МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ ФАХОВИЙ БІЗНЕС-КОЛЕДЖ
Циклова комісія комп’ютерної інженерії та інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА

на тему

ІНТЕРАКТИВНИЙ ВЕБСАЙТ-ТРЕНАЖЕР ДЛЯ ВІЗУАЛІЗАЦІЇ АЛГОРИТМІВ ТА СТРУКТУР ДАНИХ

Виконав:
студент групи 2П-22
спеціальності
121 Інженерія програмного забезпечення
Вадим БРИКУН
Керівник:
Наталя ФАЛЬЧЕНКО

Черкаси 2026

ЗМІСТ

ВСТУП 3
РОЗДІЛ 1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ 6
1.1 Аналіз ролі алгоритмів у програмуванні 6
1.2 Використання візуалізації у процесі вивчення алгоритмів та структур даних 7
1.3 Огляд існуючих інтернет-ресурсів, що характеризують предметну область 8
1.4 Аналіз переваг та недоліків оглянутих вебресурсів 12
1.5 Постановка задачі на розробку інтерактивного вебсайту-тренажера 13
РОЗДІЛ 2 ПРОЄКТУВАННЯ ВЕБСАЙТУ-ТРЕНАЖЕРА 16
2.1 Архітектура та структура вебсайту-тренажера 16
2.2 Вибір засобів і технологій розробки 17
2.3 Функціональні та нефункціональні вимоги 18
2.4 Проєктування структури інтерфейсу 21
2.5 Проєктування модуля «Алгоритми» 23
2.6 Проєктування модуля «Структури даних» 24
РОЗДІЛ 3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ВЕБСАЙТУ-ТРЕНАЖЕРА 27
3.1 Розробка функціональної структури проєкту 27
3.2 Реалізація навігації та загального макета 28
3.3 Реалізація сторінки «Алгоритми» 28
3.4 Реалізація псевдокоду та блок-схем 33
3.5 Реалізація сторінки «Структури даних» 34
3.6 Тестування вебсайту-тренажера 39
3.7 Розміщення вебсайту на Netlify 44
3.8 Загальний огляд функціоналу готового продукту 46
ВИСНОВКИ 48
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 50
ДОДАТКИ 52

ВСТУП

Актуальність теми. Алгоритми та структури даних залишаються основою програмування — без них не обходиться жодна серйозна задача, від простого сортування списку до роботи складних баз даних чи систем штучного інтелекту [1]. Студенти, які вивчають інформаційні технології, рано чи пізно стикаються з потребою не просто вивчити окремі алгоритми напам'ять, а зрозуміти логіку їх роботи — чому один підхід кращий за інший у конкретній ситуації, як змінюється стан даних на кожному кроці виконання [2]. На практиці ця тема часто дається важко. Більшість підручників і курсів подають алгоритми у вигляді тексту та статичного коду, через що студенту доводиться самостійно уявляти, як відбуваються порівняння елементів, переходи між вузлами дерева чи переміщення вказівників у списку [4]. Для багатьох це створює суттєвий бар'єр: матеріал виглядає абстрактним, а зв'язок між формулою чи кодом і реальним результатом не завжди очевидний.

Одним зі способів подолати цей розрив є візуалізація. Коли студент бачить, як саме змінюються елементи масиву під час сортування, або як вузли дерева перебудовуються після вставки нового значення, абстрактна логіка перетворюється на конкретний, відстежуваний процес [22]. Покрокове відображення дозволяє зупинитися на будь-якому моменті, повернутися назад і розібратися, чому саме відбулася та чи інша зміна — чого неможливо досягти, читаючи статичний текст або переглядаючи відео без можливості взаємодії [7]. Окремо варто відзначити роль сучасних вебтехнологій у цьому процесі. Браузер сьогодні — це повноцінне середовище для інтерактивних застосунків, які не потребують встановлення додаткового програмного забезпечення і працюють на будь-якому пристрої [12]. Це відкриває можливість створювати навчальні інструменти, доступ до яких можна отримати з телефону, ноутбука чи комп'ютера в комп'ютерному класі — без обмежень, пов'язаних з операційною системою чи потужністю обладнання.

Аналіз існуючих рішень показує, що подібні навчальні ресурси вже існують і користуються популярністю серед студентів та викладачів [20], [21], [22]. Однак кожен з них має свої особливості: одні зосереджені виключно на алгоритмах сортування, інші вимагають базових знань програмування для розуміння, треті перевантажені інформацією і складні для новачка. Це створює простір для розробки інструмента, який поєднував би простоту інтерфейсу, охоплення як алгоритмів, так і структур даних, та зручність для користувачів з різним рівнем підготовки.

Саме тому темою кваліфікаційної роботи обрано розробку інтерактивного вебсайту-тренажера для візуалізації алгоритмів та структур даних — інструмента, який допоможе студентам наочно побачити те, що раніше доводилося лише уявляти.

Метою кваліфікаційної роботи є розробка інтерактивного вебсайту-тренажера для візуалізації алгоритмів та структур даних, який забезпечує наочне, зручне та доступне вивчення основних алгоритмів і способів організації даних у навчальному процесі.

Для досягнення поставленої мети визначено такі завдання**:**

1. Провести огляд існуючих інтернет-ресурсів для візуалізації алгоритмів та структур даних.
2. Проаналізувати особливості використання візуалізації у процесі вивчення алгоритмів.
3. Сформулювати основні вимоги до інтерактивного вебсайту-тренажера.
4. Обґрунтувати вибір технологій і засобів розробки.
5. Спроєктувати логіку роботи вебсайту-тренажера та сценарії взаємодії користувача.
6. Спроєктувати структуру та основні модулі програмного продукту.
7. Провести тестування розробленого вебсайту та проаналізувати результати його роботи.

Об’єкт дослідження – система вивчення, аналізу та візуального подання алгоритмів і структур даних у навчальному середовищі.

Предмет дослідження – методи, що забезпечують наочну візуалізацію алгоритмів та структур даних.

Пактичне значення. Розроблений вебсайт-тренажер може використовуватись як навчальний ресурс при вивченні алгоритмів і структур даних. Покрокова візуалізація дозволяє побачити як саме працює алгоритм, а не просто прочитати про це в теорії. Сайт розрахований на студентів, учнів і викладачів — як для роботи на заняттях, так і для самостійного навчання.

РОЗДІЛ 1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз ролі алгоритмів у програмуванні

Алгоритми використовуються в усіх сферах, пов'язаних з інформаційними технологіями, і є основою програмування. Знання існуючих алгоритмів та вміння застосовувати їх на практиці дає можливість створювати надійні й ефективні програми для розв'язання різноманітних задач. Крім того, підготовка майбутніх програмістів передбачає не лише засвоєння певних знань, а й формування алгоритмічного мислення, тобто здатності аналізувати задачу, визначати послідовність дій для її розв'язання та обирати ефективні способи реалізації [2], [3].

З огляду на це, важливо розглянути саме поняття алгоритму більш детально. Алгоритм є фундаментальною категорією інформатики та програмування, що відображає логіку розв'язання обчислювальної задачі. У контексті розробки програмного забезпечення алгоритм визначає не просто послідовність дій, а формалізовану модель обчислювального процесу, яка згодом транслюється у програмний код [1]. Якість алгоритмічного рішення безпосередньо детермінує якість програмної реалізації: обраний підхід до побудови алгоритму впливає на читабельність, підтримуваність та ефективність коду, тому коректне алгоритмічне мислення є передумовою якісного програмування.

Ключовими властивостями алгоритму, що визначають його придатність до практичного застосування, є детермінованість, скінченність, дискретність, результативність та масовість [1]. Дотримання цих властивостей під час проєктування алгоритму не є формальною вимогою, а є критерієм коректності майбутньої програмної реалізації: недотримання скінченності призводить до нескінченних циклів, недотримання детермінованості — до недетермінованої поведінки програми, а недостатня масовість обмежує універсальність розробленого рішення. Таким чином, аналіз алгоритмічних властивостей на етапі проєктування дозволяє виявити потенційні проблеми реалізації ще до

написання коду, що особливо актуально при розробці складних програмних систем.

Окрему роль у теорії алгоритмів відіграє поняття складності. Один і той самий результат може бути досягнутий різними алгоритмами, однак вони можуть відрізнятися швидкістю роботи, обсягом необхідної пам'яті та загальною ефективністю. Саме тому у процесі навчання важливо не лише засвоювати алгоритми як послідовності дій, а й порівнювати їх між собою, аналізувати переваги та недоліки, оцінювати наскільки виправдане використання в конкретних умовах і розуміти можливості оптимізації [1], [6].

Вивчення алгоритмів нерозривно пов'язане зі структурами даних. Якщо алгоритм визначає, які дії виконуються над інформацією, то структура даних визначає спосіб її організації та зберігання. У навчальній практиці розглядаються як базові структури даних, так і складніші, зокрема масиви, списки, стеки, черги, хеш-таблиці, дерева та словники. Вибір структури даних безпосередньо впливає на спосіб реалізації алгоритму та ефективність його роботи. Саме тому алгоритми і структури даних варто розглядати в тісному взаємозв'язку [5].

1.2 Використання візуалізації у процесі вивчення алгоритмів та структур даних

Вивчення алгоритмів та структур даних часто викликає труднощі через абстрактний характер багатьох процесів. Під час аналізу алгоритму студенту потрібно не лише читати його опис або програмний код, а й уявляти, як змінюються дані на кожному етапі виконання, яким чином здійснюються порівняння, перестановки, переходи між елементами чи обхід структур даних. Саме тому важливу роль у навчальному процесі відіграє візуалізація, яка дозволяє подати роботу алгоритмів та структур даних у більш наочному і зрозумілому вигляді.

Особливо корисною візуалізація є під час вивчення алгоритмів сортування, пошуку, а також роботи з масивами, списками, стеками, чергами,

деревами та графами. Вона дає змогу не лише побачити кінцевий результат, а й простежити весь процес виконання алгоритму поетапно, зрозуміти логіку окремих операцій та відмінності між різними підходами до розв'язання однієї й тієї самої задачі. Поєднання теоретичного матеріалу з графічним відображенням процесів робить навчання більш доступним, підвищує зацікавленість користувача та сприяє кращому засвоєнню матеріалу.

Наукові дослідження показують, що ефективність алгоритмічної візуалізації залежить не лише від якості графічного відображення, а й від рівня активної взаємодії користувача з навчальним матеріалом [26], [27]. Можливість змінювати вхідні дані, керувати виконанням алгоритму та аналізувати його окремі кроки сприяє глибшому розумінню логіки обчислювального процесу.

Сучасні вебтехнології дають можливість реалізувати інтерактивні засоби навчання, у яких візуалізація поєднується з активною взаємодією користувача із системою [12]. Користувач може обирати алгоритм, змінювати вхідні дані, керувати швидкістю демонстрації та спостерігати за покроковим виконанням процесу. Такий підхід працює добре, оскільки користувач бачить результат своїх дій одразу — це спрощує сприйняття складних алгоритмічних процесів і показує, навіщо потрібні інтерактивні вебресурси та тренажери для вивчення алгоритмів та структур даних.

1.3 Огляд існуючих інтернет-ресурсів, що характеризують предметну область

У сучасних умовах розвитку інформаційних технологій зростає потреба у використанні інноваційних підходів до навчання програмуванню, зокрема у вивченні алгоритмів та структур даних. Одним із ефективних способів підвищення якості засвоєння таких тем є використання спеціалізованих інтернет-ресурсів, які надають можливість візуально спостерігати за виконанням алгоритмів, аналізувати зміни в структурах даних та взаємодіяти з навчальним матеріалом у зручному інтерактивному форматі [20], [21], [22]. Аналіз існуючих вебресурсів у цій предметній області дає змогу визначити

сучасні підходи до реалізації подібних систем, оцінити їх функціональні можливості та сформувати основу для подальшої розробки власного програмного продукту.

Sort Visualizer є спеціалізованим вебресурсом, призначеним для візуалізації алгоритмів сортування. Основною особливістю цього ресурсу є наочне відображення процесу впорядкування елементів у режимі анімації, що дозволяє користувачу спостерігати за послідовністю виконання обраного алгоритму та зміною положення елементів масиву на кожному кроці. Такий підхід сприяє кращому розумінню принципів роботи алгоритмів сортування та відмінностей між ними. На вебресурсі представлені різні алгоритми сортування, серед яких бульбашкове сортування, сортування вибором, сортування вставками, швидке сортування, сортування злиттям, пірамідальне сортування та інші. Користувач має можливість обрати конкретний алгоритм, змінити кількість елементів, налаштувати швидкість анімації та спостерігати за процесом сортування у візуальному форматі. Це дозволяє не лише ознайомитися з алгоритмами, а й порівняти особливості їх роботи на однакових наборах даних. Інтерфейс ресурсу Sort Visualizer наведено на рис. 1.1.

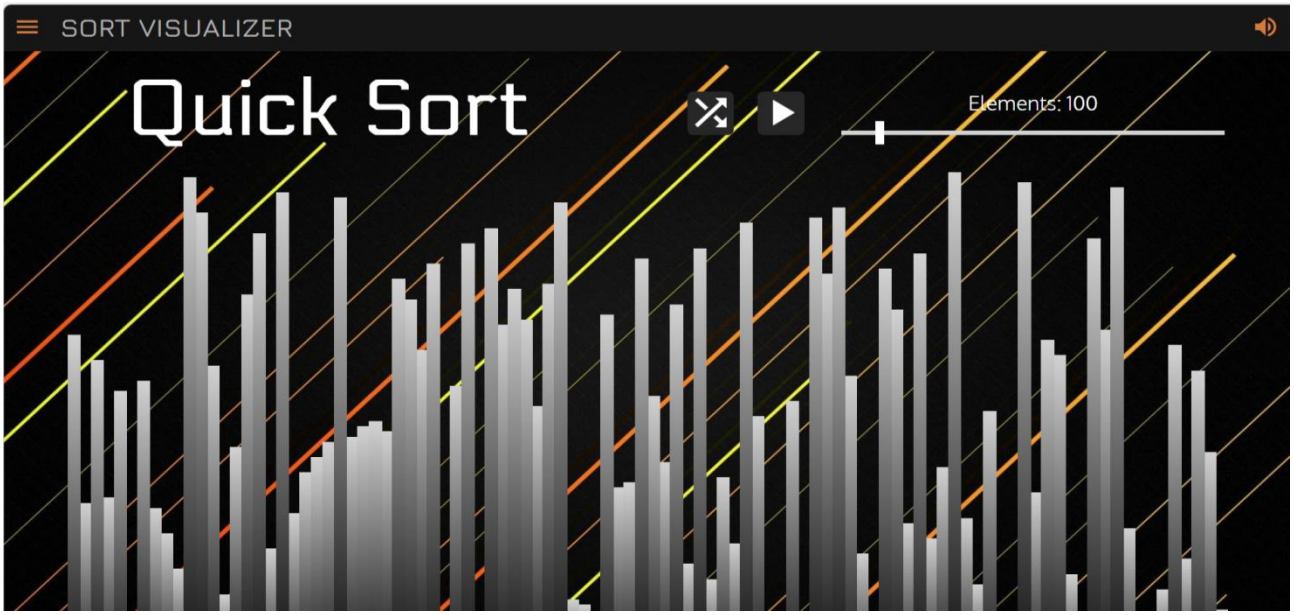

Рисунок 1.1 – Інтерфейс вебресурсу Sort Visualizer

Ресурс має простий інтерфейс і зосереджений переважно на алгоритмах сортування, що робить його зручним для початкового ознайомлення з даною темою. Водночас його функціональність пов’язана саме з сортуванням, тому інші алгоритми та структури даних на ньому представлені не так широко [20].

