Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет _____Комп'ютерних наук_____
(повна назва)

Кафедра __Комп'ютерного моделювання та інтелектуальних технологій_____
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти _____другий (магістерський)_____
_Дослідження та порівняльний аналіз інструментальних засобів проєктування та розробки інформаційних систем_____
(тема)

Виконав:
здобувач ___2_____ року навчання,
групи ____СПРм-24-1_____
_____Максим Медяник_____
(власне ім'я, прізвище)

Спеціальність _122 Комп'ютерні науки_____
(код і повна назва спеціальності)

Тип програми _освітньо-наукова_____
(освітньо-професійна або освітньо-наукова)

Освітня програма __Системне проєктування_
(повна назва освітньої програми)

Керівник _проф. Надія Калита_____
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри КМІТ _____ ___Ігор Гребеннік_____
(підпис) (власне ім'я, прізвище)

2026 р.

Я, як студент ХНУРЕ, розумію і підтримую політику закладу із академічної доброчесності. Я не надавав і не одержував недозволену допомогу під час підготовки кваліфікаційної роботи. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

16.05.2026

Кваліфікаційна робота не містить відомостей заборонених до відкритого опублікування.

Керівник кваліфікаційної роботи _____ проф. Калита Н.І

Кваліфікаційна робота виконана у відповідності до стандартів, що діють в Україні.

Керівник кваліфікаційної роботи _____ проф. Калита Н.І.

Попередній захист проведено 16 травня 2026 р.

Керівник кваліфікаційної роботи _____ проф. Калита Н.І.

Харківський національний університет радіоелектроніки

Факультет_____Комп'ютерних наук_____

Кафедра___Комп'ютерного моделювання та інтелектуальних систем_____

Рівень вищої освіти_____другий (магістерський)_____

Спеціальність___122 Комп'ютерні науки_____
(код і повна назва)

Тип програми _____освітньо-наукова_____
(освітньо-професійна або освітньо-наукова)

Освітня програма_____Системне проєктування_____
(повна назва освітньої програми)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____
(підпис)
«18» травня 2026 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві_____Медянику Максиму Юрійовичу_____
(прізвище, власне ім'я, по батькові)

1. Тема роботи «Дослідження та порівняльний аналіз інструментальних засобів проєктування та розробки інформаційних систем»
затверджена наказом університету від « 06 » квітня 2026р. № 268 Cm
2. Термін подання студентом роботи до екзаменаційної комісії « 18 » травня 2026 р.
3. Вихідні дані до роботи архітектурні підходи, методи та інструментальні засоби стилізації візуальних користувацьких інтерфейсів інформаційних систем електронної комерції; вхідні дані: архітектурна структура, функціональні специфікації та дизайн-макети еталонних сторінок інтернет-магазину (сторінка товару, каталог товарів із системою фільтрації, сторінка оформлення замовлення Checkout) ; стандарти та технічні вимоги Core Web Vitals; система об'єктивних інструментальних метрик продуктивності та мережевого навантаження підсистеми стилізації, ПК 16 ГБ ОП, ОС Windows 11, середовище виконання Node.js та менеджер пакетів NPM; автоматизований інструмент збірки (бандлер) Vite; мова розмітки HTML5; компонентний фреймворк Bootstrap та utility-first фреймворк Tailwind CSS; діагностичний інструментарій Chrome DevTools (панель Network) та система автоматизованого аудиту Google Lighthouse; інтеграція з інженерними онлайн-платформами розробки StackBlitz та Tailwind Play.

4. Перелік питань, що потрібно опрацювати в роботі 1. Вступ; 2. Аналіз предметної області; 3 Методика оцінювання інструментальних засобів стилізації; 4 Постановка задачі дослідження;; 5.Розробка інформаційної системи; результатів; 6 Порівняльний аналіз моделей; 6. Висновки.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) Схема класифікації сучасних CSS-фреймворків за архітектурним підходом; схема взаємодії компонентів за методологією BEM; схема архітектури сітки компонентного фреймворку Bootstrap; схема архітектурної структури еталонного тестового інтерфейсу; UML-діаграма діяльності алгоритмів кастомізації та впровадження стилів; тестовий інтерфейс програмної реалізації прототипів вебсторінок інтернет-магазину (сторінка товару, каталог, оформлення замовлення); інтерфейс інструментального вимірювання показників ефективності, мережевого навантаження та швидкодії вебсторінок; схема математичної моделі багатокритеріального оцінювання альтернатив за методом узагальненого адитивного критерію.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Аналіз предметної області	10.04.2026 –17.04.2026	Виконано
2	Дослідження критеріїв оцінювання та теоретичних основ багатокритеріального вибору	18.04.2026 – 21.04.2026	Виконано
3	Проєктування та програмна реалізація еталонних прототипів сторінок	22.04.2026 – 27.04.2026	Виконано
4	Проведення інструментальних вимірювань метрик продуктивності та мережевого навантаження.	28.04.2026 – 04.05.2026	Виконано
5	Багатокритеріальна оцінка результатів експерименту та формування висновків.	05.05.2026 – 10.05.2026	Виконано
6	Оформлення пояснювальної записки до кваліфікаційної роботи та презентації	11.05.2026 – 18.05.2026	Виконано
7	Попередній захист	19.05.2026	Виконано
8	Подання кваліфікаційної роботи до екзаменаційної комісії	20.05.2026	Виконано

Дата видачі завдання « 06 » квітня 2026 р.

Здобувач _____
(підпис)

Керівник роботи _____ проф. Калита Н.І
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Кваліфікаційна робота: 82 с., 14 рис., 3 табл., 2 додатки, 20 джерел інформації.

CSS-АРХІТЕКТУРА, BOOTSTRAP, TAILWIND CSS, КОМПОНЕНТНИЙ ПІДХІД, UTILITY-FIRST CSS, ЕЛЕКТРОННА КОМЕРЦІЯ, UI/UX ДИЗАЙН, ОПТИМІЗАЦІЯ ПРОДУКТИВНОСТІ.

Об'єктом дослідження є процес проєктування та розробки візуальних інтерфейсів сучасних інформаційних систем електронної комерції.

Предметом дослідження є інструментальні засоби, архітектурні підходи та методи впровадження стилів і дизайну на прикладі розробки сторінки товару інтернет-магазину з використанням технологій Bootstrap та Tailwind CSS.

Мета роботи – дослідити та здійснити порівняльний аналіз компонентного та utility-first підходів до стилізації інформаційних систем для визначення їх архітектурної ефективності, гнучкості, впливу на продуктивність розробки та швидкодію кінцевого продукту.

Методи дослідження – системний аналіз, технології та інструментальні засоби створення інтерфейсів, прототипування, методи оцінки метрик продуктивності програмного продукту.

Результатом роботи є комплексне порівняльне дослідження та розроблений прототип сторінки товару, які практично довели, що utility-first підхід (Tailwind CSS) дозволяє зменшити розмір фінального файлу стилів більш ніж у дванадцять разів порівняно з компонентним підходом (Bootstrap) та повністю усуває проблему надлишкового CSS-коду (CSS bloat).

Галузь застосування – розробка візуальних користувацьких інтерфейсів для масштабованих платформ, зокрема сучасних високонавантажених платформ електронної комерції, де критично важливими є висока швидкодія, малий розмір кінцевих файлів та точна відповідність унікальному фірмовому стилю

ABSTRACT

Master’s Thesis: 82 pages, 14 figures, 3 tables, 2 appendices, 20 titles.

CSS ARCHITECTURE, BOOTSTRAP, TAILWIND CSS, COMPONENT-BASED APPROACH, UTILITY-FIRST CSS, E-COMMERCE, UI/UX DESIGN, PERFORMANCE OPTIMIZATION.

The object of research is the process of designing and developing visual interfaces for modern e-commerce information systems.

The subject of research encompasses the tools, architectural approaches, and methods of styling and design implementation, exemplified by the development of an online store product page using Bootstrap and Tailwind CSS technologies.

The purpose of the work is to investigate and conduct a comparative analysis of the component-based and utility-first approaches to styling information systems to determine their architectural efficiency, flexibility, impact on development productivity, and the performance of the final product.

Research methods include system analysis, technologies and tools for interface creation, prototyping, and methods for evaluating software performance metrics.

The result of the work is a comprehensive comparative study and a developed product page prototype, which practically proved that the utility-first approach (Tailwind CSS) allows reducing the final style file size by more than twelve times compared to the component-based approach (Bootstrap) and completely eliminates the problem of CSS bloat.

Field of application – the development of visual user interfaces for scalable platforms, in particular, modern high-load e-commerce platforms, where high performance, small end-file sizes, and precise adherence to a unique corporate style are critically important.

ЗМІСТ

ВСТУП

Проєктування сучасних інформаційних систем неможливе без глибокого розуміння принципів взаємодії людини та комп'ютера (Human-Computer Interaction). З точки зору системного проєктування, користувацький інтерфейс (UI) виступає не просто візуальною оболонкою, а критично важливою підсистемою, що забезпечує двосторонню комунікацію між користувачем та апаратно-програмним комплексом. Від якості цієї підсистеми безпосередньо залежить загальний користувацький досвід (User Experience, UX). Згідно з міжнародним стандартом ISO 9241-210 (Ергономіка взаємодії людина-система) [1], орієнтований на користувача підхід є фундаментальною вимогою до розробки інтерактивних систем, що вимагає проєктування інтуїтивно зрозумілих, доступних та адаптивних інтерфейсів .

Сучасний етап розвитку інформаційних технологій та електронної комерції характеризується експоненційним зростанням вимог до якості, швидкодії та масштабованості користувацьких інтерфейсів (UI) [2]. Вебплатформи еволюціонували від простих статичних сторінок до складних односторінкових додатків та прогресивних вебзастосунків які за своєю функціональністю не поступаються десктопним програмам. У цьому контексті візуальне подання інформації та зручність взаємодії користувача з системою (User Experience – UX) стають критичними бізнес-метриками. Пошукові системи впроваджують суворі критерії оцінки продуктивності, такі як Core Web Vitals [3], що змушує інженерів шукати нові шляхи оптимізації не лише програмної логіки, але й рівня представлення даних — каскадних таблиць стилів (Cascading Style Sheets – CSS).

