GL.Rotate(angle, 0, 0, 1);
            GL.Begin(PrimitiveType.TriangleFan);
            GL.Color4(r, g, b, 1.0f);
            GL.Vertex2(0, 0);
            int segments = 20;
            for (int i = 0; i <= segments; i++)
            {
                float theta = (float)(2.0 * Math.PI * i / segments);
                float dx = width * (float)Math.Cos(theta);
                float dy = height * (float)Math.Sin(theta);
                GL.Vertex2(dx, dy);
            }
```

```
GL.End();
            GL.PopMatrix();
        }

        private void DrawFlower(float x, float y, float r, float g,
float b)
        {
            GL.Begin(PrimitiveType.TriangleFan);
            GL.Color4(1.0f, 1.0f, 0.0f, 1.0f);
            GL.Vertex2(x, y);
            int segments = 20;
            float radius = 8;
            for (int i = 0; i <= segments; i++)
            {
                float theta = (float)(2.0 * Math.PI * i / segments);
                float dx = radius * (float)Math.Cos(theta);
                float dy = radius * (float)Math.Sin(theta);
                GL.Vertex2(x + dx, y + dy);
            }
            GL.End();
            for (int i = 0; i < 6; i++)
            {
                float angle = (float)(i * Math.PI / 3.0);
                float pX = x + 12 * (float)Math.Cos(angle);
                float pY = y + 12 * (float)Math.Sin(angle);
                GL.Begin(PrimitiveType.TriangleFan);
                GL.Color4(1.0f, 0.4f, 0.7f, 1.0f);
                GL.Vertex2(pX, pY);

                for (int j = 0; j <= 8; j++)
                {
                    float theta = (float)(2.0 * Math.PI * j / 8);
                    float dx = 6 * (float)Math.Cos(theta);
                    float dy = 6 * (float)Math.Sin(theta);
                    GL.Vertex2(pX + dx, pY + dy);
                }
                GL.End();
            }
        }

        private void DrawAdvancedButton(Button button, bool
isHighlighted)
        {
            GL.Begin(PrimitiveType.Quads);

            if (isHighlighted)
            {
                GL.Color4(0.2f, 0.6f, 0.2f, 1.0f);
            }
            else
            {
                GL.Color4(0.1f, 0.5f, 0.1f, 1.0f);
            }

            GL.Vertex2(button.X, button.Y);
```

```
            GL.Vertex2(button.X + button.Width, button.Y);
            GL.Vertex2(button.X + button.Width, button.Y +
button.Height);
            GL.Vertex2(button.X, button.Y + button.Height);
            GL.End();

            GL.Begin(PrimitiveType.LineLoop);
            GL.Color4(0.0f, 0.3f, 0.0f, 1.0f);
            GL.Vertex2(button.X, button.Y);
            GL.Vertex2(button.X + button.Width, button.Y);


            GL.Vertex2(button.X + button.Width, button.Y +
button.Height);
            GL.Vertex2(button.X, button.Y + button.Height);
            GL.End();

            GL.Begin(PrimitiveType.Quads);
            GL.Color4(1.0f, 1.0f, 1.0f, 0.3f);
            GL.Vertex2(button.X, button.Y);
            GL.Vertex2(button.X + button.Width, button.Y);
            GL.Color4(1.0f, 1.0f, 1.0f, 0.0f);
            GL.Vertex2(button.X + button.Width, button.Y +
button.Height / 2);
            GL.Vertex2(button.X, button.Y + button.Height / 2);
            GL.End();

            textRenderer.DrawText(button.Text, button.X + 20, button.Y
+ 20, Color.White);
        }

        public void HandleMouseDown(int x, int y)
        {
            foreach (var button in levelButtons)
            {
                if (x >= button.X && x <= button.X + button.Width &&
                    y >= button.Y && y <= button.Y + button.Height)
                {
                    button.OnClick();
                    break;
                }
            }
        }

        public void CheckHover(int x, int y)
        {
            foreach (var button in levelButtons)
            {
                button.IsHovered = (x >= button.X && x <= button.X +
button.Width &&
                                    y >= button.Y && y <= button.Y +
button.Height);
            }
        }
}
```

Додаток Б

Ігрове поле