Algorithm Visualizer є інтерактивним вебресурсом, який призначений для візуалізації роботи алгоритмів на основі програмного коду. Особливістю цього ресурсу є поєднання текстового подання алгоритму з його наочним відображенням, що дозволяє користувачу одночасно спостерігати за логікою виконання коду та зміною стану елементів у процесі роботи алгоритму. Такий підхід забезпечує більш глибоке розуміння взаємозв’язку між програмною реалізацією та результатом її виконання. Ресурс надає можливість працювати з готовими прикладами алгоритмів, а також аналізувати їх покрокове виконання у візуальному форматі. Algorithm Visualizer охоплює різні категорії алгоритмів, зокрема алгоритми сортування, пошуку та інші підходи до обробки даних. Платформа орієнтована не лише на демонстрацію результату, а й на розуміння внутрішньої структури алгоритму через його програмний опис. Інтерфейс ресурсу Algorithm Visualizer наведено на рис. 1.2.

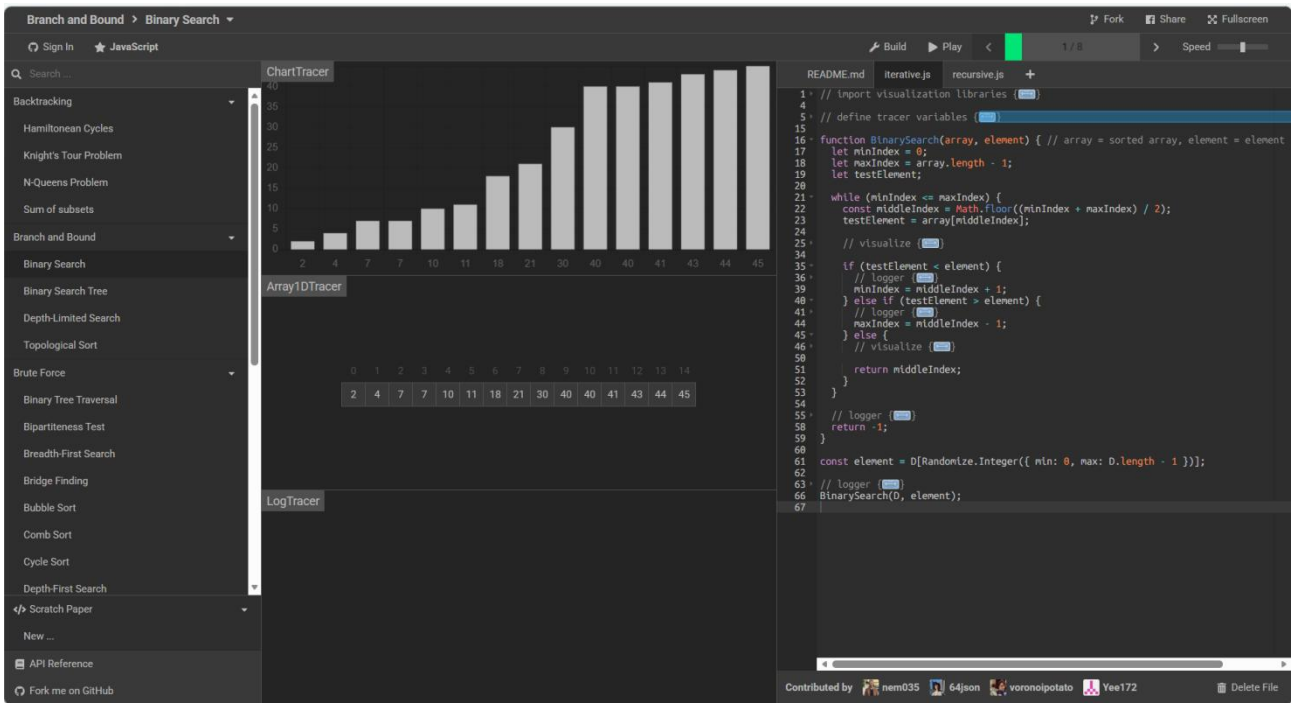

Рисунок 1.2 – Інтерфейс вебресурсу Algorithm Visualizer

Поєднання візуалізації з кодом робить цей ресурс особливо корисним для користувачів, які вже мають базові знання програмування і прагнуть глибше зрозуміти механізм роботи алгоритмів. Разом із тим такий формат подання матеріалу може бути менш зручним для початкового ознайомлення, оскільки потребує розуміння логіки програмного коду [21].

VisuAlgo є комплексним освітнім вебресурсом, який призначений для візуалізації алгоритмів та структур даних у зручному інтерактивному форматі. Цей ресурс охоплює широкий спектр тем, серед яких алгоритми сортування, пошуку, графові алгоритми, дерева, черги, стеки та інші структури даних. Основною особливістю VisuAlgo є поєднання візуальної демонстрації, теоретичних пояснень та елементів тренування, що робить його корисним не лише для ознайомлення з темою, а й для поглибленого навчання. Користувач має можливість обирати окремі алгоритми чи структури даних, спостерігати за їх покроковим виконанням, змінювати параметри демонстрації та аналізувати поведінку окремих елементів у процесі роботи. Крім того, вебресурс містить додаткові пояснення, навчальні матеріали та тренувальні елементи, що сприяють формуванню більш цілісного розуміння принципів алгоритмізації. Завдяки цьому VisuAlgo може використовуватися як допоміжний інструмент у навчальному процесі під час вивчення дисциплін, пов’язаних з програмуванням. Інтерфейс ресурсу VisuAlgo наведено на рис. 1.3.

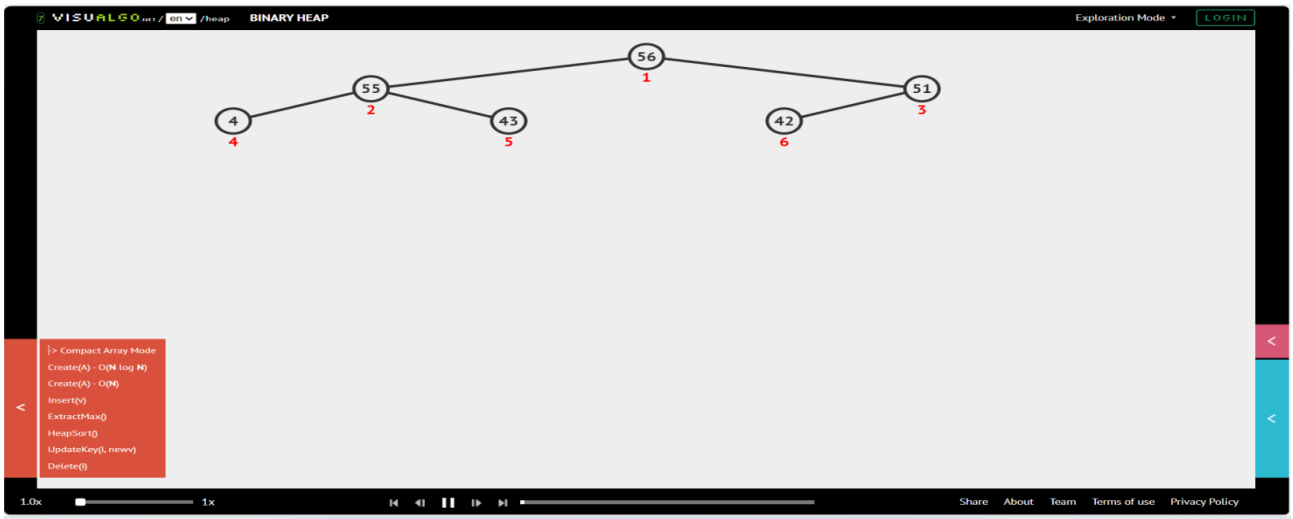

Рисунок 1.3 – Інтерфейс вебресурсу VisuAlgo

Ресурс охоплює велику кількість алгоритмів і структур даних та має досить широкі функціональні можливості. Разом із тим його інтерфейс і значний обсяг матеріалу можуть вимагати більше часу для початкового ознайомлення та адаптації користувача [22].

1.4 Аналіз переваг та недоліків оглянутих вебресурсів

Порівняння розглянутих вебресурсів представлене у таблиці 1.1 показує, що кожен із них реалізує власний підхід до візуалізації алгоритмів та структур даних. Sort Visualizer вирізняється простотою інтерфейсу та зручністю у використанні, проте має вузьку спеціалізацію, оскільки орієнтований переважно на алгоритми сортування. Algorithm Visualizer забезпечує поєднання програмного коду з візуальним поданням процесів, що є корисним для глибшого розуміння алгоритмів, однак такий формат вимагає від користувача певного рівня підготовки. VisuAlgo є найбільш комплексним серед розглянутих ресурсів, охоплює широкий спектр алгоритмів і структур даних, але водночас має складнішу структуру та більший обсяг інформації для початкового сприйняття.

Сильними сторонами сучасних вебресурсів у цій предметній області є наочність, інтерактивність та можливість покрокового спостереження за виконанням алгоритмів. Разом із тим спільними недоліками є або обмеженість функціоналу, або складність інтерфейсу, або недостатня збалансованість між простотою використання, повнотою охоплення теми та доступністю для користувачів із різним рівнем підготовки.

Проведений аналіз показує, що є сенс розробити власний інтерактивний вебсайт-тренажер, який поєднає наочність, зручність використання та можливість вивчення як алгоритмів, так і структур даних в одному місці.

Таблиця 1.1 – Порівняльна характеристика вебресурсів для візуалізації алгоритмів та структур даних

Вебресурс	Основні можливості	Переваги	Недоліки
Sort Visualizer	Візуалізація сортування, зміна розміру масиву, керування швидкістю	Простий інтерфейс, зручний для початкового ознайомлення	Орієнтований лише на сортування, не охоплює структури даних
Algorithm Visualizer	Візуалізація алгоритмів, покрокове виконання, приклади коду	Інтерактивність, детальна візуалізація	Складний для початківців, потребує базових знань програмування
VisuAlgo	Візуалізація алгоритмів і структур даних, покрокове пояснення, навчальні елементи	Широке охоплення тем, висока наочність	Перевантажений інтерфейс, надлишок інформації для новачків

Наведена таблиця дає змогу узагальнити основні особливості розглянутих вебресурсів та визначити напрями подальшої розробки власного програмного продукту.

1.5 Постановка задачі на розробку інтерактивного вебсайту-тренажера

Проведений теоретичний аналіз і огляд існуючих вебресурсів показали, що вивчення алгоритмів та структур даних потребує наочного, доступного та інтерактивного подання навчального матеріалу. Існуючі аналоги дозволяють реалізувати окремі аспекти такого підходу, однак не завжди забезпечують оптимальне поєднання простоти використання, достатнього функціонального наповнення, доступності для користувачів із різним рівнем підготовки та комплексного охоплення алгоритмів і структур даних.

Тому є сенс розробити власний інтерактивний вебсайт-тренажер, орієнтований на наочне вивчення алгоритмів та структур даних у зручному вебсередовищі. Основне призначення такого програмного продукту полягає у забезпеченні покрокової візуалізації роботи алгоритмів, відображенні змін у структурах даних та створенні умов для активної взаємодії користувача з навчальним матеріалом.

Вебсайт-тренажер дозволятиме вибирати алгоритми і структури даних для демонстрації, змінювати вхідні параметри, керувати швидкістю візуалізації та відображати процеси покроково через зручний інтерфейс. Він має бути зрозумілим для початківців і допомагати краще засвоювати принципи алгоритмізації.

Задача полягає у розробці інтерактивного вебсайту-тренажера для візуалізації алгоритмів та структур даних, який поєднуватиме наочність, доступність та зручність використання і може застосовуватися у навчальному процесі як допоміжний засіб вивчення основ програмування.

У першому розділі проаналізовано роль алгоритмів у програмуванні та визначено їх значення у підготовці майбутніх фахівців з програмування. Алгоритмічне мислення та вміння працювати зі структурами даних є базовими навичками для розробника, а традиційні текстові способи подання матеріалу не завжди дають достатню наочність — особливо коли йдеться про послідовність змін у даних на кожному кроці виконання.

Проаналізовано роль візуалізації у вивченні алгоритмів та структур даних. Покрокове графічне відображення процесу дозволяє студенту простежити логіку кожної операції, порівняти роботу різних алгоритмів на однакових даних і легше зрозуміти складні теми незалежно від рівня початкової підготовки.

Здійснено огляд трьох існуючих вебресурсів — Sort Visualizer, Algorithm Visualizer та VisuAlgo. Кожен з них має свої сильні сторони, але й обмеження: Sort Visualizer орієнтований лише на сортування, Algorithm Visualizer вимагає

базових знань програмування для роботи з кодом, а VisuAlgo, попри широке охоплення тем, має складний для новачка інтерфейс і великий обсяг інформації.

Отримані результати аналізу стали основою для формулювання конкретних вимог до власного продукту: поєднання простоти інтерфейсу Sort Visualizer, інтерактивності Algorithm Visualizer та широти охоплення VisuAlgo, без притаманних їм обмежень.

На основі проведеного аналізу прийнято рішення розробити власний інтерактивний вебсайт-тренажер, який поєднає наочність, простоту використання та охоплення як алгоритмів, так і структур даних в одному середовищі.

РОЗДІЛ 2 ПРОЄКТУВАННЯ ВЕБСАЙТУ-ТРЕНАЖЕРА

2.1 Архітектура та структура вебсайту-тренажера

Вебсайт-тренажер реалізовано як клієнтський (frontend-only) застосунок, що не потребує серверної частини або бази даних. Весь функціонал виконується безпосередньо у браузері користувача за допомогою технологій HTML, CSS та JavaScript. Такий підхід має кілька переваг. По-перше, він суттєво спрощує розгортання: сайт можна розмістити на будь-якому хостингу для статичних файлів без необхідності налаштовувати серверне середовище. По-друге, клієнтський підхід забезпечує швидкість відклику, адже жодні запити до зовнішніх API не впливають на час відображення анімацій та інтерактивних елементів. По-третє, відсутність бекенду знижує складність системи і полегшує її підтримку.