Історично склалося так, що традиційні підходи до написання CSS створюють значні архітектурні виклики при масштабуванні інформаційних систем. Каскадна природа стилів та їх глобальна область видимості призводять до того, що зміна дизайну одного елемента може викликати непередбачувані побічні ефекти в інших частинах додатку. Для вирішення проблеми конфліктів імен та структурування кодової бази індустрія розробила низку методологій,

таких як OOCSS, SMACSS та BEM (Block, Element, Modifier) [4]. Хоча ці підходи частково вирішили проблему інкапсуляції стилів, вони покладаються на сувору дисципліну розробників, призводять до створення довгих, важко читабельних класів та не вирішують проблему розростання загального обсягу CSS-файлів (так званий CSS bloat) у довгостроковій перспективі.

Відповіддю на ці виклики стала поява CSS-фреймворків. Тривалий час індустріальним стандартом залишалася компонентно-орієнтована архітектура стилізації, найяскравішим представником якої є фреймворк Bootstrap [5]. Цей підхід пропонує розробникам набір готових компонентів, що дозволяє екстремально пришвидшити прототипування та забезпечити візуальну консистентність інтерфейсу. Однак, незважаючи на очевидні переваги, компонентні фреймворки мають суттєві архітектурні недоліки. Вони нав'язують власний візуальний стиль, відхилення від якого вимагає написання великої кількості перевизначаючих стилів. Крім того, підключення монолітного фреймворку призводить до завантаження великої кількості невикористаного коду, що негативно впливає на метрики продуктивності.

У пошуках альтернативи, яка б поєднувала швидкість розробки та повний контроль над дизайном без надлишкового коду, сформувалася нова парадигма — utility-first CSS, лідером якої є фреймворк Tailwind CSS [6].

Актуальність теми дослідження зумовлена тим, що вибір між компонентною та utility-first архітектурою стилізації є одним із найважливіших стратегічних рішень на етапі проєктування сучасної інформаційної системи. Це рішення безпосередньо впливає на швидкість розробки, вартість підтримки кодової бази, гнучкість кастомізації та кінцеву продуктивність вебплатформи. Особливо гостро це питання постає при розробці систем електронної комерції, зокрема сторінок товарів, де необхідно поєднати складну адаптивну сітку, унікальний брендинг та високу швидкість завантаження для забезпечення максимальної конверсії.

Об'єктом дослідження є процес проєктування та розробки візуальних інтерфейсів сучасних інформаційних систем електронної комерції.

Предметом дослідження є інструментальні засоби, архітектурні підходи та методи впровадження стилів і дизайну на прикладі розробки сторінки товару інтернет-магазину з використанням технологій Bootstrap та Tailwind CSS.

Мета роботи – дослідити та здійснити порівняльний аналіз компонентного та utility-first підходів до стилізації інформаційних систем для визначення їх архітектурної ефективності, гнучкості, впливу на продуктивність розробки та швидкодію кінцевого продукту.

.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Роль користувацького інтерфейсу та UX у системному проєктуванні

У контексті системного аналізу будь-який вебзастосунок розглядається як складна соціотехнічна система, де користувацький інтерфейс (UI) виконує функцію шлюзу для передачі даних між людиною та бізнес-логікою програми. Досвід взаємодії (UX) визначає те, наскільки ефективно, результативно та із задоволенням користувач може досягати своїх цілей у цій системі. Архітектурно користувацький інтерфейс складається з трьох базових компонентів:

а) інформаційного, що відповідає за семантичне подання даних (типографіка, медіа-контент);

б) навігаційного, який забезпечує просторову орієнтацію користувача в системі за допомогою меню та засобів навігації;

в) інтерактивного, що містить елементи управління для ініціації змін стану системи.

До сучасних користувацьких інтерфейсів систем електронної комерції висувається низка жорстких вимог. Згідно з принципами евристичної оцінки та положеннями стандарту ISO 9241-210 [1, 8], інтерфейс має бути:

- консистентним;
- толерантним до помилок користувача;
- забезпечувати високий рівень доступності та адаптивності до різних типів пристроїв.

вимагає застосування чітко визначеного інструментарію проєктування, продуктивність якого оцінюється через об'єктивні кількісні метрики швидкодії та мережевого навантаження, тому на сучасному етапі проєктування інформаційних систем вибір технологічного стеку для реалізації UI/UX не обмежується лише традиційним написанням каскадних таблиць стилів.

1.2 Сучасні підходи до розробки користувацьких інтерфейсів

Розвиток інструментальних засобів для фронтенд-розробки призвів до появи широкого спектра рішень, які намагаються оптимізувати процес створення користувацьких інтерфейсів. На сучасному етапі проєктування інформаційних систем вибір технологічного стеку не обмежується лише традиційними підходами [7] або виключно фреймворками Bootstrap чи Tailwind. Існує низка альтернативних платформ та екосистем, які вирішують схожі архітектурні завдання і пропонують власні концепції взаємодії з каскадними таблицями стилів.

1.2.1 Сучасні CSS-фреймворки та їх порівняльний аналіз

Аналізуючи ринок сучасних CSS-фреймворків, окрім домінуючих інструментів, варто виділити такі рішення, як Foundation [18] та Bulma [19], що пропонують альтернативні підходи до компонентної архітектури. Фреймворк Foundation історично є одним із головних конкурентів Bootstrap. Він пропонує глибокий рівень кастомізації та надзвичайно потужну семантичну сітку, що робить його привабливим для розробки складних корпоративних систем. Однак високий поріг входження та загальна складність налаштування архітектури призвели до поступового зниження його популярності серед розробників масових вебпродуктів.

Своєю чергою, Bulma являє собою сучасний компонентний фреймворк, який повністю побудований на технології Flexbox [12]. Його головною архітектурною перевагою є повна відсутність залежності від JavaScript, що дозволяє розробникам інтегрувати його з будь-якими сучасними JS-бібліотеками (React, Vue, Angular) без ризику виникнення конфліктів скриптів. Проте, як і більшість компонентних рішень, Bulma страждає від проблеми надлишкового обсягу коду та демонструє суттєві обмеження гнучкості при спробах розробника відійти від стандартного візуального стилю фреймворку.

1.2.2 Аналіз рішень з utility-first підходом

Проводячи детальний аналіз існуючих рішень, що вирішують проблему стилізації за допомогою utility-first підходу (виступаючи прямою альтернативою Tailwind CSS), необхідно виділити такі інструментальні засоби, як Windi CSS та UnoCSS. Обидва проєкти створювалися з метою подолання обмежень швидкості компіляції ранніх версій Tailwind.

Інструмент Windi CSS пропонував розширені можливості групування класів та миттєве завантаження, проте з часом його активний розвиток зупинився, оскільки найкращі архітектурні ідеї Windi були імплементовані в офіційний Just-In-Time компілятор Tailwind CSS. Натомість UnoCSS представляє собою нове покоління атомарних CSS-рушіїв. Це екстремально легкий і швидкий інструмент, який взагалі не має базових стилів і працює виключно на основі правил генерації, що задаються розробником. UnoCSS ідеально підходить для створення власних utility-first мікрофреймворків, проте через свою високу складність конфігурації та відсутність широкої екосистеми готових рішень, він значно гірше підходить для швидкого старту типових проєктів.

Проєктування оптимального візуального інтерфейсу для систем електронної комерції є класичною задачею багатокритеріальної оптимізації, яка вимагає знаходження раціонального компромісу між часовими витратами на прототипування, технічними метриками продуктивності та наявністю розвиненої екосистеми інструментарію для забезпечення подальшої підтримки.

. Саме тому, зважаючи на недоліки альтернативних рішень, для подальшого практичного дослідження та порівняльного аналізу було обрано найбільш зрілі, стабільні та затребувані на ринку інструменти у своїх парадигмах — Bootstrap та Tailwind CSS. Загальна класифікація проаналізованих сучасних CSS-фреймворків за їхнім архітектурним підходом наведена на рис. 1.1.

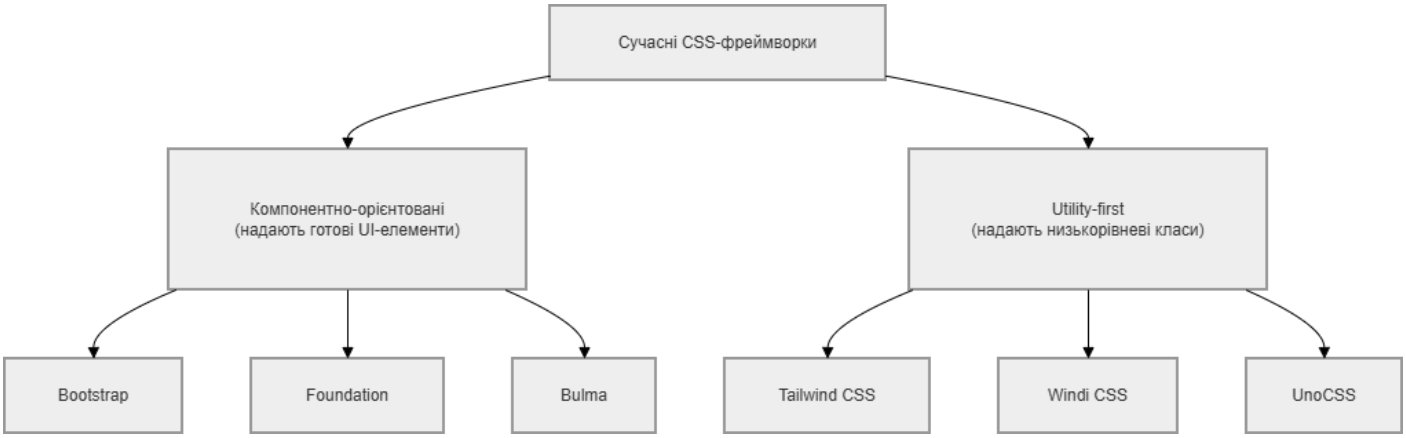


Рисунок 1.1 — Класифікація сучасних CSS-фреймворків за архітектурним підходом

1.3 Огляд підходів до стилізації та дизайну вебплатформ

Візуальний інтерфейс є фундаментальним елементом взаємодії користувача з будь-якою інформаційною системою [8]. У сфері електронної комерції якість користувацького інтерфейсу безпосередньо корелює з такими показниками, як:

а) коефіцієнт конверсії (відсоткове відношення кількості користувачів, які виконали цільову дію, до загальної кількості відвідувачів вебплатформи);

б) час сесії;

в) рівень утримання клієнтів (retention rate).