```
namespace PlantQuizGame
{
    public class GameScreen
    {
        public int CurrentLevel { get; private set; } = 1;
        private Plant currentPlant;
        private bool isDragging = false;
        private float dragOffsetX, dragOffsetY;
        private List<PlacementSpot> placementSpots = new
List<PlacementSpot>();
        private TextRenderer textRenderer;

        private Button menuButton;
        private Button nextLevelButton;

        public event EventHandler ReturnToMenu;
        public event EventHandler NextLevel;

        private string plantFeedbackText = "";
        private Color plantFeedbackColor = Color.Black;

        public GameScreen(TextRenderer textRenderer)
        {
            this.textRenderer = textRenderer;

            placementSpots.Add(new PlacementSpot(150, 300, 50));
            placementSpots.Add(new PlacementSpot(400, 300, 80));
            placementSpots.Add(new PlacementSpot(650, 300, 100));

            menuButton = new Button
            {
                Text = "Menu",
                X = 50,
                Y = 50,
                Width = 120,
                Height = 40
            };
            menuButton.Click += (sender, e) =>
ReturnToMenu?.Invoke(this, EventArgs.Empty);

            nextLevelButton = new Button
            {
                Text = "Next",
                X = 630,
                Y = 50,
                Width = 120,
                Height = 40
            };
            nextLevelButton.Click += (sender, e) =>
NextLevel?.Invoke(this, EventArgs.Empty);
        }
```

```
        public void SetLevel(int level)
        {
            CurrentLevel = level;

            switch (level)
            {
                case 1:
                    currentPlant = new Plant("Cactus", 100, 400, 400);
                    break;
                case 2:
                    currentPlant = new Plant("Orchid", 80, 400, 400);
                    break;
                case 3:
                    currentPlant = new Plant("Violet", 50, 400, 400);
                    break;
                case 4:
                    currentPlant = new Plant("Avocado", 90, 400, 400);
                    break;
                case 5:
                    currentPlant = new Plant("Monstera", 70, 400,
400);
                    break;
            }

            plantFeedbackColor = Color.DarkBlue;
        }

        public void Render(int width, int height)
        {
            DrawRoom();
            DrawWindow();
            DrawLightRays();

            foreach (var spot in placementSpots)
            {
                DrawPlacementSpot(spot);
            }

            DrawPlant();
            DrawButton(menuButton);
            DrawButton(nextLevelButton);

            int lightQuality = CalculateLightQuality();
            string percentText = $"{lightQuality}%";
            float textWidth = percentText.Length * 20;
            textRenderer.DrawText(percentText, 400 - textWidth / 2,
100, GetQualityColor(lightQuality));
        }

        private Color GetQualityColor(int quality)
        {
            if (quality < 30) return Color.Red;
            if (quality < 70) return Color.Orange;
            return Color.Green;
        }
```

```
        private int CalculateLightQuality()
        {
            PlacementSpot nearestSpot = null;
            float minDistance = float.MaxValue;

            foreach (var spot in placementSpots)
            {
                float distance = (float)Math.Sqrt(
                    Math.Pow(currentPlant.X - spot.X, 2) +
                    Math.Pow(currentPlant.Y - spot.Y, 2));

                if (distance < minDistance)
                {
                    minDistance = distance;
                    nearestSpot = spot;
                }
            }

            if (nearestSpot == null || minDistance > 50)
            {
                plantFeedbackText = "Place the plant closer to a
lighting zone!";
                plantFeedbackColor = Color.Red;
                return 0;
            }

            int diff = Math.Abs(currentPlant.OptimalLight -
nearestSpot.LightLevel);
            int quality = Math.Max(0, 100 - diff * 2);

            return quality;
        }

        private void DrawRoom()
        {
            GL.Color4(0.9f, 0.9f, 0.8f, 1.0f);