Загальна архітектура продукту побудована за принципом розподілу обов'язків: HTML відповідає за структуру сторінок, CSS — за їх зовнішній вигляд і адаптивність, JavaScript — за логіку роботи алгоритмів, управління станом інтерфейсу та анімаційні ефекти. Між цими трьома рівнями відсутня жорстка прив'язка: HTML-розмітка не містить вбудованих стилів, а JavaScript-логіка чітко відокремлена від верстки, що полегшує розвиток і масштабування продукту.

Структурно сайт складається з двох основних сторінок: «Алгоритми» та «Структури даних». Кожна сторінка є окремим HTML-файлом і містить власні секції з підключеними стилями та скриптами. Такий підхід дозволяє завантажувати лише потрібні ресурси, не перевантажуючи браузер зайвим кодом. Загальна архітектура розробленого вебсайту-тренажера представлена на рис. 2.1. На рисунку 2.1 показана загальна архітектура розробленого вебсайту-тренажера.

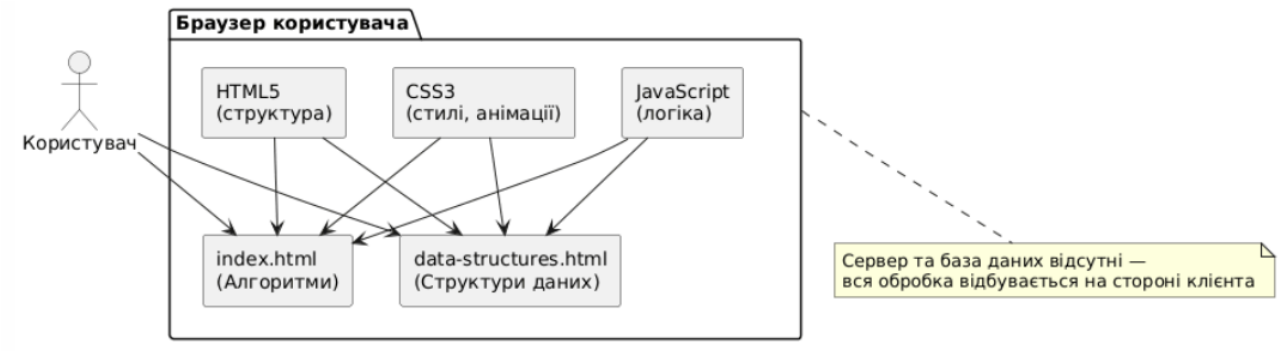

Рисунок 2.1 – Архітектура вебсайту-тренажера (клієнтський застосунок)

Користувач взаємодіє з браузером, який завантажує HTML-сторінки разом із CSS-стилями та JavaScript-скриптами. Уся обробка запитів та логіка виконання алгоритмів відбувається на стороні клієнта — додатковий сервер не задіяний.

2.2 Вибір засобів і технологій розробки

Для реалізації вебсайту-тренажера обрано стек технологій, що є стандартним для клієнтської веброзробки та не потребує додаткових залежностей або пакетних менеджерів. Всі три компоненти — HTML5, CSS3 та vanilla JavaScript — підтримуються всіма сучасними браузерами без необхідності встановлення плагінів або транспіляції.

HTML5 використовується для побудови семантичної розмітки сторінок. Семантичні теги (<header>, <main>, <section>, <nav>) забезпечують логічну структуру документу, що є корисним як для зрозумілості коду, так і для пошукової оптимізації. Форми введення, кнопки та поля реалізовані стандартними елементами HTML, що гарантує коректне відображення у різних браузерах [10], [14], [17].

CSS3 відповідає за оформлення інтерфейсу. Стилі організовані таким чином, що загальні правила, наприклад, кольорова схема і типографіка, описані в окремому файлі, тоді як специфічні стилі для кожної секції знаходяться поряд із відповідними компонентами. Для анімацій використовуються CSS-transitions та keyframe-анімації, що забезпечує плавну роботу без зайвого навантаження на JavaScript [11], [15], [18].

JavaScript реалізує всю логіку програмного продукту: обробку подій, виконання алгоритмів, управління станом анімації та взаємодію з DOM. Код написаний без сторонніх бібліотек, що підвищує продуктивність і знижує ризик конфліктів залежностей. Окремі модулі відповідають за логіку кожного алгоритму або структури даних, що спрощує підтримку та розширення функціоналу [12], [13], [16], [19]. Технологічний стек вебсайту-тренажера наведено в таблиці 2.1.

Таблиця 2.1 – Технологічний стек вебсайту-тренажера

Технологія	Версія / стандарт	Призначення у проєкті
HTML5	HTML Living Standard	Структура та розмітка сторінок
CSS3	CSS Level 3	Стилізація, анімації, адаптивний дизайн
JavaScript	ES6+	Логіка алгоритмів, анімація, управління DOM
Netlify	Static Hosting	Хостинг та публікація сайту в мережі
VS Code	—	Середовище розробки

У таблиці 2.1 наведено основний технологічний стек проєкту. Вибір кожного інструменту обумовлений практичністю та відповідністю задачам: для навчального вебтренажера важливо мінімізувати залежності та забезпечити стабільну роботу у будь-якому сучасному браузері без додаткових налаштувань.

2.3 Функціональні та нефункціональні вимоги

На основі проведеного аналізу предметної області та особливостей існуючих аналогів було сформульовано вимоги до розробленого вебсайту-тренажера. Вимоги поділяються на функціональні — тобто те, що система повинна робити, — і нефункціональні — обмеження та характеристики якості.

До функціональних вимог відносяться такі: можливість вибору алгоритму або структури даних зі списку; введення користувацьких вхідних даних або

генерація випадкових; покрокова візуалізація виконання з поясненням кожного кроку; керування процесом демонстрації (запуск, пауза, зупинка, крок вперед, крок назад, скидання); регулювання швидкості анімації; відображення псевдокоду та блок-схеми для алгоритмів; виконання операцій зі структурами даних (додавання, видалення, пошук тощо) з візуальним відображенням змін стану; навігація між розділами сайту. Перелік функціональних вимог до вебсайту-тренажера наведено в таблиці 2.2.

Таблиця 2.2 – Функціональні вимоги до вебсайту-тренажера

№	Вимога	Пріоритет
1	Вибір алгоритму або структури даних зі списку	Високий
2	Введення власних вхідних даних користувачем	Високий
3	Генерація випадкових вхідних даних	Середній
4	Покрокова візуалізація з поясненням кроку	Високий
5	Кнопки керування (старт, пауза, стоп, крок вперед/назад, скидання)	Високий
6	Регулювання швидкості анімації	Середній
7	Відображення псевдокоду алгоритму	Середній
8	Відображення блок-схеми алгоритму	Середній
9	Виконання операцій зі структурами даних із візуалізацією	Високий
10	Навігація між розділами сайту	Високий

Нефункціональні вимоги охоплюють такі аспекти: сумісність із сучасними браузерами (Chrome, Firefox, Edge, Safari); коректна робота без підключення до Інтернету після завантаження сторінки; зрозумілий і привабливий інтерфейс для користувачів із різним рівнем підготовки; час відклику на дію користувача не повинен перевищувати 100 мс; код повинен бути організованим і зрозумілим для подальшої підтримки. Перелік нефункціональних вимог до вебсайту-тренажера наведено в таблиці 2.3.

Таблиця 2.3 – Нефункціональні вимоги до вебсайту-тренажера

№	Вимога	Категорія	Спосіб перевірки
1	Сумісність із сучасними браузерами (Chrome, Firefox, Edge, Safari)	Переносимість	Ручне тестування у кожному браузері
2	Коректна робота без підключення до Інтернету після завантаження сторінки	Надійність	Перевірка роботи в офлайн-режимі
3	Зрозумілий і привабливий інтерфейс для користувачів із різним рівнем підготовки	Зручність використання	Експертна оцінка інтерфейсу
4	Час відклику на дію користувача не повинен перевищувати 100 мс	Продуктивність	Вимірювання часу відгуку інтерфейсу
5	Код повинен бути організованим і зрозумілим для подальшої підтримки	Підтримуваність	Аналіз структури файлів та модульності коду
6	Адаптивність інтерфейсу для різних розмірів екрана	Зручність використання	Перевірка відображення на різних роздільних здатностях

Кожна з наведених вимог враховувалась на етапі проєктування та реалізації вебсайту-тренажера. Сумісність із сучасними браузерами забезпечується використанням стандартизованих технологій HTML5, CSS3 та ES6+, які підтримуються всіма основними браузерами без додаткових поліфілів. Відсутність звернень до зовнішніх API та серверної частини гарантує коректну роботу сторінки після її первинного завантаження навіть за відсутності мережевого з'єднання.

Зручність використання забезпечується послідовною структурою сторінок, темною кольоровою схемою та мінімалістичним інтерфейсом, що

зменшує кількість елементів, які відволікають увагу користувача. Продуктивність інтерфейсу досягається завдяки попередньому обчисленню всіх кроків алгоритмів — це дозволяє уникнути затримок при покроковій навігації, оскільки кожен крок уже міститься в масиві результатів і відображається без додаткових обчислень у реальному часі.

Підтримуваність коду забезпечується розподілом логіки між файлами script.js та structures.js відповідно до функціонального призначення, а також використанням окремих функцій для кожного алгоритму чи операції зі структурою даних. Адаптивність інтерфейсу реалізована засобами CSS3 з використанням гнучких одиниць вимірювання та медіазапитів, що дозволяє коректно відображати сторінки на пристроях з різними розмірами екрана — від мобільних телефонів до настільних моніторів.

2.4 Проєктування структури інтерфейсу

При проєктуванні інтерфейсу вебсайту-тренажера ключовим орієнтиром стала зручність користувача: мінімальна кількість елементів, які відволікають увагу, та максимальна концентрація корисної інформації в межах видимої області екрана. Темна кольорова схема обрана як основа дизайну, оскільки вона знижує навантаження на зір під час тривалої роботи з інтерфейсом і водночас надає продукту сучасного та лаконічного вигляду.

Окремої уваги заслуговує обґрунтування кольорового маркування елементів у зоні візуалізації. Для полегшення сприйняття процесу виконання алгоритмів використано єдину кольорову логіку: помаранчевим кольором позначаються елементи, які порівнюються на поточному кроці, що привертає увагу користувача до ключового моменту операції; зеленим кольором виділяються елементи, що вже зайняли своє остаточне місце у впорядкованому масиві, формуючи візуальне відчуття прогресу виконання; синій колір використовується для елементів, які на поточному кроці не беруть участі в активній операції. Обрані кольори мають достатній контраст відносно темного

фону, що забезпечує їх розрізнюваність і відповідає принципам доступності інтерфейсу.

Кольорове виділення доповнюється текстовим поясненням поточного кроку, у якому зазначено, які саме елементи порівнюються або яку операцію виконує алгоритм. Завдяки цьому користувач може простежувати процес не лише за зміною кольорів, а й за текстовим описом, що особливо корисно для користувачів із незначними порушеннями кольоросприйняття.

Кожна сторінка сайту має однакову структуру: верхня навігаційна панель з логотипом і посиланнями на розділи, ознайомчий блок із коротким описом розділу, основна робоча область із панеллю керування та зоною візуалізації, а також додаткові блоки з поясненнями, псевдокодом і блок-схемами. Така структура є передбачуваною для користувача і не вимагає додаткового вивчення інтерфейсу.

Панель керування розташована у верхній частині робочої зони і містить: список вибору алгоритму або структури даних, поле введення вхідних даних, поле для значення пошуку (для пошукових алгоритмів), кнопки управління демонстрацією та повзунок швидкості. Зона візуалізації займає центральну частину сторінки і відображає анімовану графіку, а блок пояснення поточного кроку розташований безпосередньо під нею. На рисунку 2.2 показано схему взаємодії користувача з програмним продуктом. Користувач обирає алгоритм або структуру даних, вводить параметри та запускає демонстрацію. JavaScript-логіка обробляє дії користувача, оновлює стан анімації та виводить відповідні пояснення.

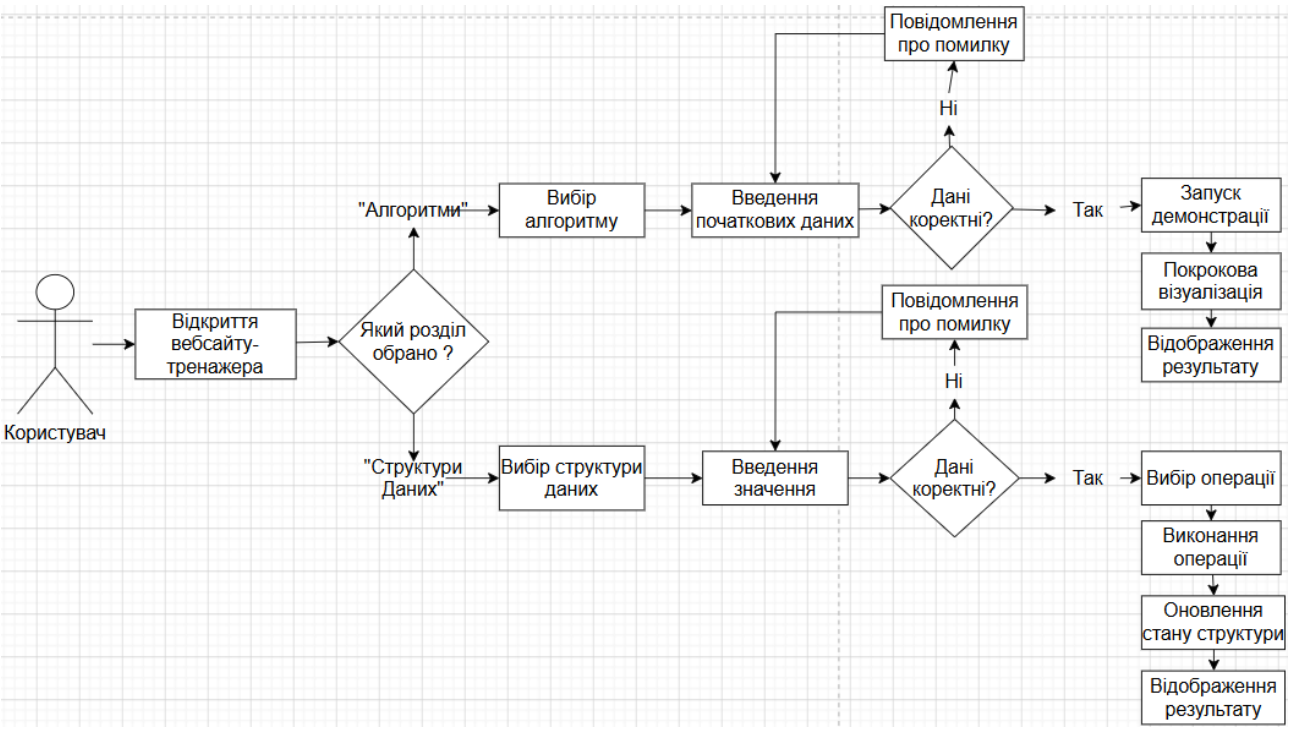

Рисунок 2.2 – Схема взаємодії користувача з вебсайтом-тренажером

Усі ці процеси відбуваються синхронно і не потребують мережевих запитів.