Історично підходи до стилізації еволюціонували від простого неструктурованого написання каскадних таблиць стилів (CSS) до використання складних архітектурних патернів та компільованих фреймворків. Зростання складності вебзастосунків вимагало від інженерів створення нових інструментів, здатних забезпечити швидку розробку (rapid development) без втрати продуктивності та гнучкості системи.

1.3.1 Архітектурні особливості традиційних CSS–підходів

Традиційний підхід до розробки візуальних інтерфейсів базується на фундаментальному інженерному принципі суворого розділення відповідальності (Separation of Concerns), де структура документа (HTML) концептуально та фізично відокремлена від його візуального представлення (CSS) [2, 9].

У цій парадигмі мова HTML (HyperText Markup Language) формує виключно семантичний каркас вебсторінки: вона визначає логічну ієрархію контенту та пряме призначення кожного елемента (заголовки, текстові блоки, списки, навігаційні панелі), повністю абстрагуючись від їхнього зовнішнього вигляду. Натомість каскадні таблиці стилів (CSS) виступають шаром представлення, який диктує браузеру точні правила відображення цих структурних елементів, включаючи типографіку, кольорову палітру, просторове позиціонування та адаптивність під різні розміри екранів.

Такий розподіл ролей гарантує високу гнучкість: розробник використовує семантичну розмітку для структурування даних, а всі візуальні правила виносить в окремі файли стилів. Це дозволяє повністю змінювати візуальний дизайн інформаційної системи (наприклад, проводити ребрендинг), модифікуючи лише CSS-код, без жодного втручання у базову HTML-розмітку. У цій парадигмі розробник використовує семантичну розмітку документа, а всі візуальні правила виносить в окремі файли стилів. Головною архітектурною проблемою такого підходу є глобальна область видимості (global scope) каскадних таблиць. Оскільки CSS спочатку розроблявся для форматування простих текстових документів, він не має вбудованих механізмів інкапсуляції стилів. У масштабованих інформаційних системах це призводить до того, що зміна стилю одного компонента може непередбачувано вплинути на відображення інших, абсолютно не пов'язаних з ним частин вебплатформи.

Зі збільшенням кодової бази виникає проблема «війн специфічності» (specificity wars), коли для перевизначення глобальних стилів розробники змушені створювати все більш складні селектори або використовувати

антипатерн !important [10]. Це порушує підтримуваність коду. Крім того, традиційний CSS схильний до проблеми «росту лише вгору» (append-only stylesheet problem): розробники, побоюючись зламати існуючий дизайн, вважають за краще додавати нові стилі в кінець файлу замість видалення або модифікації старих, що призводить до критичного розростання обсягу коду (CSS bloat).

Поява CSS-препроцесорів, таких як Sass (Syntactically Awesome Style Sheets) та LESS [11], стала першим кроком до інженерного підходу у стилізації. Вони додали в CSS змінні, міксини (mixins), вкладеність селекторів та математичні функції, що дозволило дотримуватися принципу DRY (Don't Repeat Yourself). Однак препроцесори не вирішили фундаментальної проблеми глобальної області видимості, оскільки їхній код все одно компілюється у звичайний глобальний CSS.

Для вирішення проблеми конфліктів імен спільнота розробила низку архітектурних методологій, найвідомішою з яких є BEM (Block, Element, Modifier) [4]. Методологія BEM пропонує структурувати код шляхом створення плоских, незалежних блоків з унікальними іменами класів.

Як показано на рис. 1.2, згідно з методологією BEM, кожен елемент жорстко прив'язаний до свого батьківського блоку (наприклад, product-card__title--active). Це мінімізує конфлікти специфічності, проте в умовах великих проєктів призводить до створення надзвичайно довгих класів та дублювання CSS-правил, оскільки для двох візуально однакових блоків з різним семантичним значенням доводиться писати окремі набори ідентичних стилів.

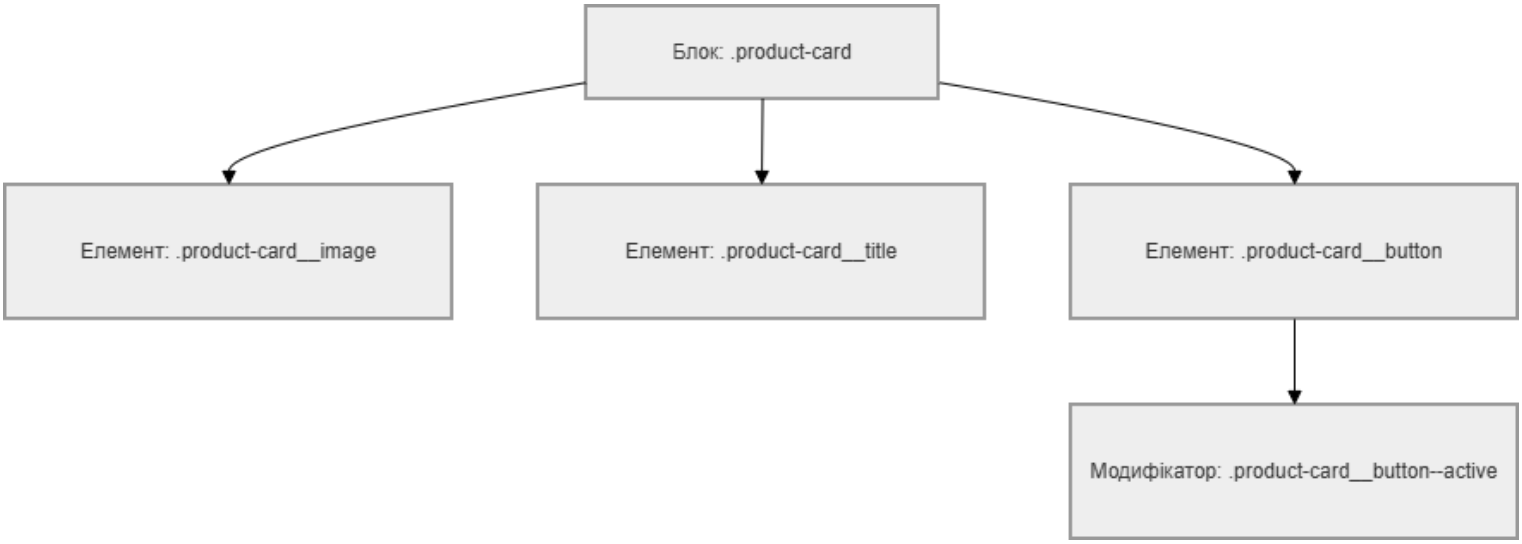


Рисунок 1.2 — Структурна схема взаємодії компонентів за методологією BEM

1.3.2 Архітектурні особливості компонентних фреймворків (Bootstrap)

Усвідомлення обмежень традиційного підходу та методології BEM призвело до створення готових бібліотек користувацького інтерфейсу у фреймворку Bootstrap (початкова назва Twitter Blueprint), який започаткував еру компонентно-орієнтованої стилізації.

В основі архітектури Bootstrap лежить концепція попередньо стилізованих, семантично названих компонентів. Розробнику не потрібно з нуля писати CSS для кнопок, навігаційних панелей, модальних вікон або карток товарів — достатньо додати відповідний клас (наприклад, .btn, .navbar, .card) до HTML-елемента. Фреймворк самостійно застосовує комплексний набір стилів, що інкапсульований у цьому класі. Це дозволяє екстремально пришвидшити процес прототипування та забезпечити сувору візуальну консистентність усієї інформаційної системи.

Важливим архітектурним досягненням Bootstrap стала його гнучка система сітки (Grid System), побудована на базі технології Flexbox [12]. Вона використовує 12-колонкову структуру та серію медіа-запитів для створення адаптивних макетів (рис. 1.3):

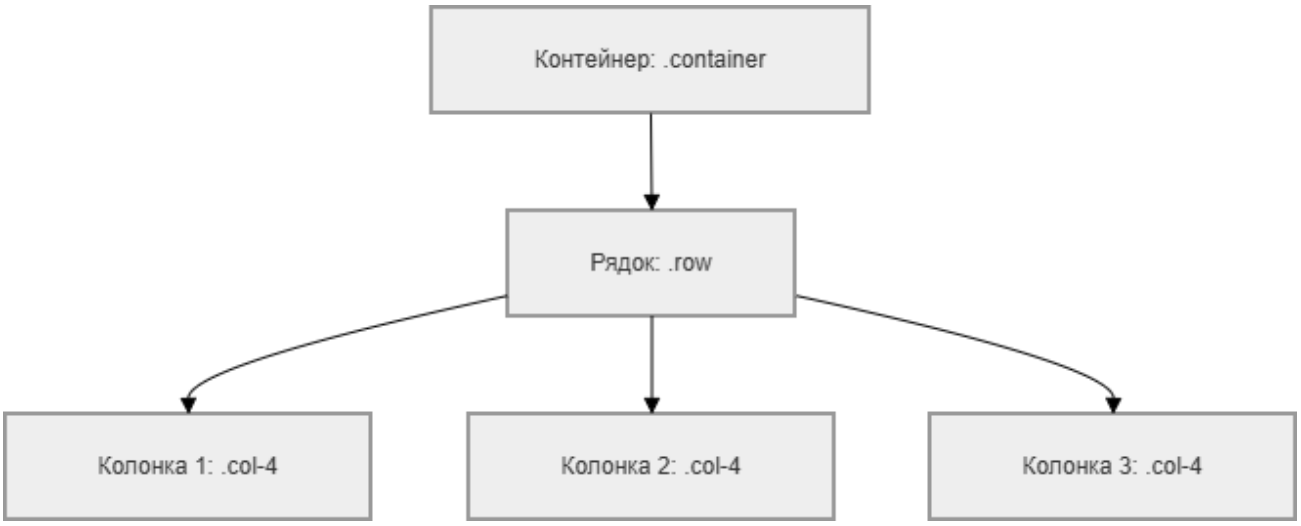


Рисунок 1.3 – Архітектура 12-колонкової системи сітки фреймворку Bootstrap

Незважаючи на незаперечні переваги у швидкості розробки, компонентна архітектура Bootstrap має низку суттєвих інженерних недоліків.

По-перше, вона нав'язує так званий «вигляд Bootstrap» (Bootstrap look). Оскільки всі компоненти мають заздалегідь визначений дизайн, вебплатформи, створені на його базі, часто виглядають однотипно.