            GL.Begin(PrimitiveType.Quads);
            GL.Vertex2(0, 0);
            GL.Vertex2(800, 0);
            GL.Vertex2(800, 600);
            GL.Vertex2(0, 600);
            GL.End();

            GL.Color4(0.6f, 0.4f, 0.2f, 1.0f);
            GL.Begin(PrimitiveType.Quads);
            GL.Vertex2(0, 400);
            GL.Vertex2(800, 400);
            GL.Vertex2(800, 600);
            GL.Vertex2(0, 600);
            GL.End();
        }
```

```
private void DrawWindow()
{
    GL.Color4(0.6f, 0.6f, 0.6f, 1.0f);
    GL.Begin(PrimitiveType.Quads);
    GL.Vertex2(650, 100);
    GL.Vertex2(750, 100);
    GL.Vertex2(750, 250);
    GL.Vertex2(650, 250);
    GL.End();

    GL.Color4(0.7f, 0.9f, 1.0f, 0.8f);
    GL.Begin(PrimitiveType.Quads);
    GL.Vertex2(660, 110);
    GL.Vertex2(740, 110);
    GL.Vertex2(740, 240);
    GL.Vertex2(660, 240);
    GL.End();
}

private void DrawLightRays()
{
    GL.Enable(EnableCap.Blend);
    GL.Color4(1.0f, 1.0f, 0.8f, 0.3f);

    GL.Begin(PrimitiveType.Triangles);
    GL.Vertex2(700, 175);
    GL.Vertex2(650, 300);
    GL.Vertex2(750, 300);
    GL.End();

    GL.Begin(PrimitiveType.Triangles);
    GL.Color4(1.0f, 1.0f, 0.8f, 0.2f);
    GL.Vertex2(700, 175);
    GL.Vertex2(350, 300);
    GL.Vertex2(450, 300);
    GL.End();

    GL.Begin(PrimitiveType.Triangles);
    GL.Color4(1.0f, 1.0f, 0.8f, 0.1f);
    GL.Vertex2(700, 175);
    GL.Vertex2(100, 300);
    GL.Vertex2(200, 300);
    GL.End();

    GL.Disable(EnableCap.Blend);
}
```

```
        private void DrawPlacementSpot(PlacementSpot spot)
        {
            GL.Color4(0.7f, 0.7f, 0.7f, 0.5f);

            GL.Begin(PrimitiveType.Quads);
            GL.Vertex2(spot.X - 30, spot.Y - 30);
            GL.Vertex2(spot.X + 30, spot.Y - 30);
            GL.Vertex2(spot.X + 30, spot.Y + 30);
            GL.Vertex2(spot.X - 30, spot.Y + 30);
            GL.End();

            string lightText = "";
            if (spot.LightLevel <= 50) lightText = "Low Light";
            else if (spot.LightLevel <= 80) lightText = "Medium
Light";
            else lightText = "Bright Light";

            float textWidth = lightText.Length * 8;
        }

        private void DrawPlant()
        {
            switch (currentPlant.Name)
            {
                case "Cactus":
                    DrawCactus(currentPlant.X, currentPlant.Y);
                    break;
                case "Orchid":
                    DrawOrchid(currentPlant.X, currentPlant.Y);
                    break;
                case "Violet":
                    DrawViolet(currentPlant.X, currentPlant.Y);
                    break;
                case "Avocado":
                    DrawAvocado(currentPlant.X, currentPlant.Y);
                    break;
                case "Monstera":
                    DrawMonstera(currentPlant.X, currentPlant.Y);
                    break;
            }
        }

        private void DrawCactus(float x, float y)
        {
            DrawPot(x, y);