2.5 Проєктування модуля «Алгоритми»

Модуль «Алгоритми» є першим розділом вебсайту-тренажера і охоплює десять алгоритмів, які розподілені на дві групи: алгоритми сортування та алгоритми пошуку.

До групи алгоритмів сортування входять: Bubble Sort (сортування бульбашкою), Selection Sort (сортування вибором), Insertion Sort (сортування вставками), Merge Sort (сортування злиттям) та Quick Sort (швидке сортування). До групи алгоритмів пошуку включено: Linear Search (лінійний пошук), Binary Search (бінарний пошук), Jump Search (стрибковий пошук), Interpolation Search (інтерполяційний пошук) та Exponential Search (експоненціальний пошук) [6].

Кожен алгоритм у модулі має однотипну структуру відображення. Після вибору алгоритму зі списку користувач бачить: поле введення початкових даних у вигляді числового масиву; кнопки керування; зону візуалізації у вигляді кольорових стовпчиків, де висота кожного стовпця відповідає значенню

елементу масиву; поле пояснення поточного кроку; блок із прикладом роботи алгоритму; псевдокод; блок-схему для певних алгоритмів.

Принципи кольорового маркування елементів під час візуалізації описано в підрозділі 2.4. Таке маркування допомагає користувачеві стежити за логікою роботи алгоритму і розуміти, що відбувається на кожному кроці.

2.6 Проєктування модуля «Структури даних»

Модуль «Структури даних» є другим основним розділом сайту і охоплює шість структур: Stack (стек), Queue (черга), Linked List (однозв'язний список), Deque (дек), Binary Search Tree (бінарне дерево пошуку) та Graph (граф) [8], [9].

На відміну від алгоритмів, де демонстрація відбувається у покроковому автоматичному режимі, у модулі структур даних взаємодія є більш інтерактивною: користувач самостійно вводить значення і натискає кнопки операцій, а сайт відображає результат цієї операції у вигляді анімації та повідомлення про статус виконання.

Для кожної структури даних передбачені специфічні операції. Для стеку — push та pop; для черги — enqueue та dequeue; для двозв'язного списку — додавання на початок, у кінець та видалення; для деку — додавання і видалення з обох кінців; для бінарного дерева пошуку — вставка, видалення та пошук; для графа — додавання вузла та ребра, виконання обходів BFS та DFS.

Зона візуалізації відображає поточний стан структури у вигляді графічних елементів: прямокутників для стеку і черги, схематичного зв'язного списку з вузлами та стрілками для Linked List і Deque, дерева та графа для відповідних структур. Кожна зміна стану супроводжується анімацією, яка допомагає зрозуміти, які саме вузли або зв'язки зазнали змін.

У другому розділі виконано проєктування вебсайту-тренажера. Архітектуру обрано як клієнтський застосунок без бекенду — весь функціонал виконується безпосередньо в браузері користувача, без звернень до серверів чи зовнішніх API. Такий підхід значно спростив розгортання: сайт можна

розмістити на будь-якому хостингу для статичних файлів, а робота алгоритмів та анімацій не залежить від швидкості мережі.

Як технологічний стек обрано HTML5, CSS3 та vanilla JavaScript — без використання сторонніх бібліотек чи фреймворків. HTML5 забезпечив семантичну структуру сторінок, CSS3 — оформлення, кольорову схему та анімації за допомогою transitions і keyframes, а JavaScript — усю логіку: обробку дій користувача, виконання алгоритмів і керування станом інтерфейсу. Розподіл між трьома рівнями дозволив тримати код організованим: розмітка не містить вбудованих стилів, а логіка чітко відокремлена від верстки.

Сформульовано функціональні та нефункціональні вимоги до системи. До функціональних увійшли: вибір алгоритму чи структури даних зі списку, введення власних або генерація випадкових вхідних даних, покрокова візуалізація з поясненням кожного кроку, повний набір елементів керування демонстрацією (старт, пауза, стоп, крок вперед/назад, скидання, регулювання швидкості), а також відображення псевдокоду та блок-схем. Серед нефункціональних вимог — сумісність із сучасними браузерами, робота без підключення до Інтернету після завантаження сторінки та швидкий відгук інтерфейсу на дії користувача (до 100 мс).

Розроблено структуру інтерфейсу обох основних модулів — «Алгоритми» та «Структури даних». Обрано темну кольорову схему, яка зменшує навантаження на очі під час тривалої роботи. Кожна сторінка має однакову структуру: навігаційна панель, ознайомчий блок, панель керування, зона візуалізації та блок пояснень. Для модуля «Алгоритми» спроєктовано покрокову автоматичну демонстрацію з кольоровим маркуванням елементів (порівнювані, переставлювані, відсортовані), а для модуля «Структури даних» — інтерактивну взаємодію, де користувач самостійно виконує операції, а система відображає результат у вигляді анімації та статусу виконання.

Прийняті проєктні рішення стали основою для практичної реалізації, описаної у третьому розділі.

РОЗДІЛ 3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ВЕБСАЙТУ-ТРЕНАЖЕРА

3.1 Розробка функціональної структури проєкту

Проєкт організовано у вигляді набору статичних файлів без використання систем збірки або пакетних менеджерів. Така структура забезпечує простоту розгортання і відтворюваність проєкту без додаткових залежностей. Структура файлів проекту наведена в таблиці 3.1.

Таблиця 3.1 – Структура файлів проєкту

Файл / директорія	Призначення
index.html	Сторінка «Алгоритми», на якій реалізовано вибір алгоритмів сортування та пошуку, панель керування, візуалізацію, пояснення, приклади роботи, псевдокод і блок-схеми.
data-structures.html	Сторінка «Структури даних», на якій реалізовано вибір структур даних, інтерактивні операції, статус виконання, псевдокод і візуальне відображення поточного стану структури.
style.css	Основний файл стилів, який відповідає за оформлення сторінок, кольорову схему, розміщення елементів, кнопки, панелі, картки, фон і адаптацію інтерфейсу.
script.js	JavaScript-файл, який містить логіку роботи сторінки «Алгоритми»: вибір алгоритмів, обробку введених даних, запуск демонстрації, покрокове виконання, паузу, скидання та візуалізацію.
structures.js	JavaScript-файл, який містить логіку роботи сторінки «Структури даних»: операції зі стеком, чергою, зв’язним списком, деком, бінарним деревом пошуку та графом.
images/	Директорія для графічних матеріалів сайту: логотипа, зображень, блок-схем та інших візуальних елементів.

У таблиці 3.1 наведено структуру основних файлів проєкту. Сайт реалізовано як статичний вебзастосунок без серверної частини. Основна сторінка index.html відповідає за розділ «Алгоритми», а сторінка data-structures.html — за розділ «Структури даних». Оформлення обох сторінок винесено в окремий файл style.css, що забезпечує єдиний стиль інтерфейсу. Логіка роботи розділів розділена між файлами script.js та structures.js, що полегшує підтримку коду та подальше розширення функціоналу.

3.2 Реалізація навігації та загального макета

Навігаційна панель є спільним елементом для обох сторінок сайту. Вона містить логотип та посилання на розділи «Алгоритми» і «Структури даних». Реалізація виконана за допомогою тегу <nav>, а оформлення навігації задано у файлі style.css.

Лістинг 3.1 – Фрагмент HTML-структури навігаційної панелі

```
<nav class="top-nav" aria-label="Основна навігація">
  <a href="index.html" class="active" aria-current="page">Алгоритми</a>
  <a href="data-structures.html">Структури даних</a>
</nav>
```

Лістинг 3.1 демонструє HTML-структуру навігаційної панелі вебсайту. Посилання index.html відкриває сторінку «Алгоритми», а data-structures.html — сторінку «Структури даних». Клас active використовується для позначення поточної сторінки, а атрибут aria-current="page" вказує, який розділ зараз відкритий.

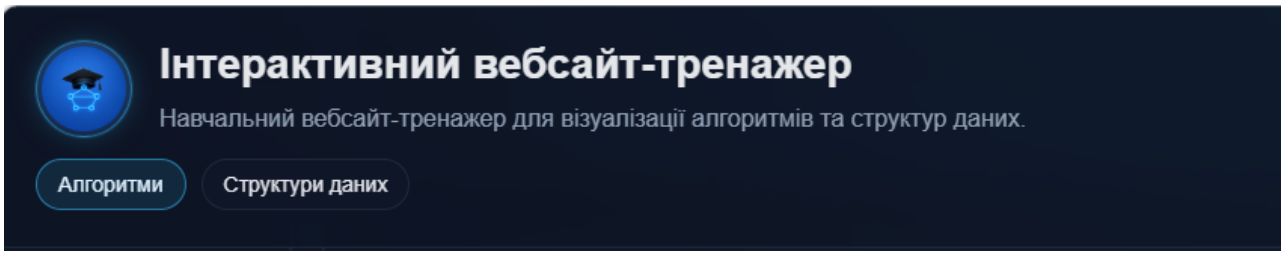

Рисунок 3.1 – Навігаційна панель вебсайту-тренажера

На рисунку 3.1 зображена навігаційна панель вебсайту-тренажера. Вона містить логотип, назву сайту, короткий опис і кнопки переходу між сторінками «Алгоритми» та «Структури даних». Така структура дозволяє користувачу швидко перемикатися між основними сторінками сайту.

3.3 Реалізація сторінки «Алгоритми»

Сторінка «Алгоритми» є основним робочим інструментом для вивчення алгоритмів сортування та пошуку. Її можна умовно розділити на дві частини: панель управління та зону візуалізації.

Панель управління містить: випадаючий список вибору алгоритму; текстове поле для введення масиву чисел через кому; поле для введення значення пошуку (відображається лише для алгоритмів пошуку); кнопки «Автоматичний запуск», «Пауза / Стоп», «Крок вперед», «Крок назад», «Скинути»; кнопку «Випадкові дані»;

Лістинг 3.2 – Фрагмент HTML-коду панелі керування сторінки «Алгоритми»

```
<div class="control-group">
  <label for="algorithmSort">Алгоритм сортування</label>
  <select id="algorithmSort">
    <option value="">Оберіть алгоритм сортування</option>
    <option value="bubble">Bubble Sort</option>
    <option value="selection">Selection Sort</option>
    <option value="insertion">Insertion Sort</option>
    <option value="merge">Merge Sort</option>
    <option value="quick">Quick Sort</option>
  </select>
</div>

<div class="control-group">
  <label for="algorithmSearch">Алгоритм пошуку</label>
  <select id="algorithmSearch">
    <option value="">Оберіть алгоритм пошуку</option>
    <option value="linear">Linear Search</option>
    <option value="binary">Binary Search</option>
    <option value="jump">Jump Search</option>
    <option value="interpolation">Interpolation Search</option>
    <option value="exponential">Exponential Search</option>
  </select>
</div>
```

У лістингу 3.2 наведено фрагмент HTML-коду панелі керування сторінки «Алгоритми». Для зручності користувача алгоритми поділено на дві

групи: алгоритми сортування та алгоритми пошуку. Такий поділ спрощує вибір потрібного алгоритму та робить інтерфейс зрозумілішим для навчального використання. Списки мають початкові пункти «Оберіть алгоритм сортування» та «Оберіть алгоритм пошуку», тому після відкриття сторінки користувач спочатку бачить ознайомчий блок, а вже після вибору алгоритму переходить до демонстрації.

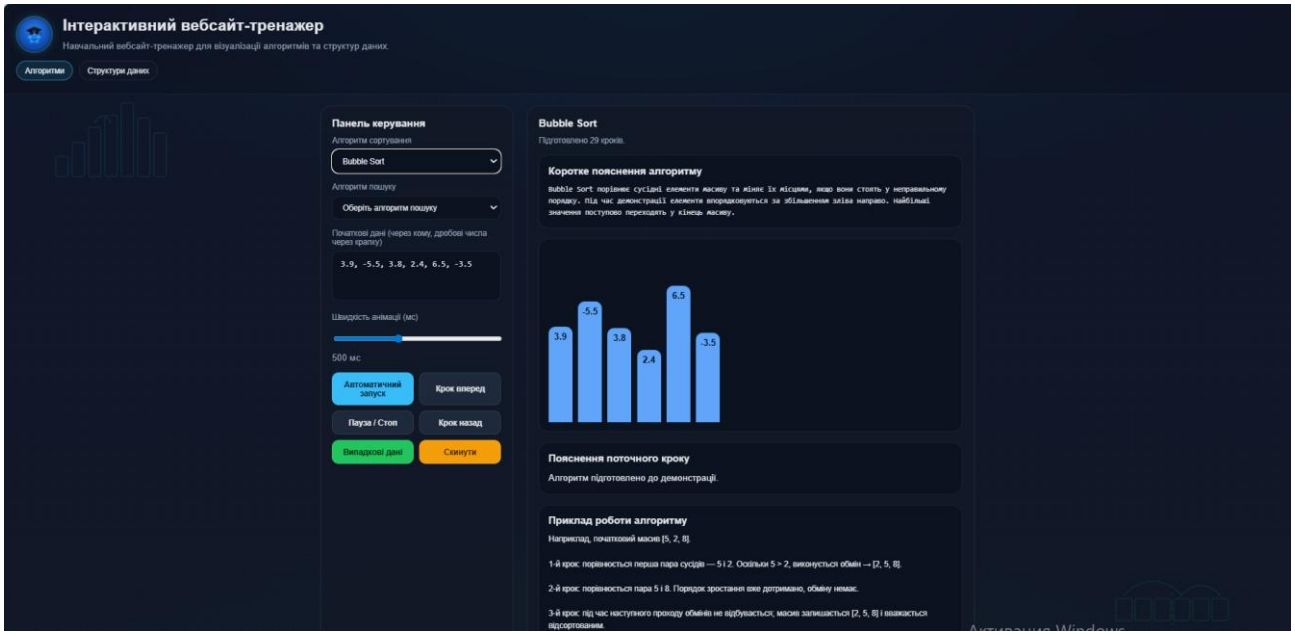

Рисунок 3.2 – Сторінка «Алгоритми» вебсайту-тренажера

На рисунку 3.2 зображено загальний вигляд сторінки «Алгоритми». Зверху розташована панель управління, у центрі — зона візуалізації зі стовпчиками, знизу — блок пояснення поточного кроку. Темний фон покращує читабельність кольорових маркувань стовпчиків. У лівій частині панелі керування розміщено випадаючі списки для вибору алгоритму сортування та пошуку, поле введення початкових даних і повзунок регулювання швидкості анімації. Кнопки керування демонстрацією — «Автоматичний запуск», «Крок вперед», «Пауза / Стоп» та «Крок назад» — дозволяють користувачеві самостійно контролювати темп перегляду алгоритму. Праворуч від зони візуалізації відображається коротке пояснення алгоритму, приклад роботи з

конкретним масивом та опис поточного кроку, що дозволяє одночасно спостерігати за анімацією і читати текстовий супровід.