По-друге, кастомізація під унікальний фірмовий стиль (наприклад, брендинг інтернетмагазину) вимагає значних зусиль. Розробнику доводиться писати об'ємні файли перевизначень (overrides), щоб перебити високу специфічність базових класів фреймворку.

По-третє, монолітна природа Bootstrap historically призводила до того, що користувач завантажував весь обсяг CSS-фреймворку (понад 150 КБ мініфікованого коду), навіть якщо на сторінці використовувалося лише 5% його можливостей. І хоча сучасні інструменти збірки дозволяють частково вирішувати цю проблему, архітектурно Bootstrap залишається важким компонентним рішенням.

1.3.3 Особливості utility-first підходу (Tailwind CSS)

У пошуках альтернативи, яка б дозволила уникнути проблеми розростання CSS-коду (CSS bloat) та обмежень жорстко заданого дизайну компонентних фреймворків, у фронтенд-спільноті сформувалася нова архітектурна парадигма – utility-first CSS [6]. Найяскравішим представником та фактичним індустріальним стандартом цього підходу на сьогодні є фреймворк Tailwind CSS [6]. Фундаментальна відмінність utility-first архітектури полягає у відмові від попередньо стилізованих семантичних компонентів на користь низькорівневих утилітарних класів. Кожен такий клас відповідає за одну конкретну CSS-властивість [13] (наприклад, .flex встановлює display: flex;, .pt-4 відповідає за padding-top, а .text-center — за вирівнювання тексту). Замість того, щоб створювати новий клас у зовнішньому файлі стилів для кожного унікального елемента інтерфейсу, розробник конструює дизайн безпосередньо в HTML-розмітці, комбінуючи необхідні утиліти. Візуальне порівняння традиційного семантичного та utility-first підходів до формування стилів наведено на рис. 1.4:

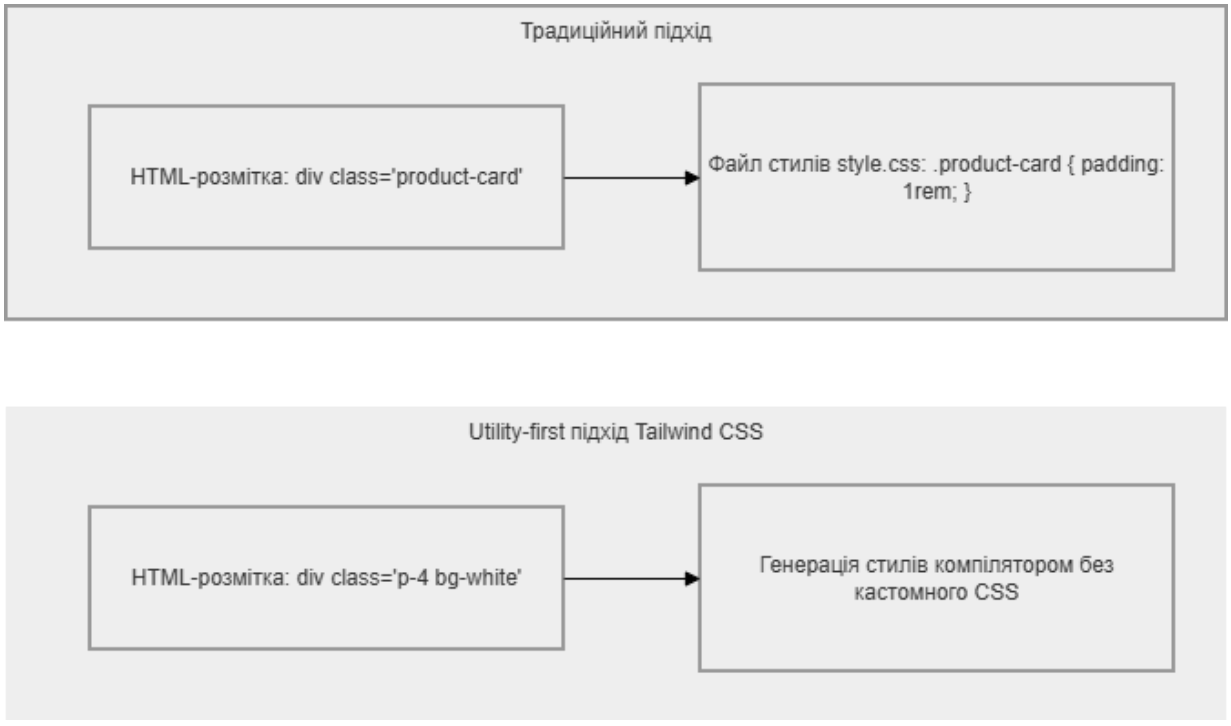


Рисунок 1.4 – Порівняльна схема традиційного та utility-first підходів до стилізації

Такий підхід вирішує одразу кілька архітектурних проблем.

По-перше, повністю усувається проблема глобальної області видимості та конфліктів специфічності, оскільки стилі застосовуються локально до конкретних елементів. По-друге, розробникам не потрібно витрачати час на вигадування семантичних назв для класів, що значно пришвидшує процес розробки та зменшує когнітивне навантаження.

Найважливішою інженерною перевагою Tailwind CSS є його механізм компіляції, який дозволяє екстремально зменшити розмір фінального бандлу (зазвичай до 10-20 КБ мініфікованого коду), що критично важливо для метрик продуктивності інтернет-магазинів.

Водночас utility-first підхід має свої недоліки. Перенесення всіх стилів у HTML призводить до перевантаження розмітки великою кількістю класів, що може ускладнювати читабельність коду (проблема «class soup»). Крім того, для уникнення дублювання коду при створенні однакових елементів (наприклад, кнопок), парадигма utility-first вимагає винесення логіки в перевикористовувані компоненти на рівні JavaScript-фреймворків (React, Vue, Angular) або шаблонізаторів [14], що робить цей підхід менш придатним для класичних статичних сайтів без використання компонентної архітектури на клієнті.

Проведений аналіз предметної області та еволюції архітектурних підходів до стилізації користувацьких інтерфейсів дозволяє зробити висновок, що сучасна фронтенд-розробка потребує зваженого підходу до вибору інструментальних засобів. З огляду на архітектурні недоліки традиційного написання каскадних таблиць стилів та обмеження методології BEM, індустрія зосередилася на двох домінуючих парадигмах: компонентній (Bootstrap) та utility-first (Tailwind CSS).

1.4 Постановка задачі дослідження

З огляду на проведенний аналіз предметної області, архітектурних особливостей існуючих інструментальних засобів та методологій стилізації, виникає необхідність формалізації науково-дослідного апарату даної роботи.

Об’єктом дослідження є процес проєктування та розробки візуальних користувацьких інтерфейсів сучасних інформаційних систем електронної комерції.

Предметом дослідження є архітектурні підходи, методи та інструментальні засоби впровадження стилів і дизайну на прикладі розробки інтерфейсних рішень інтернет-магазину з використанням технологій Bootstrap та Tailwind CSS.

Метою роботи є дослідження та здійснення комплексного порівняльного аналізу компонентного та utility-first підходів до стилізації інформаційних систем на основі визначеної системи об'єктивних технічних метрик для визначення їхньої архітектурної ефективності, гнучкості, впливу на швидкість розробки та швидкодію кінцевого програмного продукту.

Для досягнення поставленої мети у роботі необхідно розв’язати такі задачі:

- проаналізувати еволюцію, архітектурні особливості та сучасні підходи до стилізації користувацьких інтерфейсів (UI) інформаційних систем;
- дослідити теоретичні основи, принципи та методи багатокритеріальної оптимізації для прийняття рішень щодо вибору оптимального технологічного стеку фронтенд-розробки;
- спроєктувати еталонну архітектурну структуру тестових інтерфейсів електронної комерції, що включає індивідуальну сторінку товару, каталог із системою фільтрації та форму оформлення замовлення;
- реалізувати програмні прототипи зазначених інтерфейсів з паралельним використанням компонентного (Bootstrap) та utility-first (Tailwind CSS) підходів;

– провести інструментальні вимірювання технічних метрик продуктивності, швидкості рендерингу, розміру вихідного коду та мережевого навантаження для обох варіантів реалізації;

– виконати порівняльний аналіз отриманих експериментальних результатів із застосуванням методу узагальненого критерію оптимізації та сформувати обґрунтовані інженерні рекомендації щодо впровадження оптимального підходу в комерційних IT-проєктах.

2 МЕТОДИКА ОЦІНЮВАННЯ ІНСТРУМЕНТАЛЬНИХ ЗАСОБІВ СТИЛІЗАЦІЇ ІНФОРМАЦІЙНИХ СИСТЕМ

2.1 Порівняння компонентного та utility-first підходів

Ретроспективний аналіз еволюції методологій стилізації свідчить, що вибір між компонентно-орієнтованою (Bootstrap) та utility-first (Tailwind CSS) архітектурами є класичною задачею пошуку системного компромісу. У даному контексті часові витрати на початкове прототипування інтерфейсу та рівень інженерної гнучкості при його кастомізації перебувають у зворотній залежності від кінцевих метрик продуктивності системи. На практиці це означає, що висока швидкість розгортання стандартних рішень у компонентному підході супроводжується ризиком надлишковості коду, тоді як перехід до атомарних утиліт вимагає більшої деталізації на етапі проєктування, але гарантує мінімізацію мережевого навантаження та оптимізацію рендерингу сторінок.

Компонентний підхід ідеально підходить для швидкого запуску мінімально життєздатних продуктів (MVP) та створення адміністративних панелей, де візуальна унікальність не є пріоритетом. Натомість utility-first підхід демонструє свою ефективність у довгостроковій перспективі при розробці масштабованих платформ (зокрема, систем електронної комерції), де критично важливими є висока швидкодія, малий розмір кінцевих файлів та точна відповідність інтерфейсу унікальному фірмовому стилю (pixel-perfect дизайн).

Систематизоване відображення відмінностей досліджуваних архітектурних підходів за основними критеріями наведено у таблиці 2.1. Оскільки порівняння архітектурних підходів здійснюється за декількома критеріями, то надалі слід розглянути задачу багатокритеріального оцінювання, функції згортки критеріїв та методи вибору найкращого рішення.