            GL.Color4(0.0f, 0.6f, 0.0f, 1.0f);
            GL.Begin(PrimitiveType.Quads);
            GL.Vertex2(x - 15, y - 80);
            GL.Vertex2(x + 15, y - 80);
            GL.Vertex2(x + 20, y - 10);
            GL.Vertex2(x - 20, y - 10);
            GL.End();

            GL.Color4(0.7f, 0.7f, 0.7f, 1.0f);
```

```
    GL.Begin(PrimitiveType.Lines);
    for (int i = 0; i < 15; i++)
    {
        float y1 = y - 75 + i * 5;
        GL.Vertex2(x - 15, y1);
        GL.Vertex2(x - 25, y1);

        GL.Vertex2(x + 15, y1);
        GL.Vertex2(x + 25, y1);
    }
    GL.End();
}

private void DrawOrchid(float x, float y)
{
    DrawPot(x, y);

    GL.Color4(0.0f, 0.5f, 0.0f, 1.0f);
    GL.Begin(PrimitiveType.Lines);
    GL.Vertex2(x, y - 10);
    GL.Vertex2(x, y - 80);
    GL.End();

    GL.Color4(0.9f, 0.5f, 0.9f, 1.0f);
    for (int i = 0; i < 3; i++)
    {
        float cy = y - 40 - i * 20;
        DrawFlower(x, cy, 10);
    }
}

private void DrawViolet(float x, float y)
{
    DrawPot(x, y);

    GL.Color4(0.0f, 0.4f, 0.0f, 1.0f);
    for (int i = 0; i < 6; i++)
    {
        float angle = (float)(i * Math.PI / 3);
        float lx = x + 20 * (float)Math.Cos(angle);
        float ly = y - 20 + 10 * (float)Math.Sin(angle);

        GL.Begin(PrimitiveType.Triangles);
        GL.Vertex2(x, y - 10);
        GL.Vertex2(lx - 10, ly - 10);
        GL.Vertex2(lx + 10, ly - 10);
        GL.End();
    }
```

```
    GL.Color4(0.6f, 0.2f, 0.8f, 1.0f);
    for (int i = 0; i < 4; i++)
    {
        float angle = (float)(i * Math.PI / 2 + Math.PI / 4);
        float fx = x + 15 * (float)Math.Cos(angle);
        float fy = y - 30 + 5 * (float)Math.Sin(angle);

        DrawFlower(fx, fy, 5);
    }
}

private void DrawAvocado(float x, float y)
{
    DrawPot(x, y);

    GL.Color4(0.5f, 0.3f, 0.0f, 1.0f);
    GL.Begin(PrimitiveType.Quads);
    GL.Vertex2(x - 5, y - 80);
    GL.Vertex2(x + 5, y - 80);
    GL.Vertex2(x + 5, y - 10);
    GL.Vertex2(x - 5, y - 10);
    GL.End();

    GL.Color4(0.0f, 0.6f, 0.0f, 1.0f);
    for (int i = 0; i < 5; i++)
    {
        float ly = y - 30 - i * 10;
        float size = 15 - i * 2;

        GL.Begin(PrimitiveType.Triangles);
        GL.Vertex2(x, ly);
        GL.Vertex2(x - size, ly - size);
        GL.Vertex2(x - size, ly + size);

        GL.Vertex2(x, ly);
        GL.Vertex2(x + size, ly - size);
        GL.Vertex2(x + size, ly + size);
        GL.End();
    }
}

private void DrawMonstera(float x, float y)
{
    DrawPot(x, y);

    GL.Color4(0.2f, 0.5f, 0.2f, 1.0f);
    GL.Begin(PrimitiveType.Lines);
    GL.Vertex2(x, y - 10);
    GL.Vertex2(x, y - 60);
    GL.End();
```

```
    GL.Color4(0.0f, 0.5f, 0.0f, 1.0f);
    for (int i = 0; i < 3; i++)
    {
        float angle = (float)(i * Math.PI * 2 / 3);
        float lx = x + 10 * (float)Math.Cos(angle);
        float ly = y - 50;