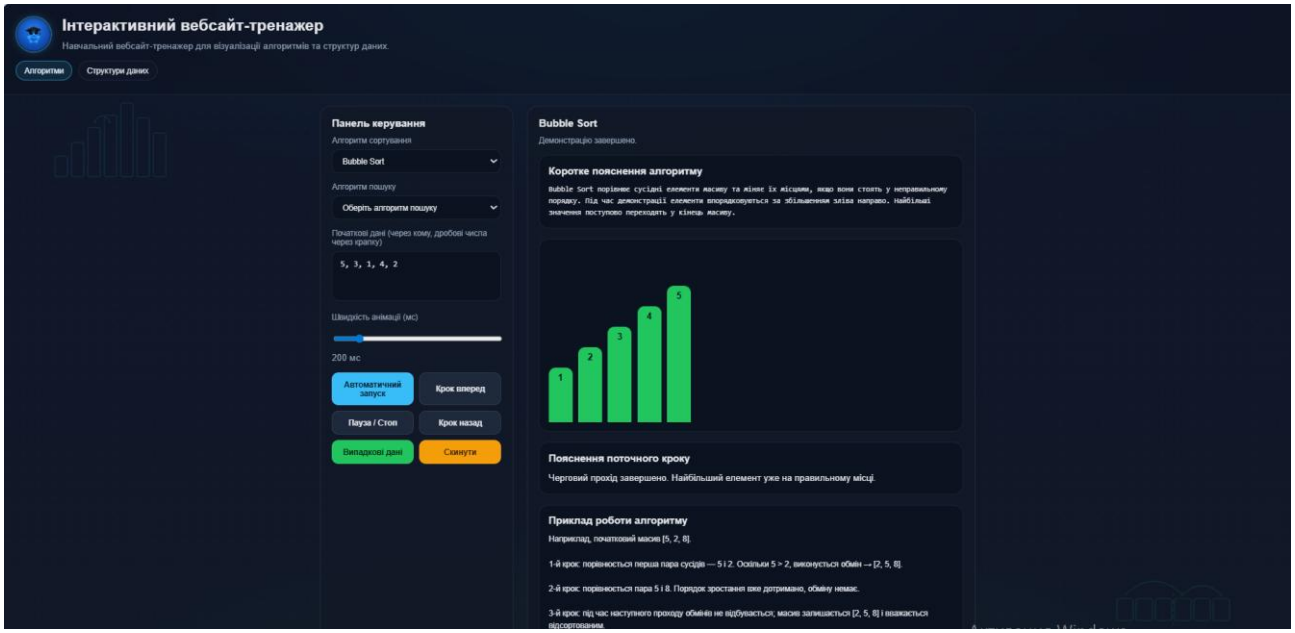

Рисунок 3.3 – Візуалізація алгоритму Bubble Sort у процесі виконання

На рисунку 3.3 продемонстровано результат виконання алгоритму Bubble Sort. У робочій області елементи масиву відображені у вигляді стовпчиків, які після виконання алгоритму розміщені за зростанням. У блоці пояснення поточного кроку система повідомляє, що черговий прохід завершено, а найбільший елемент уже знаходиться на правильному місці.

Алгоритм Bubble Sort реалізований за класичною схемою: зовнішній цикл проходить n разів, внутрішній цикл порівнює сусідні елементи і виконує переставляння, якщо лівий елемент більший за правий. Для забезпечення покрокової візуалізації алгоритм не виконується «одразу весь» — замість цього функція generateBubbleSortSteps() заздалегідь генерує всі кроки і зберігає їх у масиві. Це дозволяє реалізувати функції «крок назад» і «крок вперед» без повторного виконання алгоритму. Лістинг алгоритму предаставлений в додатку А.

У лістингу А.1 наведено функцію bubbleSteps(), яка реалізує алгоритм сортування методом обміну (Bubble Sort). Зовнішній цикл відповідає за

кількість проходів по масиву, а внутрішній — за послідовне порівняння сусідніх елементів. Для кожного порівняння формується окремий крок з поясненням, а якщо лівий елемент виявляється більшим за правий, виконується обмін значень і додається окремий крок з повідомленням про переставляння. Після завершення кожного проходу додається крок, який позначає найбільший елемент поточної непосортованої частини масиву як такий, що вже зайняв своє остаточне місце (масив done).

Аналогічно до інших алгоритмів сортування, усі кроки обчислюються заздалегідь та зберігаються у масиві out, що дозволяє реалізувати покрокову навігацію («крок вперед», «крок назад») без повторного виконання алгоритму.

Ключовим архітектурним рішенням модуля «Алгоритми» є попереднє обчислення всіх кроків до початку анімації. Функції bubbleSteps(), selectionSteps(), insertionSteps(), mergeSteps(), quickSteps() та аналогічні функції пошуку формують масив steps, де кожен елемент містить: поточний стан масиву (data), індекси активних елементів (active), індекси завершених елементів (done), текст пояснення (text) та, за потреби, індекс знайденого елемента (found). Такий підхід дозволяє реалізувати навігацію як вперед, так і назад — просто змінюючи змінну currentStepIndex, без повторного виконання алгоритму.

На рисунку 3.4 зображена візуалізація бінарного пошуку. Алгоритм виділяє поточний «середній» елемент і звужує діапазон пошуку. Різними кольорами позначено: активний діапазон, поточний середній елемент і вже відхилені частини масиву. Пояснення у нижньому блоці описує, чому алгоритм рухається вліво або вправо. Такий підхід дозволяє користувачеві наочно простежити логіку поділу масиву навпіл на кожному кроці та зрозуміти, чому бінарний пошук є значно ефективнішим за лінійний на впорядкованих масивах.

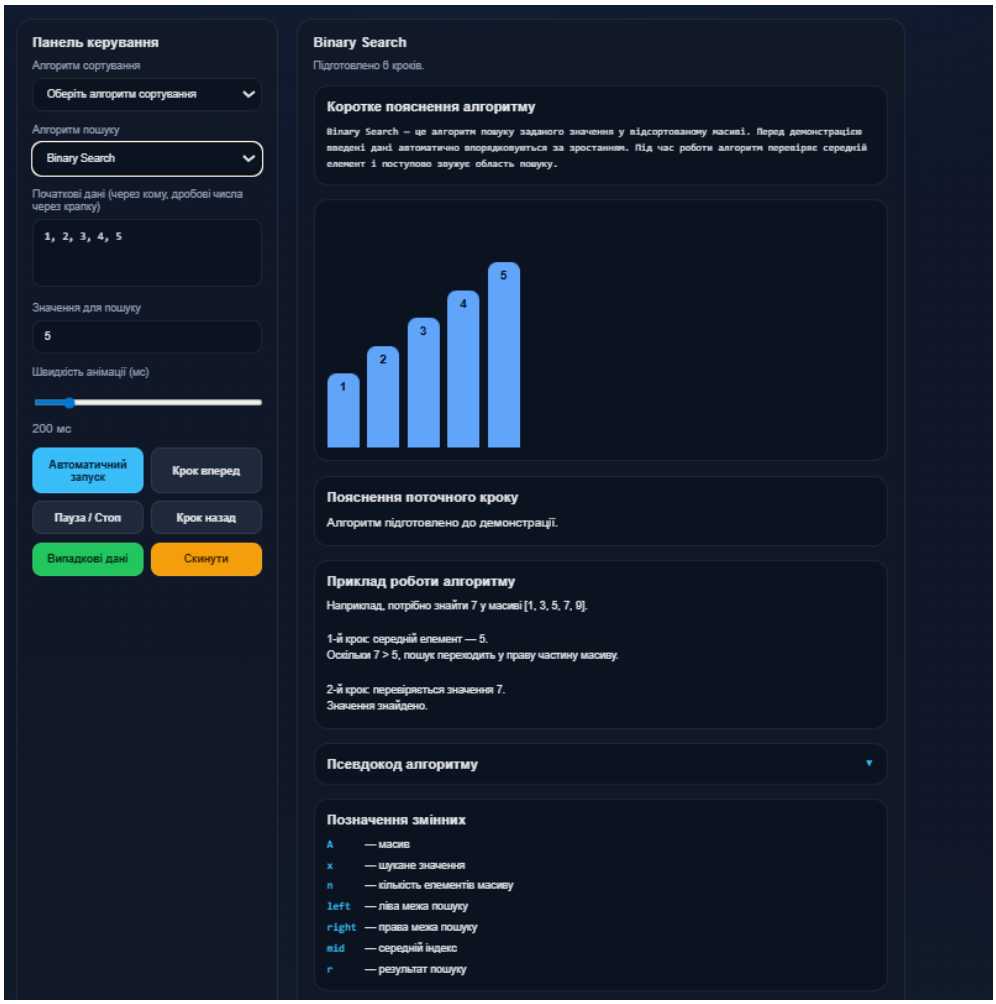

Рисунок 3.4 – Візуалізація алгоритму Binary Search

Попереднє сортування вхідних даних виконується автоматично, що забезпечує коректну роботу алгоритму незалежно від порядку введених користувачем значень. Лістинг алгоритму представлений в додатку Б.

У лістингу Б.1 представлено функцію binarySteps(), яка реалізує алгоритм бінарного пошуку (Binary Search). На початку масив сортується за зростанням, після чого визначаються межі діапазону пошуку — змінні left та right. На кожній ітерації обчислюється середній елемент діапазону, і для нього формується крок з поясненням. Якщо значення середнього елемента дорівнює шуканому, додається завершальний крок із позначкою found та повідомленням про знайдену позицію, після чого функція завершує роботу. Якщо шукане значення менше за середній елемент, права межа діапазону зменшується і пошук продовжується у лівій половині масиву; в іншому випадку — звужується

ліва межа і пошук переходить у праву половину. Якщо діапазон пошуку звужується до порожнього, додається крок з повідомленням про те, що значення відсутнє в масиві.

На відміну від алгоритмів сортування, кількість кроків бінарного пошуку залежить не лише від розміру масиву, а й від положення шуканого елемента: завдяки логарифмічній складності алгоритму кількість кроків зазвичай значно менша за кількість елементів масиву.

3.4 Реалізація псевдокоду та блок-схем

Блок-схеми додано для окремих алгоритмів як додатковий навчальний матеріал. Вони підготовлені у вигляді графічних зображень і розміщені у директорії images. Блок-схема відображається у відповідному блоці сторінки та допомагає користувачу зрозуміти послідовність виконання алгоритму, структуру умов, циклів і переходів між окремими етапами. Блок-схему алгоритму бінарного пошуку наведено в додатку В.

На рисунку В.1 зображено блок-схему алгоритму Binary Search. Вона демонструє послідовність виконання бінарного пошуку: визначення меж пошуку, обчислення середнього елемента, порівняння його зі шуканим значенням та подальше звуження діапазону пошуку. Використання блок-схеми разом із псевдокодом і візуалізацією допомагає краще зрозуміти логіку роботи алгоритму.

Для кращого розуміння роботи алгоритмів на сторінці «Алгоритми» передбачено блок із псевдокодом. Псевдокод подано у спрощеному алгоритмічному форматі з відступами, умовами та циклами. Такий спосіб подання дозволяє користувачу не лише спостерігати за візуалізацією, а й одночасно бачити логіку виконання алгоритму у текстовій формі.

Псевдокод змінюється залежно від того, який алгоритм обрано користувачем. Після вибору алгоритму користувач може відкрити відповідний блок і переглянути послідовність основних дій алгоритму.

У лістингу 3.3 наведено псевдокод алгоритму Insertion Sort, який використовується на сторінці «Алгоритми». У ньому показано, як алгоритм послідовно бере поточний елемент, порівнює його з попередніми елементами та вставляє на правильну позицію у вже впорядкованій частині масиву. Такий псевдокод допомагає користувачу зрозуміти логіку алгоритму без прив’язки до конкретної мови програмування.

Лістинг 3.3 – Фрагмент збереження псевдокоду алгоритму Insertion Sort

```
for i = 1 to n - 1
    key = A[i]
    j = i - 1
    while j >= 0 and A[j] > key
        A[j + 1] = A[j]
        j = j - 1
    A[j + 1] = key
```

3.5 Реалізація сторінки «Структури даних»

Сторінка «Структури даних» побудована за принципом, схожим до сторінки «Алгоритми»: є панель вибору структури, поле введення, кнопки операцій та зона візуалізації. Головна відмінність у тому, що стан структури даних є постійним між операціями — тобто після кожної дії користувача структура зберігає свій поточний стан, і нова операція виконується над ним.

На рисунку 3.5 зображено сторінку «Структури даних» із вибраною структурою Queue. Зона візуалізації відображає елементи черги у вигляді послідовних блоків. Перший елемент позначається як Front, а останній — як Rear, що допомагає користувачу зрозуміти принцип роботи черги.

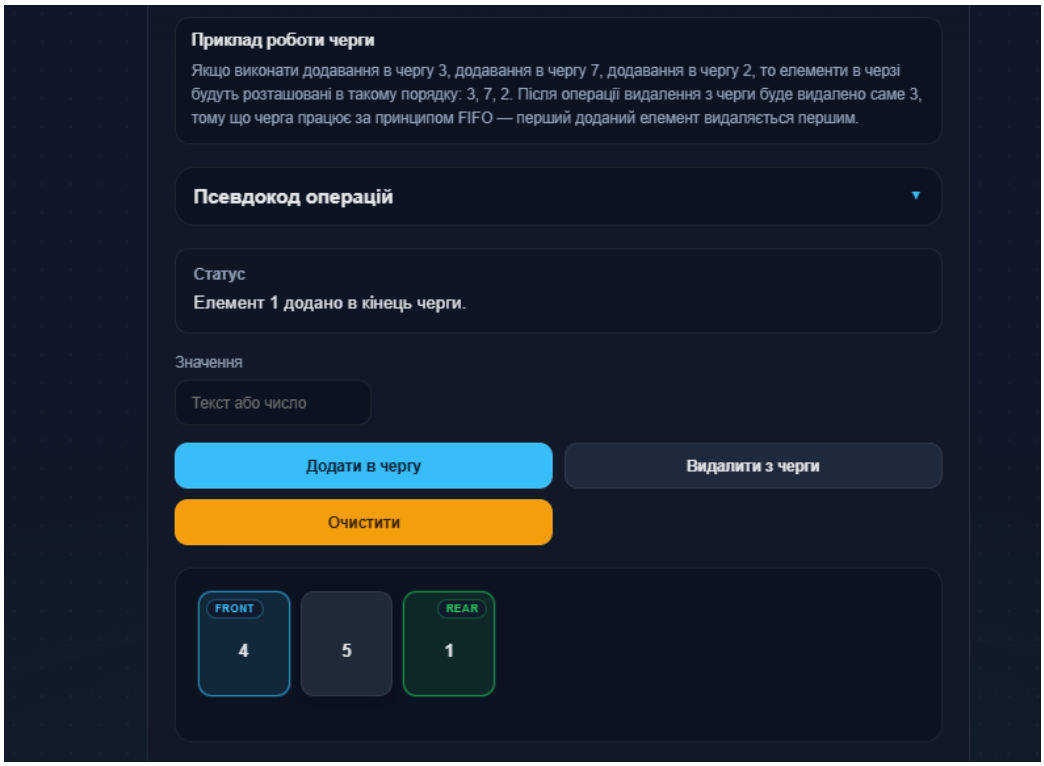

Рисунок 3.5 — Сторінка «Структури даних»: робота з чергою

У полі статусу виводиться повідомлення про останню виконану операцію. Повний код реалізації наведено у Додатку Г.