Таблиця 2.1 – Порівняльний аналіз архітектурних підходів до стилізації вебплатформ

Критерій порівняння	**Компонентний підхід (Bootstrap)**	**Utility-first підхід (Tailwind CSS)**
Базова концепція	Використання готових, семантично названих UI-компонентів	Використання низькорівневих утилітарних класів
Швидкість розробки (старт)	Дуже висока (готові блоки)	Середня (потребує конструювання елементів з нуля)
Гнучкість та кастомізація	Низька (вимагає перевизначення базових стилів)	Дуже висока (повний контроль над дизайном)
Обсяг CSS-коду	Великий (генерується багато невикористаного коду)	Мінімальний (JIT-компілятор залишає лише використані класи)
Конфлікти специфічності	Можливі (через каскадну природу та глибоку вкладеність)	Відсутні (кожен клас відповідає за одну властивість)
Крива навчання	Низька (інтуїтивно зрозумілі назви компонентів)	Середня/Висока (необхідно вивчити систему найменувань утиліт)
Оптимальна сфера застосування	Панелі керування, внутрішні корпоративні системи, швидкі MVP	Високонавантажені платформи, інтернет-магазини, унікальні UI

2.2 Теоретичні основи багатокритеріального вибору альтернатив

Під час проєктування сучасних інформаційних систем часто виникає задача вибору оптимального технологічного рішення з множини наявних альтернатив за кількома критеріями, які нерідко є суперечливими між собою. Згідно з базовими положеннями теорії прийняття рішень [20], для формалізації

та розв'язання задач багатокритеріальної оптимізації застосовуються такі фундаментальні принципи та підходи:

1. Принцип головного критерію. Його суть полягає у виділенні одного, найбільш вагомого показника $k_i(x)$, який оптимізується (максимізується або мінімізується), тоді як на решту критеріїв накладаються жорсткі граничні обмеження у вигляді нерівностей. Математично цей принцип описується системою:

$$\begin{cases} k_0(x) \to max_{x \in X} \text{ (або } min) \\ k_i(x) \geq k_i^{min}, \ \ i = \overline{1,m}, \ i \neq 0 \end{cases}, \qquad (2.1)$$

де X – множина допустимих альтернатив;

xi – конкретна досліджувана альтернатива, $xi \in X$;

$k_0(x)$ – значення головного критерію;

$k_i(x)$ – значення і-го допоміжного критерію;

k_i^{min} – мінімально допустиме граничне значення для і-го критерію;

m — загальна кількість критеріїв.

2. Принцип послідовної оптимізації (лексикографічного упорядкування). Ідея цього методу полягає в трансформації багатокритеріальнох оптимізаційної задачі в упорядковану послідовність однокритеріальних. Для цього всі частини критерії упорядковуються в послідовності спадання вадливості, тобто встановлюється лінійний порядок

$$k_1 \succ k_2 \succ \ldots \succ k_n$$

де $\succ$ – знак відношення переваги.

У цій послідовності розв’язуються однокритеріальні задачі за кожним частинним критерієм.

3. Функціонально-вартісний аналіз. Вихідна множина частинних критеріїв $K = \{k_i(x)\}, \overline{i = 1,n}$, що за визначенням досить повно характеризує

ефективність допустимих рішень $x \in X$ у цілому, розбивається на дві підмножини: $К_ф = \{k_j(x)\}, \overline{j = 1, m}$ і $К_В = \{k_l(x)\}, \overline{l = 1, L}; m + L = n.$ Перша група критеріїв $К_ф$ характеризує функціональну якість рішення, тобто ступінь досягнення цілі системи, що аналізується, а друга група $К_В$ – витрати, потрібні для реалізації рішення х, тобто досягнення цілі. З кожної з зазначених підмножин виділяється один головний критерій; позначимо їх відповідно $К_Ф^*$ і $К_В^*$, а інші частини критерії переводяться в обмеження. Оптимальне рішення має вид

$$x_2^0 = \max_{x \in X^*} \left[\frac{K_Ф^*(x)}{K_В^*(x)} \right]$$

4. Метод узагальненого критерію (адитивна згортка). Полягає у формуванні єдиного інтегрального показника ефективності як зваженої суми нормованих значень усіх локальних критеріїв. Цей метод є найбільш раціональним для порівняння технологічних стеків, оскільки дозволяє врахувати різнопланові метрики. Математична модель адитивної згортки має вигляд:

$$P(x) = \sum_{i=1}^{n} a_i k_i^H(x) \qquad (2.2)$$

де $P(x) -$ узагальнена оцінка ефективності альтернативи;

$k_i^H(x) -$ нормалізовані, тобто зведені до ізоморфного вигляду частинні критерії;

a_i – безрозмірні вагові коефіцієнти відносної важливості частинних критеріїв, для яких виконуються обмеження

$$\sum_{i=1}^{n} a_i = 1, \text{ при цьому } 0 \le a_i \le 1 \qquad (2.3)$$

Оскільки початкові інструментальні метрики мають різні одиниці виміру (кілобайти, лінії коду, бали), перед обчисленням узагальненого критерію за формулою (2.2) здійснюється процедура їхнього лінійного нормування. Для умов порівняння двох альтернатив (*m=2*) значення вагових коефіцієнтів a_i зводяться до бінарної шкали, де найкращому результату за конкретним критерієм присвоюється значення $a_i = 1$, а найгіршому $a_i = 0$.

2.3 Критерії оцінки та обґрунтування вибору метрик порівняння

Для забезпечення однозначності термінології в межах даного дослідження, загальні критерії оцінювання формалізуються через конкретні технічні метрики:

а) швидкодія та продуктивність кінцевого продукту вимірюється інтегральною метрикою Performance Score (інструмент Google Lighthouse);

б) мережеве навантаження вимірюється двома метриками: обсягом мініфікованого CSS-бандлу (Кб) та сумарним розміром переданих ресурсів;

в) гнучкість кастомізації формалізується через кількість кастомного CSS-коду (у рядках), який розробнику доводиться писати вручну понад можливості фреймворку;

г) ризик перевантаження розмітки оцінюється через метрику обсягу згенерованого HTML-файлу (Кб).

Для проведення всебічного порівняльного аналізу архітектурних підходів до стилізації (компонентного та utility-first) необхідно визначити систему показників, які дозволяють оцінити ефективність інструментальних засобів з різних точок зору. Теоретично, порівняння таких фреймворків, як Bootstrap та Tailwind CSS, може здійснюватися за широким спектром критеріїв.

1. Суб’єктивні (якісні) показники:

   а) швидкість навчання (Learning Curve): час, необхідний розробнику для опанування синтаксису та методології фреймворку;

б) зручність підтримки (Maintainability): оцінка складності внесення змін у кодову базу через 6–12 місяців після завершення активної фази розробки;

в) якість документації та спільноти: наявність готових рішень, бібліотек компонентів та швидкість пошуку відповідей на технічні питання.

2. Проєктувальні критерії:

а) гнучкість дизайну (Design Flexibility): здатність інструменту точно реалізувати унікальний макет (pixel-perfect) без конфліктів із базовою дизайн-системою;

б) масштабованість архітектури: можливість ефективно використовувати напрацювання при розростанні проєкту від однієї сторінки до великої платформи.

3. Об’єктивні (технічні) метрики:

а) мережеве навантаження (Payload Size): обсяг даних, що передаються по мережі;

б) швидкість візуалізації (Rendering Speed): час, що витрачається браузером на побудову дерева стилів та відмальовування контенту.

Враховуючи специфіку системного проєктування інтерфейсів електронної комерції, де продуктивність безпосередньо впливає на бізнес-показники, для дослідження обрано шість ключових метрик:

1) обсяг фінального CSS-бандлу (Payload Size) - це підсумковий оптимізований файл каскадних таблиць стилів, який генерується автоматизованими інструментами збірки (бандлерами) з вихідного коду для подальшого завантаження браузером кінцевого користувача у робочому (production) середовищі. Вимірюється в кілобайтах (Кб) після мініфікації та оптимізації. Ця метрика є критичною для швидкодії, оскільки безпосередньо впливає на час завантаження та рендерингу сторінки, особливо в умовах обмеженого інтернет-з'єднання;

2) складність та обсяг HTML-розмітки: оцінюється через розмір HTML-файлу в Кб. Це дозволяє проаналізувати побічний ефект utility-first підходу – так званий «class soup», який може ускладнювати читабельність та потенційно збільшувати час парсингу документа браузером;

3) кількість кастомного CSS-коду: обчислюється як кількість рядків стилів, які розробнику довелося написати вручну понад можливості фреймворку. Показник демонструє гнучкість інструменту та його здатність адаптуватися до унікального дизайну без виникнення конфліктів специфічності;

4) сумарне мережеве навантаження (Total Network Payload): вага всіх ресурсів стилізації та допоміжних скриптів, що завантажуються браузером. Це дозволяє оцінити загальну ефективність обраної архітектури в контексті кешування та зовнішніх залежностей;

5) швидкість прототипування та кастомізації: якісна метрика, що оцінює часові витрати на реалізацію унікальних UI-елементів, які виходять за межі стандартної бібліотеки компонентів;

6) комплексна продуктивність за Google Lighthouse (Performance Score): інтегральний показник (0–100), що базується на алгоритмах Google. Він підсумовує технічні параметри (вагу файлів, швидкість рендерингу, Core Web Vitals) у єдину оцінку якості користувацького досвіду, що є стандартом для сучасних вебплатформ.

Вибір саме такої системи показників дозволяє охопити всі рівні функціонування інтерфейсу: від фізичного обсягу коду до суб’єктивного сприйняття швидкодії користувачем.

2.4 Обґрунтування вибору технологічного стеку

Для об'єктивного підтвердження теоретичних висновків та оцінки реальної ефективності фреймворків Bootstrap та Tailwind CSS виникає необхідність у створенні стандартизованого середовища порівняння. У рамках даного дослідження таким середовищем виступатиме прототип користувацького

інтерфейсу сторінки товару для гіпотетичної інформаційної системи електронної комерції (інтернетмагазину). Вибір саме такого типу інтерфейсу зумовлений його високою структурною складністю, наявністю різноманітних UI-елементів та суворими вимогами до адаптивності на різних типах пристроїв.

При проведенні дослідження необхідно дотримуватися наступних вимог:

а) суворе дотримання принципу Mobile-First, що передбачає початкове проєктування інтерфейсу для мобільних пристроїв із подальшою адаптацією під планшетні та десктопні екрани;

б) тестова сторінка повинна містити кілька обов'язкових функціональних блоків:

- глобальну навігаційну панель (Header);
- блок основної інформації про товар (який включає інтерактивну галерею зображень, секцію вибору параметрів товару та блок заклику до дії — кнопку додавання у кошик);
- інформаційний підвал сайту (Footer).