        DrawMonsteraLeaf(lx, ly, angle);
    }
}

private void DrawMonsteraLeaf(float x, float y, float angle)
{
    GL.PushMatrix();
    GL.Translate(x, y, 0);
    GL.Rotate(angle * 180 / Math.PI, 0, 0, 1);

    GL.Begin(PrimitiveType.TriangleFan);
    GL.Vertex2(0, 0);

    for (int i = 0; i <= 10; i++)
    {
        float a = (float)(i * Math.PI / 10);
        float r = 30;
        if (i % 2 == 1) r = 20;

        float px = r * (float)Math.Cos(a);
        float py = r * (float)Math.Sin(a);

        GL.Vertex2(px, py);
    }
    GL.End();

    GL.PopMatrix();
}

private void DrawPot(float x, float y)
{
    GL.Color4(0.8f, 0.4f, 0.0f, 1.0f);
    GL.Begin(PrimitiveType.Quads);
    GL.Vertex2(x - 20, y);
    GL.Vertex2(x + 20, y);
    GL.Vertex2(x + 15, y - 10);
    GL.Vertex2(x - 15, y - 10);
    GL.End();

    GL.Color4(0.6f, 0.3f, 0.0f, 1.0f);
    GL.Begin(PrimitiveType.Quads);
    GL.Vertex2(x - 15, y - 10);
    GL.Vertex2(x + 15, y - 10);
    GL.Vertex2(x + 15, y + 10);
    GL.Vertex2(x - 15, y + 10);
    GL.End();
}
```

```
        private void DrawFlower(float x, float y, float size)
        {
            GL.Begin(PrimitiveType.TriangleFan);
            GL.Vertex2(x, y);

            for (int i = 0; i <= 20; i++)
            {
                float angle = (float)(i * Math.PI * 2 / 20);
                float px = x + size * (float)Math.Cos(angle);
                float py = y + size * (float)Math.Sin(angle);

                GL.Vertex2(px, py);
            }
            GL.End();

            GL.Color4(1.0f, 1.0f, 0.0f, 1.0f);
            GL.Begin(PrimitiveType.TriangleFan);
            GL.Vertex2(x, y);

            for (int i = 0; i <= 20; i++)
            {
                float angle = (float)(i * Math.PI * 2 / 20);
                float px = x + (size / 3) * (float)Math.Cos(angle);
                float py = y + (size / 3) * (float)Math.Sin(angle);

                GL.Vertex2(px, py);
            }
            GL.End();
        }

        private void DrawButton(Button button)
        {
            if (button.IsHovered)
                GL.Color4(0.7f, 0.7f, 0.7f, 1.0f);
            else
                GL.Color4(0.8f, 0.8f, 0.8f, 1.0f);

            GL.Begin(PrimitiveType.Quads);
            GL.Vertex2(button.X, button.Y);
            GL.Vertex2(button.X + button.Width, button.Y);
            GL.Vertex2(button.X + button.Width, button.Y +
button.Height);
            GL.Vertex2(button.X, button.Y + button.Height);
            GL.End();

            GL.Color4(0.0f, 0.0f, 0.0f, 1.0f);
            GL.Begin(PrimitiveType.LineLoop);
            GL.Vertex2(button.X, button.Y);
            GL.Vertex2(button.X + button.Width, button.Y);
            GL.Vertex2(button.X + button.Width, button.Y +
button.Height);
            GL.Vertex2(button.X, button.Y + button.Height);
            GL.End();
```

Продовження додатку Б

```
            textRenderer.DrawText(button.Text, button.X + 10, button.Y
+ 10, Color.Black);