У лістингу Г.1 представлено основні операції роботи з черги: enqueue() додає елемент у кінець черги за допомогою методу push(), попередньо перевіряючи коректність введеного значення та обмеження на максимальний розмір черги. Функція dequeue() видаляє елемент з початку черги методом shift(), повідомляючи користувача про видалене значення. Якщо черга порожня, видалення не виконується і виводиться відповідне повідомлення. Така реалізація відповідає принципу роботи черги за схемою FIFO (First In, First Out) — першим обробляється елемент, який був доданий першим.

На відміну від модуля «Алгоритми», де стан скидається при кожному новому запуску, у модулі «Структури даних» стан кожної структури зберігається між операціями. Стек, черга, дек та однозв'язний список представлені звичайними JavaScript-масивами (stack, queue, deque, list), бінарне дерево — об'єктами-вузлами з полями value, left, right через змінну root, а граф — двома масивами: vertices (назви вершин) та edges (пари вершин). Після

кожної операції викликається відповідна функція відображення — renderStack(), renderQueue(), renderList(), renderDeque() або render() — яка повністю перебудовує DOM-вміст зони візуалізації на основі поточного стану структури.

На рисунку 3.6 зображено стан однозв'язного списку після кількох операцій. Кожен вузол відображається у вигляді прямокутника з числовим значенням, а стрілки між вузлами наочно показують напрямок зв'язків.

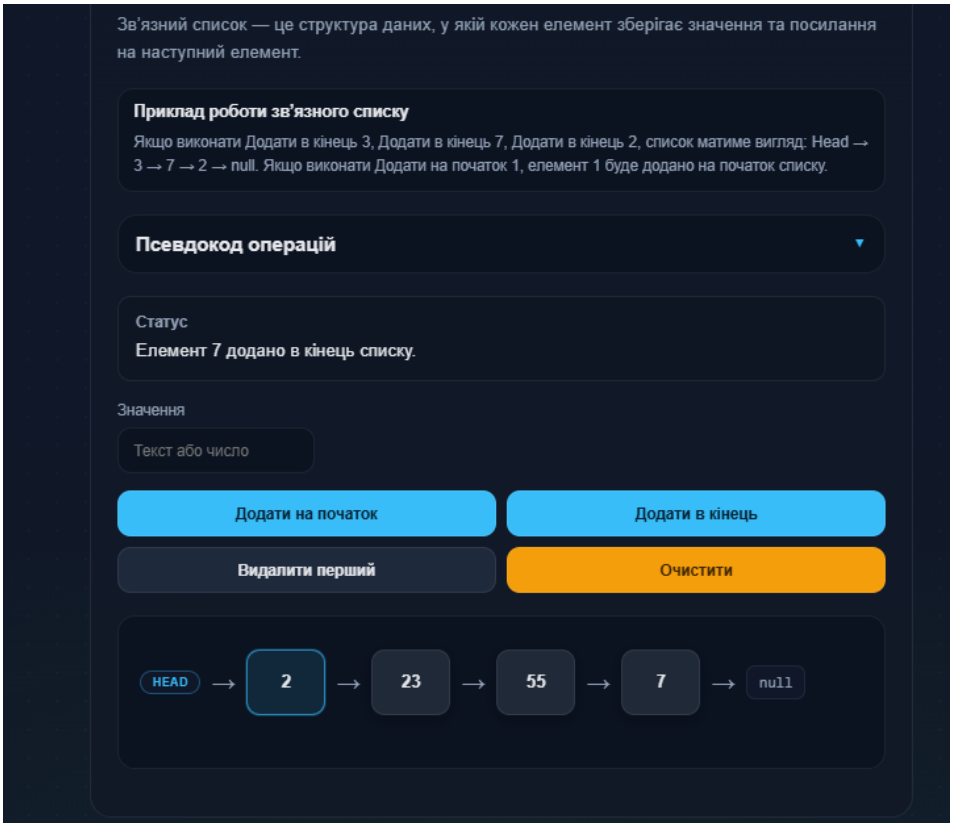

Рисунок 3.6 – Візуалізація однозв'язного списку (Linked List)

Останній вузол списку позначений маркером NULL. Повний код реалізації наведено у Додатку Д.

У лістингу Д.1 представлено основні операції зі зв'язним списком: додавання елемента на початок (addFirst), додавання елемента в кінець (addLast) та видалення елемента з початку списку (removeFirst). Функції addFirst і addLast використовують відповідно методи unshift() і push() масиву, який слугує внутрішнім представленням списку. Перед виконанням операції

викликається функція readValue(), яка перевіряє введене значення на коректність та обмеження розміру списку. Функція removeFirst перевіряє, чи список не є порожнім, і у разі успішного видалення повідомляє користувача про вилучене значення.

Функція renderList() відповідає за візуальне відображення зв'язного списку на сторінці «Структури даних». Якщо список порожній, користувачу виводиться відповідне повідомлення. В іншому випадку для першого елемента додається підпис Head, що позначає початок списку, після чого для кожного елемента створюється окремий блок зі стрілкою-зв'язком між ними. Наприкінці списку додається маркер null, що символізує відсутність наступного елемента та відповідає теоретичному визначенню зв'язного списку, у якому останній вузол не має посилання на наступний.

На рисунку 3.7 показано бінарне дерево пошуку після вставки кількох значень. Кожен вузол відображається як коло з числом, а ребра між батьківськими і дочірніми вузлами — як лінії. Корінь розташований у верхній частині, ліве піддерево містить менші значення, праве — більші. Такий вигляд точно відповідає теоретичному визначенню бінарного дерева пошуку.

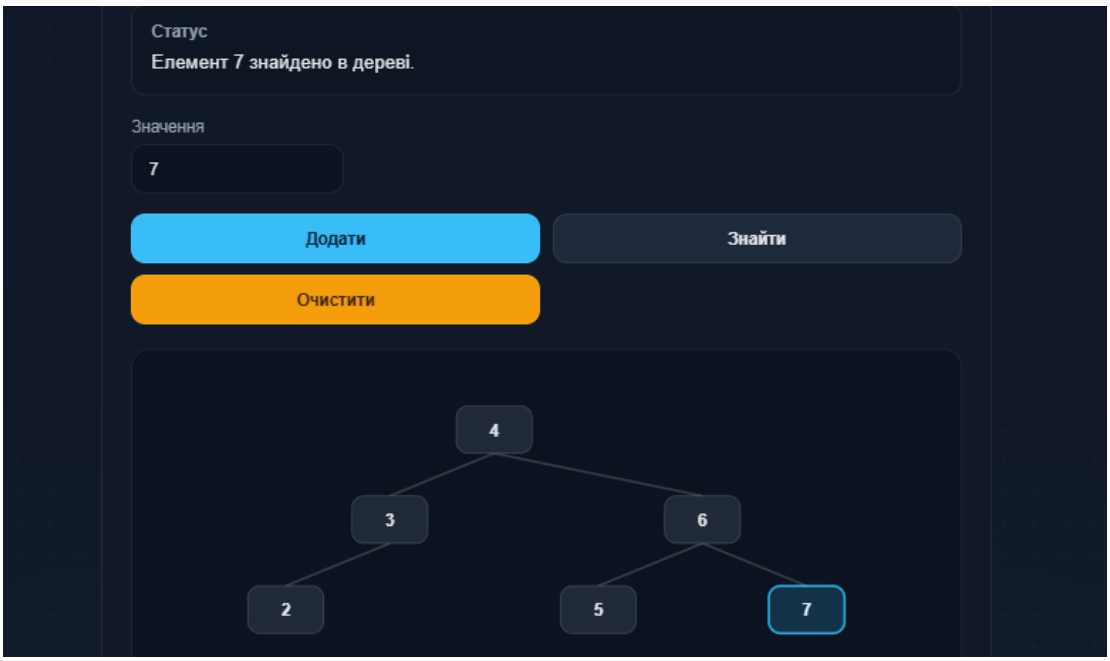

Рисунок 3.7 – Візуалізація бінарного дерева пошуку (BST)

Візуалізація динамічно перебудовується після кожної операції вставки або видалення, що дозволяє користувачеві спостерігати за зміною структури дерева в реальному часі та розуміти, як кожен новий елемент впливає на його форму. Окрім вставки, реалізовано операції пошуку вузла та видалення, кожна з яких супроводжується покроковим виділенням відповідних вузлів і текстовим поясненням виконуваної дії. Повний код реалізації наведено у Додатку Е.

У лістингу Е.1 представлено функцію add(), яка реалізує вставку нового елемента у бінарне дерево пошуку. Якщо дерево порожнє, новий елемент стає коренем. В іншому випадку функція рухається від кореня вниз: на кожному кроці значення порівнюється з поточним вузлом, і залежно від результату пошук продовжується у лівому або правому піддереві. Якщо відповідна гілка відсутня, на цьому місці створюється новий вузол. Якщо значення вже присутнє в дереві, повторне додавання не виконується, а користувачу виводиться відповідне повідомлення.

Лістинг 3.4 — Фрагмент функції пошуку елемента в бінарному дереві пошуку

```
function findNode(value) {
  let current = root;
  while (current !== null) {
    if (value === current.value) return current;
    current = value < current.value ? current.left : current.right;
  }
  return null;
}
```

Функція findNode() реалізує пошук значення в бінарному дереві пошуку. Починаючи з кореня, на кожному кроці значення порівнюється з поточним вузлом: якщо значення менше, пошук продовжується у лівому піддереві, якщо більше — у правому. Якщо вузол із шуканим значенням знайдено, функція повертає його; якщо досягнуто кінця дерева без збігу — повертається null, що означає відсутність елемента в дереві. Складність такого пошуку залежить від

висоти дерева: у середньому випадку вона становить O(log n), у найгіршому — O(n) для незбалансованого дерева.

3.6 Тестування вебсайту-тренажера

Тестування розробленого вебсайту-тренажера проводилося вручну за заздалегідь визначеними сценаріями. Метою тестування було перевірити коректність роботи всіх реалізованих алгоритмів і структур даних, стабільність інтерфейсу та коректну обробку помилкових дій користувача. Тестування охопило шість категорій перевірок: функціональне, тестування граничних випадків, обробки помилок, кросбраузерне, продуктивності та адаптивне. Результати функціонального тестування наведено в таблицях 3.2 та 3.3.

Таблиця 3.2 – Результати функціонального тестування алгоритмів

Алгоритм	Тестовий сценарій	Очікуваний результат	Фактичний результат	Статус
Bubble Sort	Масив [5,3,1,4,2]	Відсортований масив [1,2,3,4,5]	[1,2,3,4,5]	Пройдено
Selection Sort	Масив [9,7,6,1,0]	[0,1,6,7,9]	[0,1,6,7,9]	Пройдено
Insertion Sort	Масив з 1 елемента [7]	[7]	[7]	Пройдено
Merge Sort	Вже відсортований [1,2,3]	[1,2,3]	[1,2,3]	Пройдено
Quick Sort	Масив у зворотньому порядку	Відсортований масив	Коректно	Пройдено
Binary Search	Пошук 3 у [1,2,3,4,5]	Знайдено на позиції 2	Позиція 2	Пройдено
Linear Search	Пошук відсутнього елемента	Повідомлення «Не знайдено»	Виведено повідомлення	Пройдено
Jump Search	Масив [2,4,6,8,10], пошук 6	Знайдено на позиції 2	Позиція 2	Пройдено
Interpolation Search	Масив [10,20,30,40,50]	Знайдено на позиції 3	Позиція 3	Пройдено
Exponential Search	Масив [5,10,20,30,40,50]	Знайдено на позиції 4	Позиція 4	Пройдено

Таблиця 3.3 – Результати тестування структур даних

Структура	Операція	Вхідні дані	Очікуваний результат	Статус
Stack	Push → Pop	Push(5), Push(10), Pop()	Повернуто 10	Пройдено
Stack	Pop із порожнього стеку	—	Повідомлення про помилку	Пройдено
Queue	Enqueue → Dequeue	Enqueue(1), Enqueue(2), Dequeue()	Повернуто 1	Пройдено
Linked List	Додавання на початок	AddFirst(3), AddFirst(5)	Список: 5→3→NULL	Пройдено
Deque	AddFront та AddBack	AddFront(1), AddBack(2)	Дек: 1→2	Пройдено
BST	Вставка та пошук	Insert(5,3,7), Search(3)	Вузол 3 знайдено	Пройдено
Graph	DFS обхід	Граф з 4 вузлами, старт з 1	Коректна послідовність обходу	Пройдено

Таблиці 3.2 і 3.3 підтверджують, що всі реалізовані алгоритми та операції зі структурами даних працюють коректно в стандартних і граничних умовах. Жодного критичного дефекту не було виявлено. Усі перевірені сценарії завершились із очікуваними результатами.

На рисунку 3.8 зображено реакцію вебсайту на некоректні дії користувача — зокрема, спробу запустити алгоритм із порожнім полем введення. У верхній частині робочої зони відображається панель керування з порожнім полем даних, а в зоні візуалізації замість анімації з'являється читабельне повідомлення про помилку «Будь ласка, введіть дані». Аналогічна логіка обробки помилок реалізована для всіх структур даних — спроба виконати операцію на порожній структурі також супроводжується відповідним повідомленням.