Архітектурна структура тестового інтерфейсу, яка повинна бути реалізована за допомогою досліджуваних інструментальних засобів, наведена на рис. 2.1.

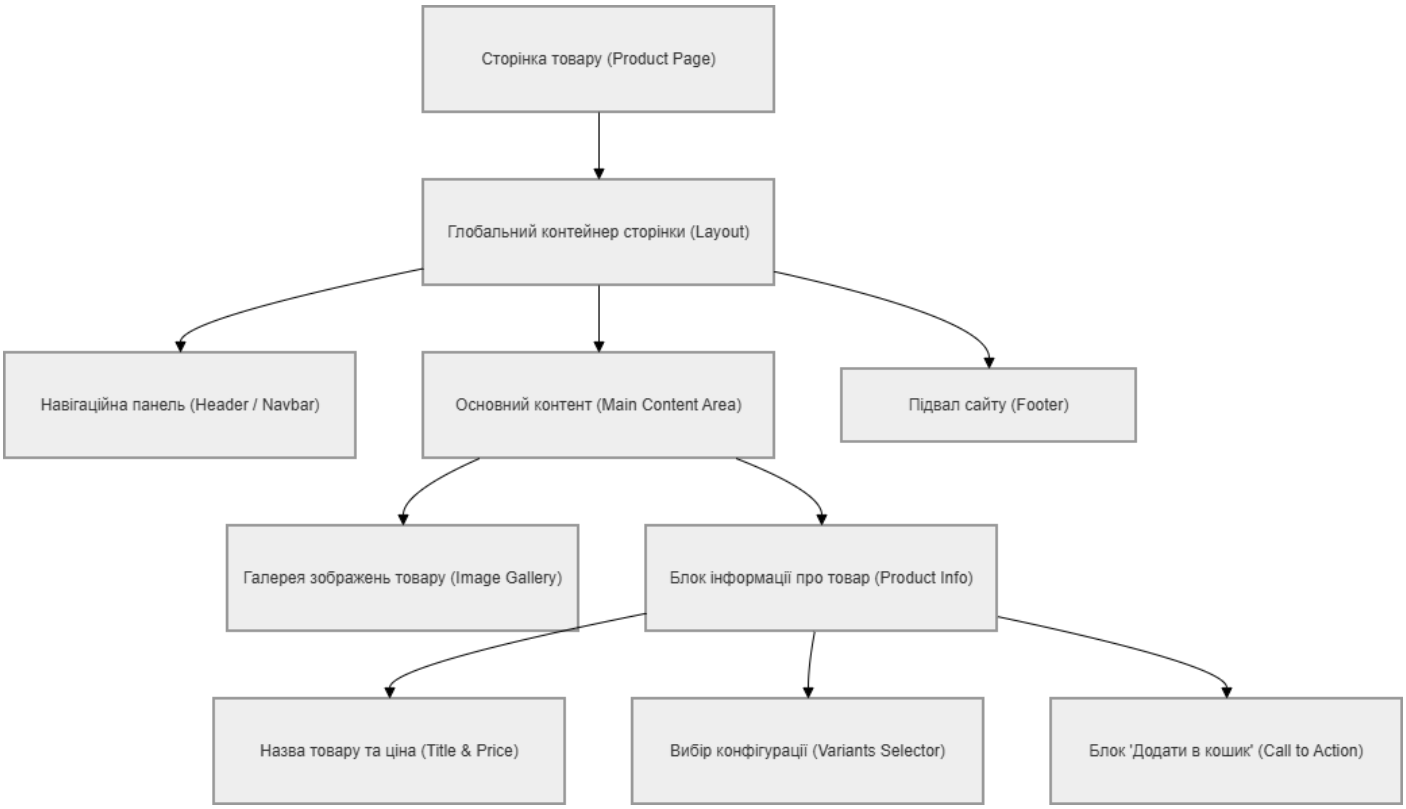


Рисунок 2.1 —Структура тестового інтерфейсу

2.5 Обґрунтування вибору технологічного стеку

Для забезпечення об'єктивності результатів практичного дослідження та можливості точного вимірювання визначених у підрозділі 1.4 метрик, було сформовано стек технологій, що є індустріальним стандартом сучасної фронтенд-розробки. Вибір кожного інструменту зумовлений необхідністю отримання чистих даних для порівняльного аналізу.

Фундаментом для розгортання експериментального середовища обрано платформу Node.js [15] у поєднанні з менеджером пакетів NPM. Це дозволяє уніфікувати процес управління залежностями та версіями фреймворків Bootstrap та Tailwind CSS, забезпечуючи ідентичність умов для обох технологій.

Головним інструментом збірки (бандлером) обрано Vite [16]. На відміну від традиційних інструментів, Vite забезпечує миттєве оновлення модулів (HMR) та, що є критично важливим для нашого дослідження, надає точні звіти про обсяг

фінальних бандлів після оптимізації. Використання саме цього бандлера дозволяє об'єктивно виміряти метрику «Обсяг згенерованого CSS-коду» (Payload Size) завдяки його вбудованим алгоритмам мініфікації та підтримці Just-In-Time компіляції для Tailwind CSS.

Для розробки структури інтерфейсів обрано мову розмітки HTML5 без використання високорівневих JavaScript-фреймворків (таких як React чи Vue). Такий підхід дозволяє повністю ізолювати вплив CSS-фреймворків на загальну вагу сторінки та виключити похибки в метриках, що могли б виникнути через час виконання сторонніх скриптів.

З метою підвищення репрезентативності дослідження та формування науково обґрунтованої вибірки даних, обраний технологічний стек дозволяє масштабувати аналіз за межі поодинокої вебсторінки. Створена архітектура проєкту підтримує оперативну розробку та паралельну збірку серії типових інтерфейсів електронної комерції:

а) індивідуальна сторінка товару (Product Page);

б) каталог товарів із системою фільтрації (Product Listing Page);

в) кошик та сторінка оформлення замовлення (Checkout).

Такий комплексний підхід до формування тестового середовища дозволяє узагальнити отримані значення метрик на основі різних за складністю інтерфейсних структур, що забезпечує високу достовірність фінальних висновків роботи.

2.6 Визначення архітектурних особливостей тестових інтерфейсів

Проєктування користувацького інтерфейсу сторінки товару вимагає чіткого визначення архітектурних патернів, які будуть застосовані під час верстки. Оскільки тестовий інтерфейс розробляється паралельно з використанням двох різних технологій, необхідно виділити спільні структурні вимоги до макета. Фундаментальною архітектурною особливістю обох рішень є використання гнучкої сітки для організації контенту. У випадку фреймворку Bootstrap ця задача

вирішується за допомогою класичної 12-колонкової системи на базі Flexbox, де просторовий розподіл елементів керується набором жорстко стандартизованих класів. Натомість Tailwind CSS дозволяє імплементувати сучасніший підхід на базі технології CSS Grid [17], що забезпечує точніший контроль над розмірами колонок та проміжками між ними без необхідності створення додаткових обгорток (wrapper elements) у розмітці.

Архітектура адаптивності будується за принципом поступового розширення робочої області (Mobile-First). На екранах мобільних пристроїв основні функціональні блоки сторінки (галерея зображень та блок інформації про товар) шикуються в єдину вертикальну колонку для забезпечення максимальної читабельності контенту. При досягненні першої контрольної точки розриву (breakpoint), яка відповідає розмірам планшетів та десктопних моніторів, архітектура макета трансформується у двоколонкову структуру. Логіка просторової трансформації тестового інтерфейсу залежно від розміру екрана наведена на рис. 2.2.

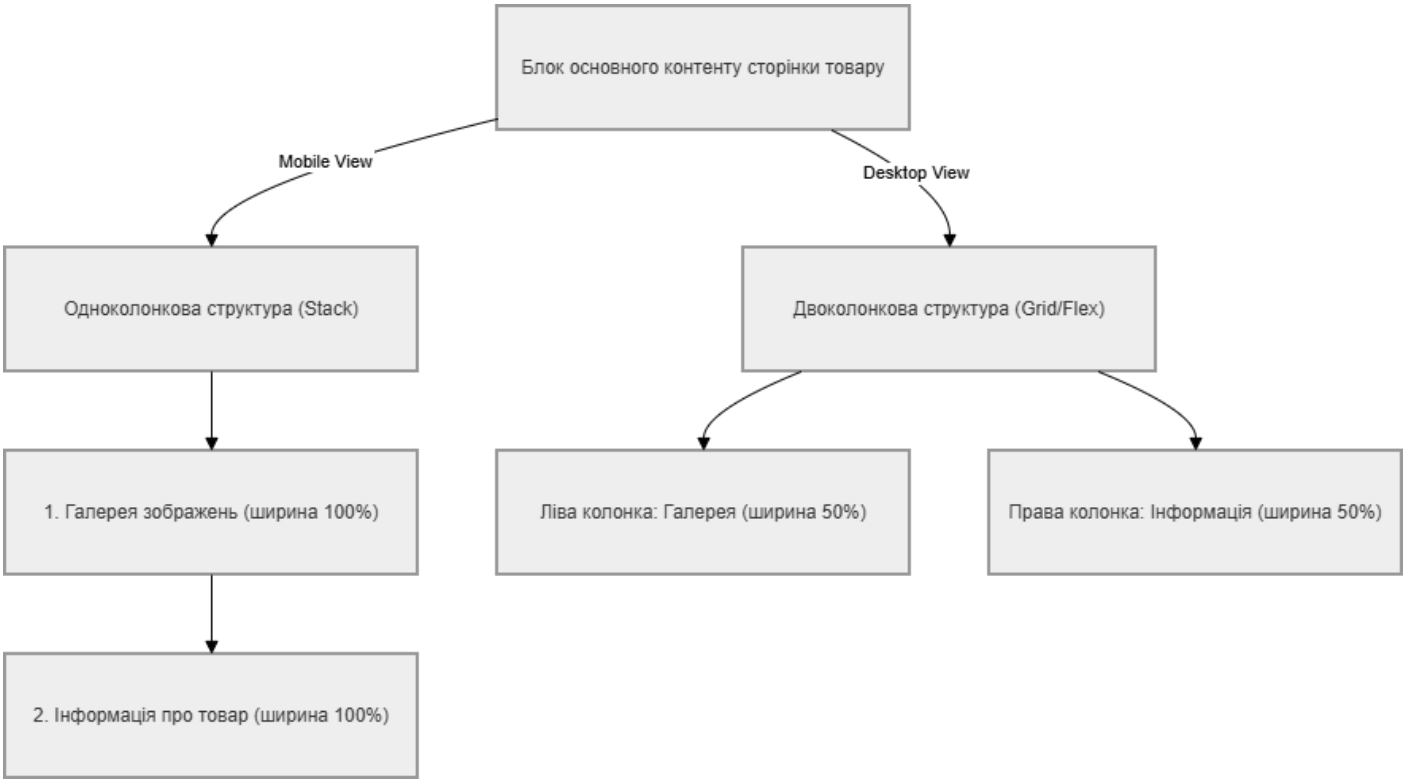


Рисунок 2.2 — Схема просторової трансформації сітки макета сторінки товару

Окрім глобальної сітки, архітектурного проєктування вимагають і внутрішні мікрокомпоненти. Наприклад, галерея зображень проєктується як модульний блок, що складається з головного екрана перегляду (main view) та сітки мініатюр (thumbnails). Блок інформації про товар містить складну типографічну ієрархію (заголовок, ціна, опис) та інтерактивні елементи форм [8] (вибір розміру, кольору, кнопка додавання в кошик). Структурна організація основного контентного блоку з використанням utility-класів Tailwind CSS для формування адаптивної сітки наведена у лістингу 2.2.