        }

        public void HandleMouseDown(int x, int y)
        {
            if (x >= menuButton.X && x <= menuButton.X +
menuButton.Width &&
                y >= menuButton.Y && y <= menuButton.Y +
menuButton.Height)
            {
                menuButton.OnClick();
                return;
            }

            if (x >= nextLevelButton.X && x <= nextLevelButton.X +
nextLevelButton.Width &&
                y >= nextLevelButton.Y && y <= nextLevelButton.Y +
nextLevelButton.Height)
            {
                nextLevelButton.OnClick();
                return;
            }

            if (Math.Abs(x - currentPlant.X) <= 40 && Math.Abs(y -
currentPlant.Y) <= 40)
            {
                isDragging = true;
                dragOffsetX = x - currentPlant.X;
                dragOffsetY = y - currentPlant.Y;
            }
        }

        public void HandleMouseMove(int x, int y)
        {
            if (isDragging)
            {
                currentPlant.X = x - dragOffsetX;
                currentPlant.Y = y - dragOffsetY;
            }
        }

        public void HandleMouseUp(int x, int y)
        {
            isDragging = false;
        }
    }
}
```

Додаток В

Header сайту

```
import { Box, Link as JoyLink } from '@mui/joy'
import { Link } from 'react-router-dom'
import ThemeSwitcher from '../ThemeSwitcher'
import { CartIcon } from '../'

const sxLink = {
  color: 'text.primary',
  fontSize: 18,
  fontWeight: '400'
}

const Header = () => (
  <Box
    sx={{
      position: 'fixed',
      top: 0,
      width: '100%',
      zIndex: 100,
      display: 'flex',
      justifyContent: 'center',
      alignItems: 'center',
      p: 2,
      bgcolor: 'background.surface',
      color: 'text.primary',
    }}
  >
    <Box sx={{ display: 'flex', gap: 6 }}>
      <JoyLink sx={sxLink} component={Link} to='/'>Головна</JoyLink>
      <JoyLink sx={sxLink} component={Link} to='/about'>Про
нас</JoyLink>
      <JoyLink sx={sxLink} component={Link}
to='/shop'>Магазин</JoyLink>
      <JoyLink sx={sxLink} component={Link} to='/blog'>Блог</JoyLink>
      <ThemeSwitcher />
      <CartIcon/>
    </Box>
  </Box>
)

export default Header
```

Додаток Г

Footer сайту

```
import { Box, Typography } from '@mui/joy'

const Footer = () => (
  <Box
    sx={{
      p: 2,
      textAlign: 'center',
      bgcolor: 'background.surface',
      color: 'text.primary',
      zIndex: 1
    }}
  >
    <Typography level='body-sm'>
      © 2025 Зелена Нота — квіти, думки та тепло душі
    </Typography>
  </Box>
)

export default Footer
```

Додаток Д

Роутер сайту

```
import { Routes, Route } from 'react-router-dom'
import Layout from './Layout'
import { Home, About, Blog, Shop, NotFound, CreatePost } from
'../pages'
import { Box } from '@mui/joy'

const AppRouter = () => {
    return(

      <Routes>
        <Route path='/' element={<Layout />}>
          <Route index element={<Home />} />
          <Route path='about' element={<About />} />
          <Route path='shop' element={<Shop />} />
          <Route path='blog' element={<Blog />} />
          <Route path='createPost' element={<CreatePost />} />
          <Route path='*' element={<NotFound />} />
        </Route>
    </Routes>

)
}

export default AppRouter
```

Додаток Е

Кастомний хук для роботи з корзиною

```
import { createContext, useContext, useState } from 'react';

const CartContext = createContext();

export const CartProvider = ({ children }) => {
  const [cartItems, setCartItems] = useState([]);

  const addToCart = (product) => {
    setCartItems((prevItems) => {
      const existing = prevItems.find(item => item.id === product.id);
      if (existing) {
        return prevItems.map(item =>
          item.id === product.id
            ? { ...item, quantity: item.quantity + 1 }
            : item
        );
      } else {
        return [...prevItems, { ...product, quantity: 1 }];
      }
    });
  };

  const increaseQuantity = (productId) => {
    setCartItems(prevItems =>
      prevItems.map(item =>
        item.id === productId
          ? { ...item, quantity: item.quantity + 1 }
          : item
      )
    );
  };