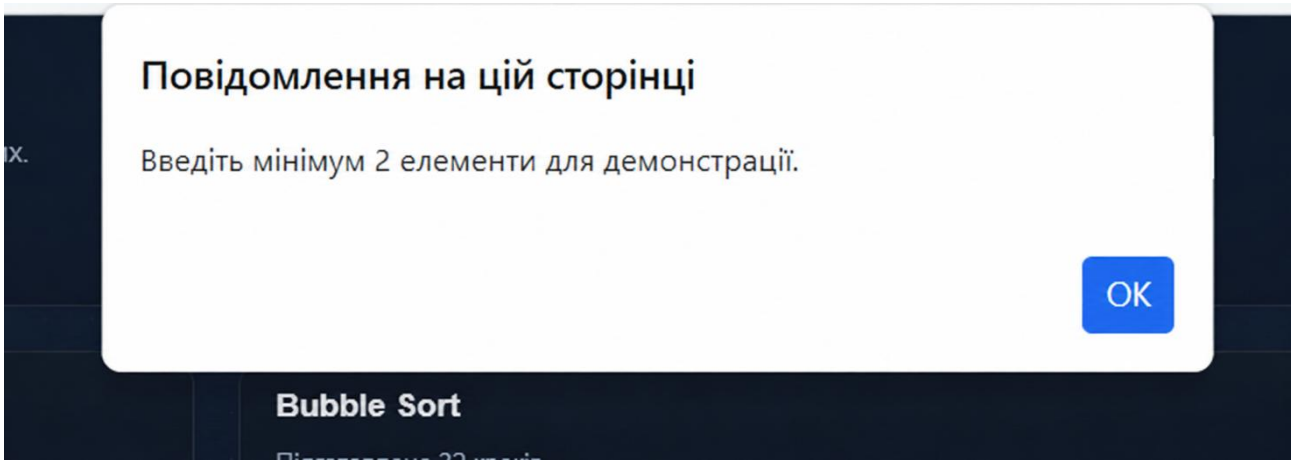

Рисунок 3.8 – Обробка помилки при порожньому полі введення

Такий підхід забезпечує стабільну роботу і не вимагає від користувача перезавантаження сторінки.

Окрему категорію тестування становить перевірка кросбраузерної сумісності вебсайту-тренажера. Метою такого тестування є підтвердження коректної роботи інтерфейсу та анімацій незалежно від використовуваного браузера, оскільки відсутність серверної частини покладає всю логіку виконання на клієнтське середовище. Перевірка здійснювалась у чотирьох браузерах: Google Chrome, Mozilla Firefox, Microsoft Edge (десктопні версії) та Safari (мобільна версія, iOS). У кожному браузері перевірялось коректне відображення сторінок «Алгоритми» та «Структури даних», запуск анімації обраного алгоритму та коректність роботи елементів керування (кнопки запуску, паузи, кроку вперед/назад). Результати кросбраузерного тестування наведено в таблиці 3.4.

Таблиця 3.4 – Результати кросбраузерного тестування

Браузер	Платформа	Відображення інтерфейсу	Робота анімації	Статус
Google Chrome	Desktop	Коректне	Коректна	Пройдено
Mozilla Firefox	Desktop	Коректне	Коректна	Пройдено
Microsoft Edge	Desktop	Коректне	Коректна	Пройдено
Safari	iOS (iPhone)	Коректне	Коректна	Пройдено

У всіх перевірених браузерах вебсайт-тренажер відображається та функціонує однаково, без візуальних або функціональних відмінностей. Це підтверджує, що використання стандартизованих веброзробницьких технологій (HTML5, CSS3, ES6+) без сторонніх бібліотек забезпечує необхідний рівень сумісності для навчального вебзастосунку.

Додатково проведено тестування продуктивності та адаптивності інтерфейсу вебсайту-тренажера. Перевірка часу відгуку інтерфейсу на дії користувача здійснювалась за допомогою вбудованого інструменту Chrome DevTools (вкладка Performance). Зафіксований показник INP (Interaction to Next Paint) склав 43 мс, що відповідає сформульованій нефункціональній вимозі щодо часу відклику не більше 100 мс (таблиця 2.3).

Результати тестування наведені в таблиці 3.5.

Таблиця 3.5 – Результати тестування продуктивності та адаптиності

Критерій тестування	Метод перевірки	Результат	Вимога (таблиця 2.3)	Статус
Час відгуку інтерфейсу (INP)	Chrome DevTools → Performance	43 мс	≤ 100 мс	Пройдено
Адаптивність на мобільному (360px)	Chrome DevTools → Emulation	Коректне масштабування	–	Пройдено
Адаптивність на планшеті (768px)	Chrome DevTools → Emulation	Коректне масштабування	–	Пройдено
Адаптивність на десктопі (1920px)	Реальний пристрій	Коректне масштабування	–	Пройдено
Перевірка на реальному пристрої	Safari, iOS (iPhone)	Елементи не виходять за межі екрана	–	Пройдено

Адаптивність інтерфейсу перевірялась шляхом відображення вебсайту на екранах різного розміру — від мобільних пристроїв до настільних моніторів, з використанням режиму емуляції пристроїв у Chrome DevTools, а також безпосередньо на мобільному пристрої (Safari, iOS). У всіх перевірених варіантах елементи інтерфейсу коректно масштабувались і залишались доступними для взаємодії, без накладання елементів чи виходу контенту за межі видимої області екрана.

3.7 Розміщення вебсайту на Netlify

Після завершення розробки та тестування вебсайт-тренажер було розміщено у мережі Інтернет за допомогою платформи Netlify. Netlify є хмарним сервісом для хостингу статичних вебсайтів, який надає безоплатний тарифний план із достатнім набором можливостей для навчального проєкту [23].

Процес розгортання відбувався у такій послідовності: файли проєкту було завантажено через веббраузер безпосередньо на платформу Netlify методом drag-and-drop або підключення до репозиторію Git [24]. Сервіс автоматично визначив, що сайт є статичним, і розгорнув його без додаткових налаштувань. Після завершення розгортання було отримано автоматично сгенерований URL [25].

Онлайн-версія вебсайту доступна за адресою: silly-croquembouche-de5eef.netlify.app

На рисунку 3.9 зображено панель керування Netlify після успішного розгортання сайту. Відображається статус деплою «Published», посилання на опублікований сайт та час останнього оновлення. Такий підхід до хостингу є повністю безоплатним і забезпечує стабільну роботу сайту з підтримкою HTTPS.

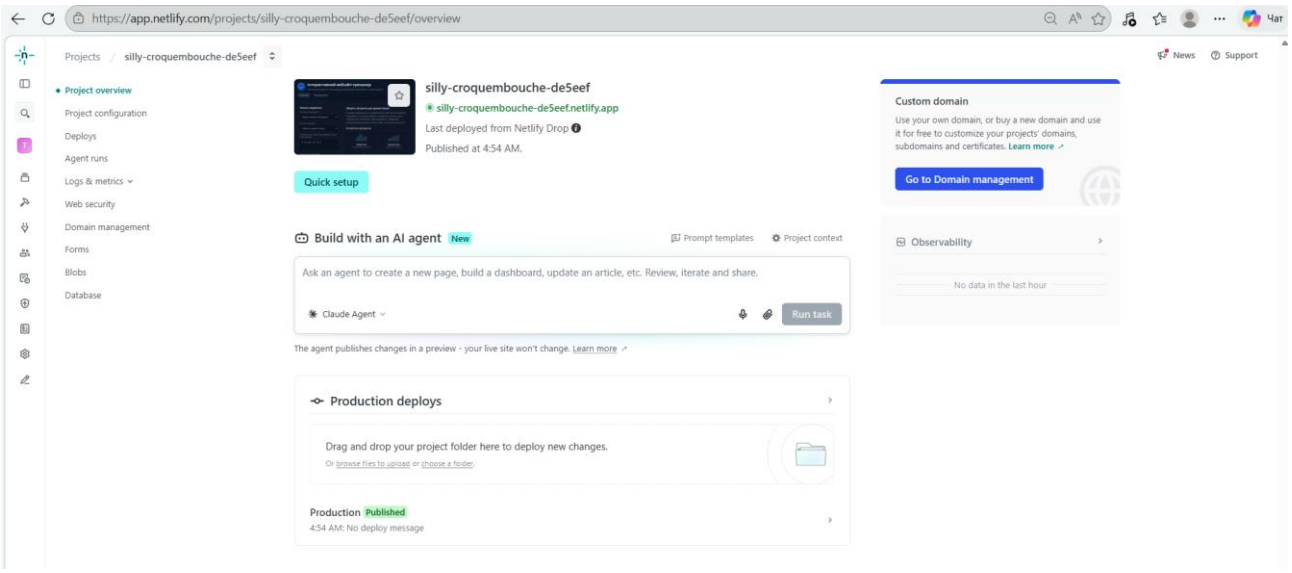

Рисунок 3.9 – Панель керування Netlify після розгортання вебсайту-тренажера

Однією з переваг використання Netlify є автоматичне налаштування HTTPS. Сайт доступний за захищеним протоколом без будь-яких додаткових налаштувань з боку розробника. Крім того, CDN-мережа Netlify забезпечує швидке завантаження ресурсів незалежно від географічного розташування користувача.

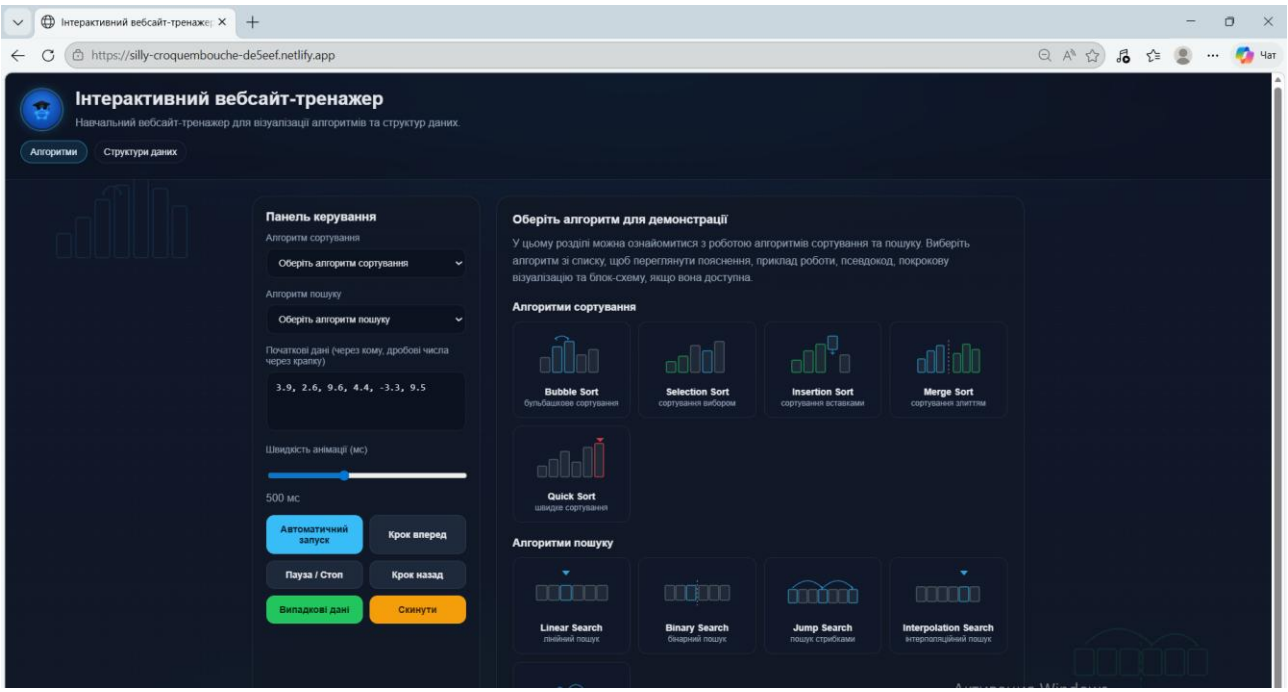

Рисунок 3.10 – Онлайн-версія вебсайту-тренажера за адресою silly-croquembouche-de5eef.netlify.app

На рисунку 3.10 показано опублікований вебсайт у браузері. Сайт доступний із будь-якого пристрою без необхідності встановлення додаткового програмного забезпечення. Онлайн-розміщення підтверджує, що розроблений продукт готовий до використання у навчальному процесі.

3.8 Загальний огляд функціоналу готового продукту

Розроблений вебсайт-тренажер у фінальному вигляді являє собою повноцінний навчальний інтерактивний ресурс, який охоплює 10 алгоритмів і 6 структур даних. Всі заплановані під час проєктування функції реалізовано та протестовано. Нижче наведено підсумковий перелік реалізованих можливостей.

Таблиця 3.6 – Реалізований функціонал готового продукту

Функціональність	Алгоритми	Структури даних
Вибір з випадаючого списку	✓	✓
Введення власних даних	✓	✓
Генерація випадкових даних	✓	—
Покрокова візуалізація	✓	✓
Пояснення поточного кроку	✓	✓
Кнопка «Старт» / «Пауза» / «Стоп»	✓	—
Крок вперед / крок назад	✓	—
Скидання до початку	✓	✓
Регулювання швидкості	✓	—
Псевдокод	✓	✓
Блок-схема	✓ (для частини)	—
Обробка помилок введення	✓	✓
Статус виконання операції	—	✓

У таблиці 3.6 підсумовано весь реалізований функціонал. Відмінності між двома модулями є обґрунтованими: алгоритми потребують більш розгорнутого керування анімацією, тоді як структури даних — підтвердження кожної окремої операції.

У третьому розділі описано практичну реалізацію вебсайту-тренажера. Проєкт організовано у вигляді набору статичних файлів — index.html та data-structures.html для двох основних сторінок, style.css для оформлення, а також script.js і structures.js, які відповідають за логіку алгоритмів та структур даних відповідно. Така структура дозволила розділити код за функціональним призначенням і полегшити подальшу підтримку.

Реалізовано навігаційну панель, спільну для обох сторінок, та загальний макет з використанням семантичних HTML-тегів. На сторінці «Алгоритми» представлено десять алгоритмів — п'ять алгоритмів сортування (Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, Quick Sort) та п'ять алгоритмів пошуку (Linear Search, Binary Search, Jump Search, Interpolation Search, Exponential Search). Для кожного з них реалізовано покрокову візуалізацію у вигляді анімованих стовпчиків з кольоровим маркуванням (порівнювані, переставлювані, відсортовані елементи), пояснення поточного кроку, псевдокод та повний набір елементів керування — старт, пауза, крок вперед/назад, скидання, регулювання швидкості та генерація випадкових даних. Для частини алгоритмів додатково підготовлено блок-схеми.

На сторінці «Структури даних» реалізовано роботу з шістьма структурами: Stack, Queue, Linked List, Deque, Binary Search Tree та Graph. На відміну від алгоритмів, тут взаємодія відбувається в реальному часі — користувач самостійно виконує операції (push/pop, enqueue/dequeue, вставка, видалення, пошук тощо), а зона візуалізації одразу оновлюється з анімацією та повідомленням про статус операції.

Проведено ручне функціональне тестування всіх алгоритмів і структур даних, включно з граничними випадками — порожній масив, масив з одного елемента, вже відсортовані дані та дані у зворотному порядку. Окремо перевірено обробку помилкових дій користувача, зокрема спроби запустити демонстрацію з порожнім полем введення чи виконати операцію на порожній структурі даних. Усі перевірені сценарії пройшли успішно, критичних дефектів не виявлено.