Лістинг 2.2 — Базова архітектура сітки сторінки товару (на прикладі Tailwind CSS)

```
<main class="container mx-auto px-4 py-8">
  <div class="grid grid-cols-1 md:grid-cols-2 gap-8">

    <section class="flex flex-col gap-4">
      <div class="aspect-square bg-gray-100 rounded-xl overflow-hidden">
        <img src="product-main.jpg" alt="Фото товару" class="object-cover w-full h-full">
      </div>
      <div class="grid grid-cols-4 gap-2">
        <div class="aspect-square bg-gray-200 rounded-md cursor-pointer"></div>
      </div>
    </section>

    <section class="flex flex-col space-y-6">
      <div class="border-b border-gray-200 pb-4">
```

```
            <h1 class="text-3xl font-bold text-gray-900 tracking-tight">Назва товару</h1>
            <p class="mt-2 text-2xl font-semibold text-brand-primary">1 500 ₴</p>
          </div>
          <button class="w-full bg-brand-primary text-white py-3 rounded-lg hover:bg-blue-700 transition-colors">
            Додати в кошик
          </button>
        </section>

      </div>
    </main>
    ]
```

Програмний код у лістингу 2.2 наочно демонструє ключову особливість utility-first архітектури: формування складного адаптивного макета (через класи grid-cols-1 md:grid-cols-2) здійснюється безпосередньо в структурі HTML-документа, без написання жодного рядка кастомного CSS-коду. Цей базовий каркас буде використано для подальшої детальної розробки інтерфейсу та його порівняння з аналогічним рішенням, реалізованим за допомогою компонентів Bootstrap.

Крім базової сторінки товару, для отримання об'єктивних метрик було спроєктовано та реалізовано додаткові типи інтерфейсів, що мають принципово відмінну структуру та функціональне навантаження:

а) каталог товарів (Product Listing Page): архітектура цього інтерфейсу базується на циклічному відтворенні однотипних компонентів (карток товарів) у межах адаптивної сітки. Головним викликом для стилізації тут є організація системи фільтрації у бічній панелі та забезпечення консистенції елементів при зміні щільності сітки;

б) сторінка оформлення замовлення (Checkout Page): на відміну від контент-орієнтованих сторінок, цей інтерфейс зосереджений на взаємодії з формами введення даних. Архітектурно він складається з багатоколонкової структури, що містить поля для персональних даних, блоки вибору методів доставки та оплати.

Розширення переліку тестових об’єктів забезпечує необхідну репрезентативність експерименту. Це дозволяє стверджувати, що отримані результати порівняння метрик не є випадковими для одного макета, а відображають системну закономірність функціонування обраних фреймворків у різних сценаріях використання.

2.7 Визначення методів та алгоритмів впровадження стилів

Процес впровадження стилів у користувацькі інтерфейси докорінно відрізняється залежно від обраної архітектурної парадигми. Для коректного проведення порівняльного аналізу необхідно формалізувати алгоритми роботи інженера при проєктуванні інтерактивних елементів тестової сторінки товару (наприклад, кнопки додавання у кошик або карток характеристик). Механізм роботи з компонентним фреймворком Bootstrap базується на принципі успадкування та перевизначення. Алгоритм стилізації розпочинається з аналізу вимог до UI-елемента, після чого розробник застосовує стандартизовані семантичні класи (наприклад, .btn та .btn-primary). У разі, якщо базовий дизайн компонента задовольняє вимоги проєкту, процес стилізації завершується. Проте при розробці унікального інтерфейсу електронної комерції виникає потреба у кастомізації. У такому випадку алгоритм ускладнюється: інженер змушений створювати додаткові CSS-файли, писати кастомні селектори з вищою специфічністю або втручатися в SCSS-змінні фреймворку для перевизначення стандартних тіней, відступів чи кольорів.

Натомість алгоритм впровадження стилів за методологією utility-first, яку пропонує Tailwind CSS, є більш лінійним. Розробник конструює дизайн елемента

безпосередньо під час написання HTML-розмітки, послідовно додаючи атомарні утилітарні класи, кожен з яких відповідає за єдину візуальну властивість. Якщо виникає потреба у застосуванні специфічного значення, яке відсутнє у базовій дизайн-системі (наприклад, унікальний шістнадцятковий код кольору або нетиповий розмір тіні), алгоритм передбачає використання механізму довільних значень (arbitrary values) безпосередньо у класі або розширення глобальної теми у файлі конфігурації. Такий підхід повністю виключає необхідність створення зовнішніх файлів стилів та гарантує відсутність побічних ефектів для інших компонентів системи. Алгоритмічна різниця між процесами кастомізації UI-елементів у досліджуваних фреймворках показана за допомогою схем на рис. 2.3.

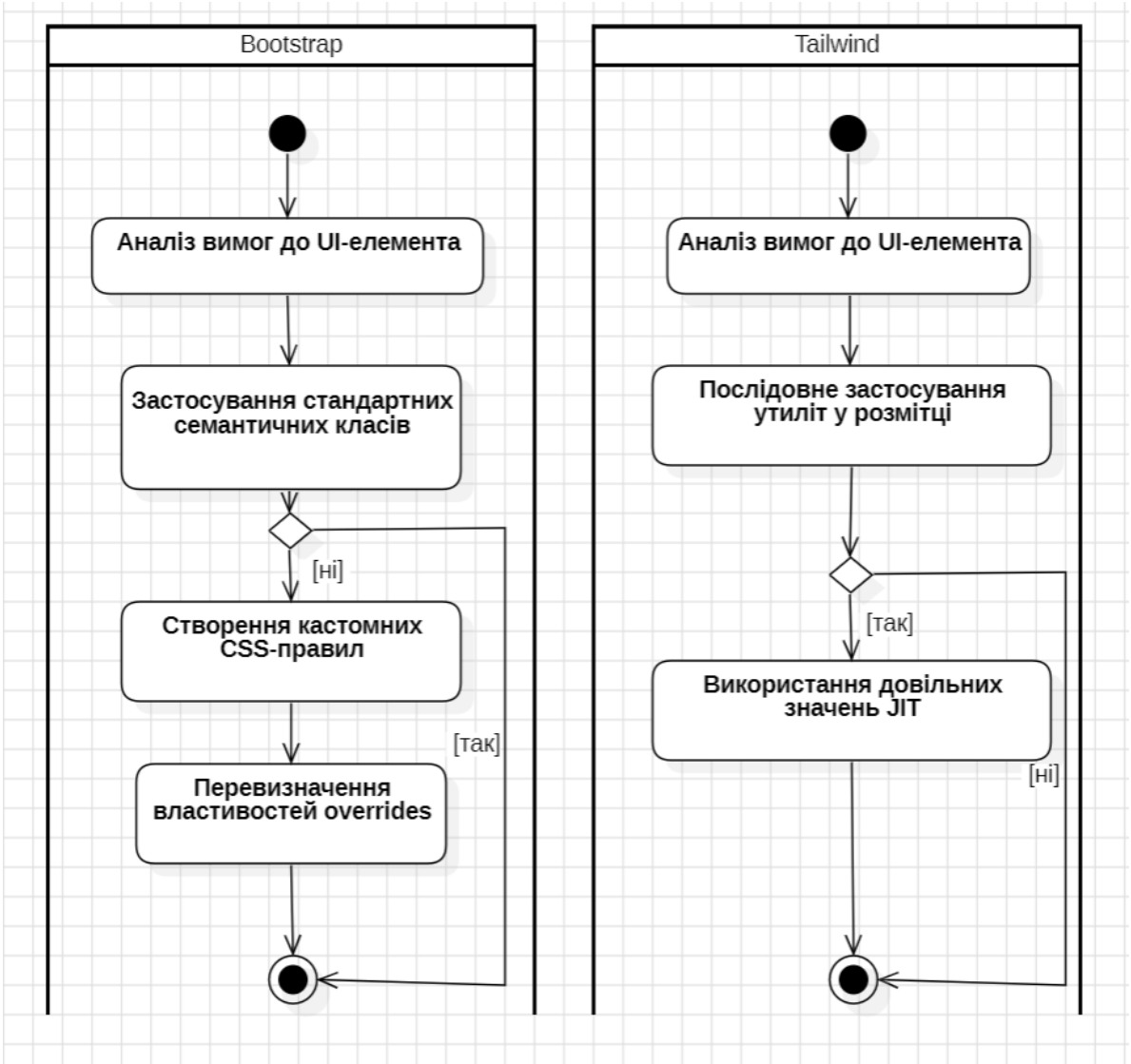


Рисунок 2.3 — UML-діаграма діяльності (Activity diagram) процесу кастомізації UI-елементів

Для практичної демонстрації описаних алгоритмів було створено ключовий елемент взаємодії сторінки товару — кнопку заклику до дії (Call to Action). У лістингу 2.3 наведено порівняльний програмний код імплементації ідентичного за візуальними характеристиками елемента з використанням обох технологій.