  const decreaseQuantity = (productId) => {
    setCartItems(prevItems =>
      prevItems
        .map(item =>
          item.id === productId
            ? { ...item, quantity: item.quantity - 1 }
            : item
        )
        .filter(item => item.quantity > 0)
    );
  };

  const removeFromCart = (productId) => {
    setCartItems(prevItems => prevItems.filter(item => item.id !==
productId));
  };

  const clearCart = () => {
    setCartItems([]);
  };
```

Продовження додатку Е

```
  const getTotalPrice = (items = cartItems) => {
    return items.reduce((sum, item) => sum + item.price *
item.quantity, 0);
  };

  const getOrderDetails = (items = cartItems) => {
    return items.map(item =>
      `${item.name}: ${item.quantity} шт | ${item.price *
item.quantity} грн`
    ).join('\n');
  };

  return (
    <CartContext.Provider
      value={{
        cartItems,
        addToCart,
        increaseQuantity,
        decreaseQuantity,
        removeFromCart,
        clearCart,
        getTotalPrice,
        getOrderDetails
      }}
    >
      {children}
    </CartContext.Provider>
  );
};

export const useCart = () => useContext(CartContext);
```

Додаток Ж

Стартова функція запуску сервера

```
import express from 'express';
import { ApolloServer } from 'apollo-server-express';
import { readFileSync } from 'fs';
import path from 'path';
import { fileURLToPath } from 'url';
import dotenv from 'dotenv';
import resolvers from './resolvers/index.js';
import db from './db/db.js';

dotenv.config();

const __filename = fileURLToPath(import.meta.url);
const __dirname = path.dirname(__filename);

const typeDefs = readFileSync(path.join(__dirname, 'schema',
'schema.graphql'), 'utf8');

const app = express();
const server = new ApolloServer({
  typeDefs,
  resolvers,
  context: () => ({ db })
});

const PORT = process.env.PORT || 4000;

async function startServer() {
  await server.start();
  server.applyMiddleware({ app });

  app.listen(PORT, () => {
    console.log(`Server ready at
http://localhost:${PORT}${server.graphqlPath}`);
  });
}

startServer();
```

Додаток И

Файли запитів до бази даних

```
import db from '../../db/db.js';

export const orders = () => {
  console.log('hello')
  return db.prepare('SELECT * FROM orders').all();
};
```

```
import db from '../../db/db.js';

export const posts = () => {
  return db.prepare(`
    SELECT p.*, u.id as userId, u.username as userUsername
    FROM posts p
    LEFT JOIN users u ON p.author_id = u.id
  `).all();
};
```

```
import db from '../../db/db.js';

export const products = () => {
  const rows = db.prepare(`
    SELECT pr.*, pc.id as categoryId, pc.name as categoryName
    FROM products pr
    LEFT JOIN product_categories pc ON pr.category_id = pc.id
  `).all();

  const mapped = rows.map(row => ({
    id: row.id,
    name: row.name,
    description: row.description,
    image_url: row.image_url,
    price: row.price,
    quantity: row.quantity,
    product_category: {
      id: row.categoryId,
      name: row.categoryName
    }
  }));

  return mapped;
};
```

```
import db from '../../db/db.js';

export const searchPost = (parent, args, context, info) => {
  const { title } = args;
```

```
const stmt = db.prepare(`
    SELECT p.*, u.id as userId, u.username as userUsername
    FROM posts p
    LEFT JOIN users u ON p.author_id = u.id
    WHERE p.title LIKE ?