Після завершення тестування сайт розміщено на платформі Netlify методом drag-and-drop, що дозволило отримати робоче посилання без додаткових налаштувань серверної частини. Готовий продукт реалізує весь запланований функціонал — 10 алгоритмів і 6 структур даних з покроковою візуалізацією, поясненнями, псевдокодом та засобами керування — і доступний для використання онлайн.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи розглянуто роль алгоритмів у підготовці фахівців з програмування та проаналізовано особливості засвоєння цих тем у навчальному процесі. Алгоритмічне мислення і вміння працювати зі структурами даних є базовими компетентностями для майбутнього розробника, а традиційні текстові способи подання матеріалу не завжди дають достатню наочність процесів, які відбуваються на кожному кроці виконання алгоритму.

Проаналізовано роль візуалізації як засобу покращення розуміння алгоритмічних процесів. Покрокове графічне відображення роботи алгоритму дозволяє простежити логіку кожної операції, порівняти поведінку різних алгоритмів на однакових даних і знизити поріг входження для користувачів з різним рівнем підготовки.

Проведено огляд трьох існуючих вебресурсів-аналогів — Sort Visualizer, Algorithm Visualizer та VisuAlgo. Кожен з них має власні обмеження: Sort Visualizer орієнтований лише на сортування, Algorithm Visualizer вимагає базових знань програмування для роботи з кодом, а VisuAlgo, попри широке охоплення тем, складний для новачка через перевантажений інтерфейс. Це стало основою для розробки власного продукту, який поєднує простоту, інтерактивність та достатнє функціональне наповнення в одному середовищі.

На основі проведеного аналізу спроєктовано архітектуру вебсайту-тренажера як клієнтського застосунку без серверної частини. Як технологічний стек обрано HTML5, CSS3 та JavaScript без зовнішніх бібліотек — це забезпечило незалежність від сторонніх залежностей і стабільну роботу у будь-якому сучасному браузері. Розроблено структуру сторінок, інтерфейс користувача та визначено склад двох основних модулів — «Алгоритми» і «Структури даних».

Реалізований вебсайт-тренажер містить дві основні сторінки. На сторінці «Алгоритми» представлено десять алгоритмів: п'ять алгоритмів сортування (Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, Quick Sort) та п'ять

алгоритмів пошуку (Linear Search, Binary Search, Jump Search, Interpolation Search, Exponential Search). Для кожного реалізовано покрокову візуалізацію у вигляді анімованих стовпчиків, пояснення поточного кроку, відображення псевдокоду, приклад роботи та блок-схеми для окремих алгоритмів. Передбачено повний набір засобів керування демонстрацією: запуск, пауза, зупинка, крок вперед і назад, скидання, регулювання швидкості та генерація випадкових даних.

На сторінці «Структури даних» реалізовано роботу з шістьма структурами: Stack, Queue, Linked List, Deque, Binary Search Tree та Graph. Для кожної структури доступні операції додавання, видалення та пошуку з відображенням поточного стану у вигляді графічної анімації, поясненням принципу роботи та псевдокодом операцій. Реалізовано перевірку введення та обробку помилкових дій користувача.

Проведено ручне функціональне тестування всіх реалізованих алгоритмів і структур даних, включно з перевіркою граничних випадків та коректності обробки некоректних вхідних даних. Усі протестовані сценарії завершились із очікуваними результатами. Сайт також перевірено у кількох сучасних браузерах, і його робота скрізь визнана коректною.

Після завершення розробки вебсайт-тренажер розміщено у відкритому доступі за допомогою платформи Netlify. Сайт доступний онлайн за адресою silly-croquembouche-de5eef.netlify.app і працює за захищеним протоколом HTTPS без необхідності встановлення додаткового програмного забезпечення.

Мету кваліфікаційної роботи досягнуто, оскільки розроблено інтерактивний вебсайт-тренажер, який забезпечує покрокову візуалізацію алгоритмів і структур даних у зручному вебсередовищі. Розроблений продукт може використовуватися як навчальний інструмент для студентів, що вивчають основи алгоритмізації та програмування, а також як самостійний освітній ресурс для будь-кого, хто хоче краще розібратися в роботі базових алгоритмів і структур даних.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. OpenDSA Data Structures and Algorithms Modules. URL: https://opendsa-server.cs.vt.edu/OpenDSA/Books/Everything/html/index.html

2. Miller B., Ranum D. Problem Solving with Algorithms and Data Structures using Python. Runestone Academy. URL: https://runestone.academy/ns/books/published/pythonds/index.html

3. Learn Data Structures and Algorithms. Programiz. URL: https://www.programiz.com/dsa

4. DSA Tutorial. GeeksforGeeks. URL: https://www.geeksforgeeks.org/dsa/dsa-tutorial-learn-data-structures-and-algorithms/

5. Data Structure and Types. Programiz. URL: https://www.programiz.com/dsa/data-structure-types

6. Sorting Algorithms. GeeksforGeeks. URL: https://www.geeksforgeeks.org/dsa/sorting-algorithms/

7. Sorting Algorithms. VisuAlgo. URL: https://visualgo.net/en/sorting

8. Binary Search Tree. VisuAlgo. URL: https://visualgo.net/en/bst

9. Linked List Data Structure. Programiz. URL: https://www.programiz.com/dsa/linked-list

10. HTML: HyperText Markup Language. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/HTML

11. CSS: Cascading Style Sheets. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/CSS

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

13. Document Object Model (DOM). MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model

14. Learn HTML. web.dev. URL: https://web.dev/learn/html

15. Learn CSS. web.dev. URL: https://web.dev/learn/css

16. Сучасний підручник з JavaScript. URL: https://uk.javascript.info/

17. HTML Tutorial. W3Schools. URL: https://www.w3schools.com/html/

18. CSS Tutorial. W3Schools. URL: https://www.w3schools.com/css/

19. JavaScript Tutorial. W3Schools. URL: https://www.w3schools.com/js/

20. Sort Visualizer. URL: https://sortvisualizer.com/

21. Algorithm Visualizer. URL: https://algorithm-visualizer.org/

22. VisuAlgo. URL: https://visualgo.net/

23. Netlify Documentation. Netlify Docs. URL: https://docs.netlify.com/

24. Get Started with Netlify Drop. Netlify Docs. URL: https://docs.netlify.com/start/get-started-with-drop/

25. Create deploys. Netlify Docs. URL: https://docs.netlify.com/deploy/create-deploys/

26. Hundhausen C. D., Douglas S. A., Stasko J. T. A Meta-Study of Algorithm Visualization Effectiveness. 2002. URL: https://doi.org/10.1006/jvlc.2002.0237

27. Naps T. L., Rößling G., Almstrum V. та ін. Exploring the Role of Visualization and Engagement in Computer Science Education. 2003. URL: https://doi.org/10.1145/782941.782998

28. Shaffer C. A., Cooper M. L., Alon A. J. D. та ін. Algorithm Visualization: The State of the Field. 2010. URL: https://doi.org/10.1145/1821996.1821997

29. Shaffer C. A., Karavirta V., Korhonen A., Naps T. L. OpenDSA: Beginning a Community Active-eBook Project. 2011. URL: https://doi.org/10.1145/2094131.2094154

Додаток А

Лістинг А.1 – Фрагмент функції генерації кроків алгоритму Bubble Sort

```
function bubbleSteps(arr) {
  const a = [...arr];
  const out = [];

  for (let i = 0; i < a.length; i++) {
    for (let j = 0; j < a.length - i - 1; j++) {
      out.push({
        data: [...a],
        active: [j, j + 1],
        text: `Порівнюються елементи ${a[j]} і ${a[j + 1]}.`,
        done: Array.from({ length: i }, (_, k) => a.length - 1 - k)
      });

      if (a[j] > a[j + 1]) {
        [a[j], a[j + 1]] = [a[j + 1], a[j]];

        out.push({
          data: [...a],
          active: [j, j + 1],
          text: `Елементи поміняно місцями.`,
          done: Array.from({ length: i }, (_, k) => a.length - 1 - k)
        });
      }
    }


 out.push({
      data: [...a],
      done: Array.from({ length: i + 1 }, (_, k) => a.length - 1 - k),
      text: `Черговий прохід завершено. Найбільший елемент уже на правильному місці.`
    });
  }

  return out;
}
```

Додаток Б

Лістинг Б.1 — Фрагмент функції генерації кроків алгоритму Binary Search

```
function binarySteps(arr, target) {
  const a = [...arr].sort((x, y) => x - y);
  const out = [];
  let left = 0, right = a.length - 1;

  while (left <= right) {
    const mid = Math.floor((left + right) / 2);
    out.push({ data: [...a], active: [mid],
      text: `Перевірка елемента ${a[mid]}` });

    if (a[mid] === target) {
      out.push({ data: [...a], active: [mid], found: mid,
        text: `Знайдено на позиції ${mid + 1}` });
      return out;
    }
    target < a[mid] ? right = mid - 1 : left = mid + 1;
  }

  out.push({ data: [...a], text: 'Не знайдено' });
  return out;
}
```

Додаток В

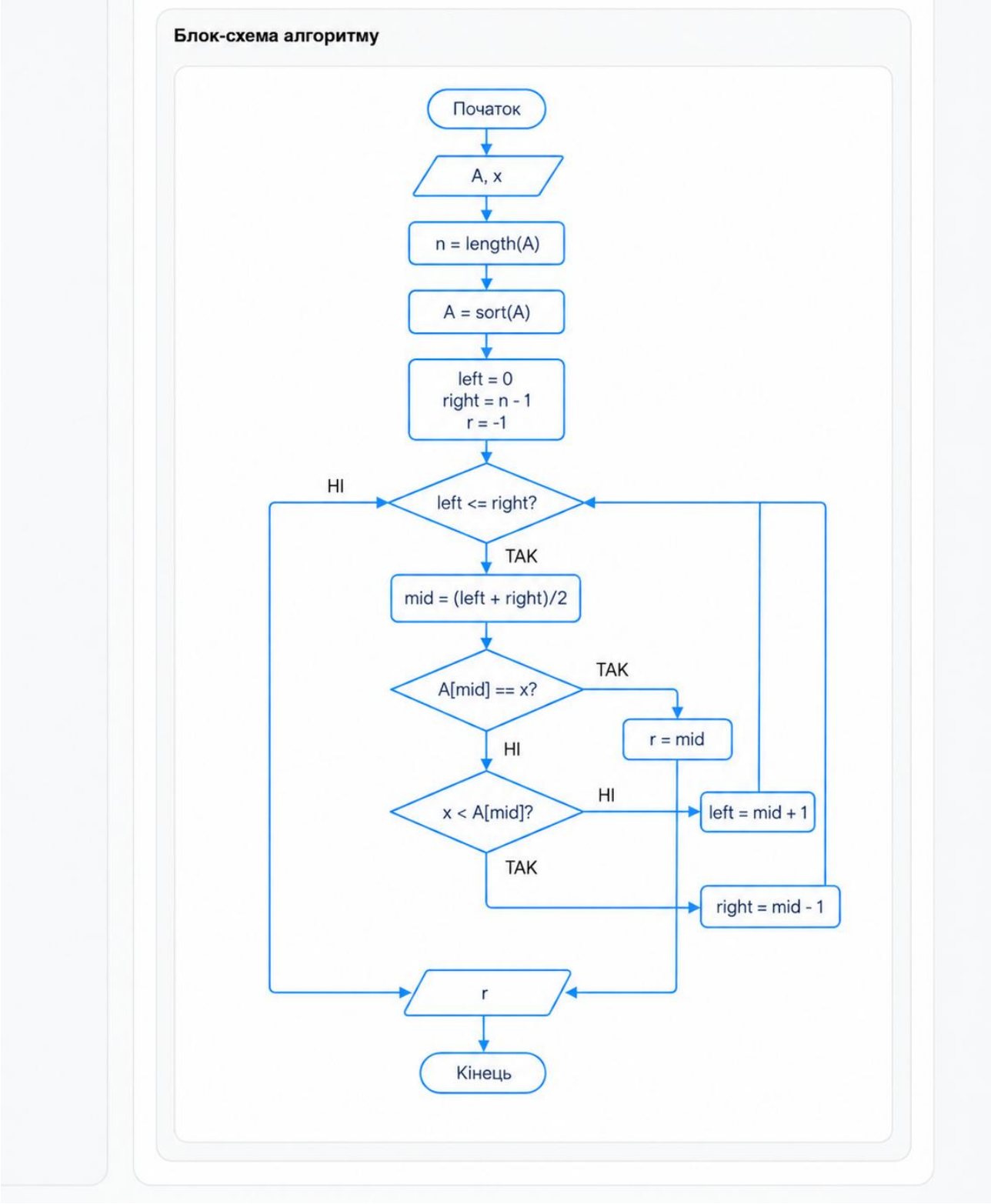

Рисунок В.1 — Блок-схема алгоритму Binary Search

Додаток Г

Лістинг Г.1 — Фрагмент функцій додавання елементів та видалення елементів черги

```
function enqueue() {
  const value = queueInput.value.trim();
  if (!value) return setStatus("Введіть значення.");
  if (queue.length >= MAX_QUEUE_SIZE) return setStatus("Черга заповнена.");

  queue.push(value);
  queueInput.value = "";
  renderQueue();
  setStatus(`Елемент ${value} додано в кінець черги.`);
}

function dequeue() {
  if (queue.length === 0) return setStatus("Черга порожня.");
  const value = queue.shift();
  renderQueue();
  setStatus(`Елемент ${value} видалено з початку черги.`);
}
```

Додаток Д

Лістинг д.1 – Фрагмент функції відображення зв’язного списку

```
function renderList() {
  llVisual.innerHTML = "";

  if (list.length === 0) {
    llVisual.innerHTML = '<p class="linked-list-empty muted">Список порожній</p>';
    return;
  }

  llVisual.innerHTML += '<span class="linked-list-head-label">Head</span>';

  list.forEach((value, i) => {
    llVisual.innerHTML += '<span class="linked-list-arrow">→</span>';
    const node = document.createElement("div");
    node.className = "linked-list-node" + (i === 0 ? " linked-list-node-head" : "");
    node.innerHTML = `<span class="linked-list-node-value">${value}</span>`;
    llVisual.appendChild(node);
  });

  llVisual.innerHTML += '<span class="linked-list-null">null</span>';
}
```

Додаток Е

Лістинг **Е.1** — Фрагмент функції вставки вузла в бінарне дерево пошуку (BST)

```
function add() {
  const value = readNumber();
  if (value === null) return;

  if (root === null) {
    root = { value, left: null, right: null };
    nodeCount++;
    render();
    return setStatus(`Елемент ${value} додано як корінь дерева.`);
  }

  let current = root;
  while (true) {
    if (value === current.value) return setStatus("Таке значення вже є.");
    const side = value < current.value ? "left" : "right";
    if (current[side] === null) {
      current[side] = { value, left: null, right: null };
      nodeCount++;
      render();
      return setStatus(`Елемент ${value} додано ${side === "left" ? "ліворуч" : "праворуч"}.`);
    }
    current = current[side];
  }
}
```