 `);

  return stmt.all(`%${title}%`);
};
```

```
import db from '../../db/db.js';

export const createPost = (parent, args, context, info) => {
  const { title, content, image_url } = args;
  console.log(args)
  const stmt = db.prepare(`
    INSERT INTO posts (title, content, image_url, author_id)
    VALUES (?, ?, ?, ?)
  `);

  const result = stmt.run(title, content, image_url || null, 1);
  const id = result.lastInsertRowid;

  return {
    id,
    title,
    content,
    image_url,
    createdAt: new Date().toISOString(),
    author: {
      id: 1,
      username: 'admin'
       }
  };
};
```

```
import db from '../../db/db.js';

export const createOrder = (parent, args, context, info) => {
  const { customerName, customerPhone, customerAddress, orderDetails,
totalPrice } = args;
```

```
  const stmt = db.prepare(`
    INSERT INTO orders (customer_name, customer_phone,
customer_address, order_text, total_price)
    VALUES (?, ?, ?, ?, ?)
  `);



  const result = stmt.run(customerName, customerPhone,
customerAddress, orderDetails, totalPrice);

  const id = result.lastInsertRowid;

  return {
    id,
    customerName,
    customerPhone,
    customerAddress,
    orderDetails,
    totalPrice,
    createdAt: new Date().toISOString()
  };
};
```

Додаток К

Запит створення бази даних sqlite

```
CREATE TABLE users (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  username TEXT NOT NULL UNIQUE,
  password TEXT NOT NULL,
  email TEXT,
  created_at TEXT DEFAULT CURRENT_TIMESTAMP,
  is_active BOOLEAN DEFAULT 1
);


CREATE TABLE posts (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  title TEXT NOT NULL,
  content TEXT NOT NULL,
  image_url TEXT,
  created_at TEXT DEFAULT CURRENT_TIMESTAMP,
  author_id INTEGER NOT NULL,
  FOREIGN KEY (author_id) REFERENCES users(id)
);


CREATE TABLE products (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  name TEXT NOT NULL,
  description TEXT,
  price REAL NOT NULL,
  image_url TEXT,
  quantity INTEGER DEFAULT 0,
  category_id INTEGER,
  created_at TEXT DEFAULT CURRENT_TIMESTAMP,
  is_active BOOLEAN DEFAULT 1,
  FOREIGN KEY (category_id) REFERENCES product_categories(id)
);


CREATE TABLE product_categories (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  name TEXT NOT NULL,
  description TEXT,
  created_at TEXT DEFAULT CURRENT_TIMESTAMP
);


CREATE TABLE orders (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  total_price REAL NOT NULL,
  order_text TEXT,
  customer_name TEXT NOT NULL,
  customer_phone TEXT NOT NULL,
  customer_address TEXT NOT NULL,
  created_at TEXT DEFAULT CURRENT_TIMESTAMP,
  status TEXT DEFAULT 'pending'
);
```

```
CREATE TABLE order_items (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  order_id INTEGER NOT NULL,
  product_id INTEGER NOT NULL,
  quantity INTEGER NOT NULL,
  unit_price REAL NOT NULL,
  FOREIGN KEY (order_id) REFERENCES orders(id),
  FOREIGN KEY (product_id) REFERENCES products(id)
);


CREATE TABLE newsletter_subscribers (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  email TEXT NOT NULL UNIQUE,
  subscribed_at TEXT DEFAULT CURRENT_TIMESTAMP
);
```

Додаток Л

Схема graphql

```
type Query {
  posts: [Post]
  products: [Product]
  orders: [Order]
  searchPost(title: String!): [Post]
}

type Mutation {
  createPost(title: String!, content: String!, image_url: String):
Post
  createOrder(customerName: String!, customerPhone: String!,
customerAddress: String!, orderDetails: String!, totalPrice: Float!):
Order
}

type Post {
  id: ID!
  title: String!
  content: String!
  image_url: String
  createdAt: String!
  author: User
}

type Product {
  id: ID!
  name: String!
  description: String!
  image_url: String
  price: Float!
  quantity: Int!
  product_category: Category!
}

type Order {
  id: ID!
  customerName: String!
  customerPhone: String!
  customerAddress: String!
  orderDetails: String!
  totalPrice: Float!
  createdAt: String!
}

type Category {
  id: ID!
  name: String!
}

type User {
  id: ID!
  username: String!
}
```