Лістинг 2.3 — Порівняльна реалізація елемента взаємодії (кнопки)

HTML

```
<button class="btn btn-primary custom-cart-btn">
  Додати в кошик
</button>

<style>
  /* Алгоритм вимагає написання перевизначаючих стилів */
  .custom-cart-btn {
    background-color: #0d6efd;
    border-radius: 8px;
    padding: 12px 24px;
    font-weight: 500;
  }
  .custom-cart-btn:hover {
    background-color: #0b5ed7;
  }
</style>
```

```
<button class="bg-blue-600 hover:bg-blue-700 text-white font-medium py-3 px-6 rounded-lg transition-colors">
  Додати в кошик
```

```
</button>
```

Аналіз наведеного програмного коду підтверджує, що метод впровадження стилів через утилітарні класи дозволяє зберігати всю інформацію про зовнішній вигляд компонента в єдиному місці (у розмітці). У випадку з Bootstrap, незважаючи на наявність базового класу .btn, досягнення унікального дизайну неминуче призводить до фрагментації кодової бази та необхідності підтримки додаткових таблиць стилів.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

3.1 Розробка компонентів та особливості їх інтеграції в проєкт

Підсумковим етапом практичного дослідження стала повна програмна реалізація прототипу сторінки товару із застосуванням обох досліджуваних архітектурних підходів та комплексна оцінка ефективності їх інтеграції у тестове середовище. Процес розробки був спрямований на створення візуально ідентичних інтерфейсів, що дозволило забезпечити чистоту подальшого метричного аналізу.

Інтеграція компонентного фреймворку Bootstrap здійснювалася класичним шляхом підключення попередньо скомпільованого ядра бібліотеки з подальшим нашаруванням кастомних файлів стилів. У ході реалізації основних модулів було встановлено, що для досягнення повної відповідності унікальному дизайну інтернет-магазину розробнику довелося написати близько 140 рядків кастомного CSS-коду. Цей додатковий обсяг коду був зумовлений необхідністю примусового перевизначення стандартних тіней, відступів та кольорових палітр, які жорстко інкапсульовані в базових компонентах фреймворку, що підтверджує проблему високої специфічності компонентних рішень.

З іншого боку, впровадження архітектури Tailwind CSS відбувалося на рівні конфігурації бандлера Vite, що дозволило повною мірою задіяти потенціал JIT-компіляції. Практична апробація цього підходу підтвердила можливість повного уникнення написання зовнішніх каскадних таблиць стилів, оскільки весь необхідний візуальний функціонал був реалізований безпосередньо через декларативні класи у розмітці. Для забезпечення системності дизайну було використано конфігураційний файл теми, що дозволило розширити стандартну бібліотеку утиліт специфічними параметрами брендингу без збільшення складності кодової бази.

Окрім базової сторінки товару, для отримання репрезентативних даних було спроєктовано та реалізовано додаткові типи інтерфейсів, зокрема каталог

товарів із системою фільтрації та сторінку оформлення замовлення, які мають принципово відмінну структуру та функціональне навантаження. Архітектура каталогу базувалася на циклічному відтворенні однотипних компонентів, тоді як інтерфейс оформлення замовлення вимагав складної стилізації інтерактивних форм введення даних . Розширення переліку тестових об’єктів дозволило стверджувати, що виявлені архітектурні особливості фреймворків не є випадковими для одного макета, а відображають системну закономірність їх функціонування у різних сценаріях використання.

Загальний результат розробки продемонстрував, що обидві технології здатні забезпечити високу якість інтерфейсу, проте метод впровадження стилів через утилітарні класи дозволяє зберігати всю інформацію про зовнішній вигляд компонента в єдиному місці, запобігаючи фрагментації коду. Візуальна демонстрація отриманих експериментальних прототипів наведена на рисунках 3.1–3.3, де представлено порівняння стилізації елементів, реалізацію каталогу та роботу з формами на сторінці замовлення .

На рисунку 3.1. продемонстровано порівняння стилізації елементів інтерфейсу: зліва — компонентний підхід (Bootstrap), справа — "utility-first" підхід (Tailwind CSS). Метою програмної реалізації було досягнення абсолютної візуальної ідентичності (pixel-perfect) обох прототипів для забезпечення чистоти експерименту. На рисунку 3.1 наведено візуальне порівняння реалізації базових елементів інтерфейсу (галереї зображень та інформаційного блоку) за допомогою компонентного підходу (а) та utility-first підходу (б). Як видно з макетів, обидві технології успішно впоралися з відтворенням складної типографіки та позиціонуванням. На рисунку 3.2 (а, б) зображено прототипи каталогу товарів. Головним завданням на цьому етапі була перевірка консистентності відображення адаптивної сітки карток товарів та інтерактивної панелі фільтрації. Рисунок 3.3 (а, б) ілюструє роботу з формами на сторінці оформлення замовлення. Ми можемо побачити, що обидва фреймворки дозволяють гнучко стилізувати елементи введення даних (input fields) та радіокнопки згідно з фірмовим стилем інтернет-магазину

А) Б)

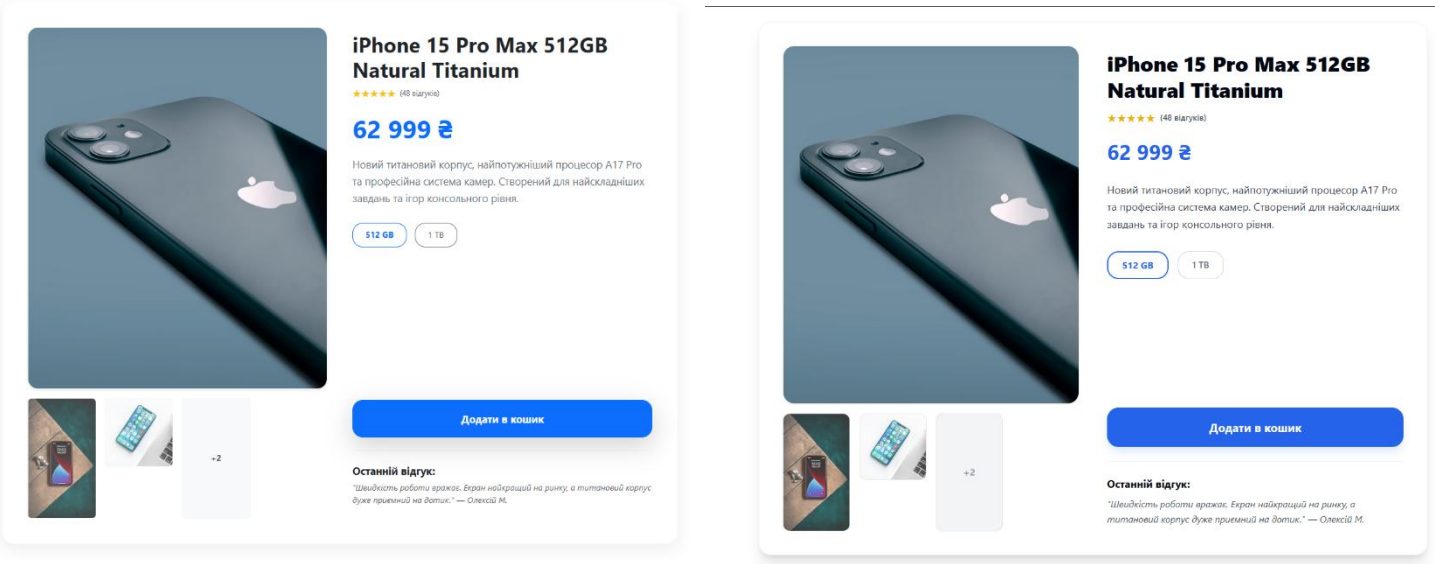
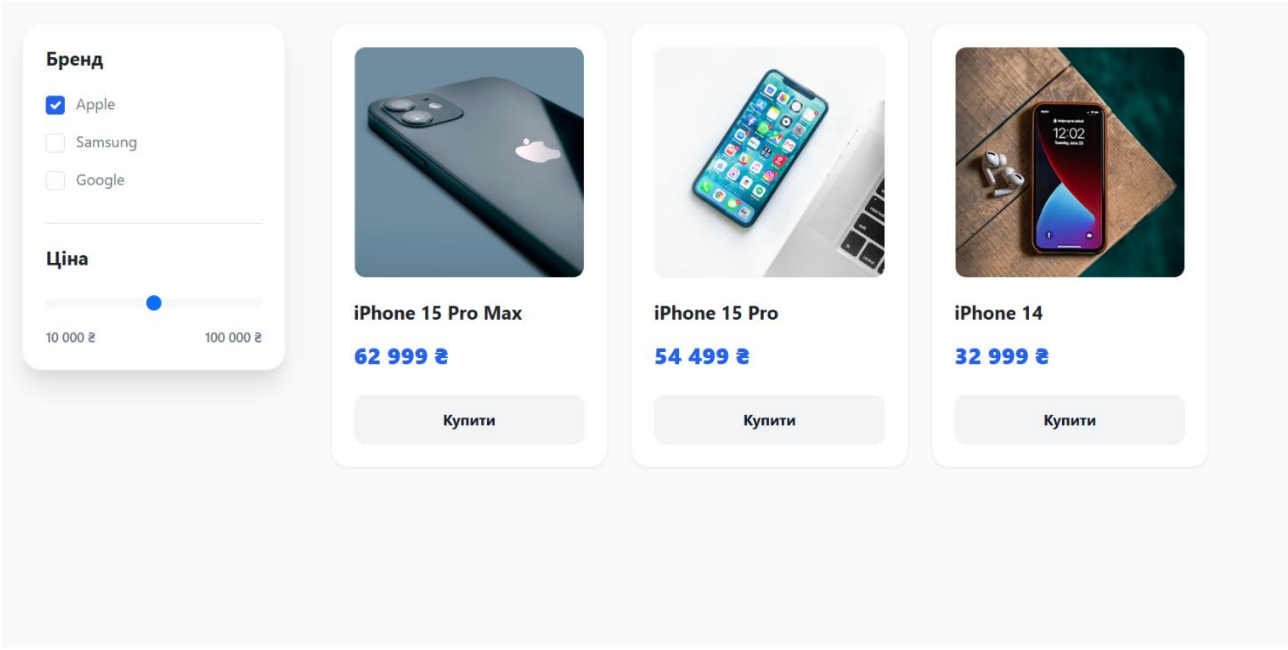


Рисунок 3.1 — Візуальна демонстрація порівняння стилізації елементів інтерфейсу: А) Bootstrap; Б) Tailwind CSS

А)



Б)

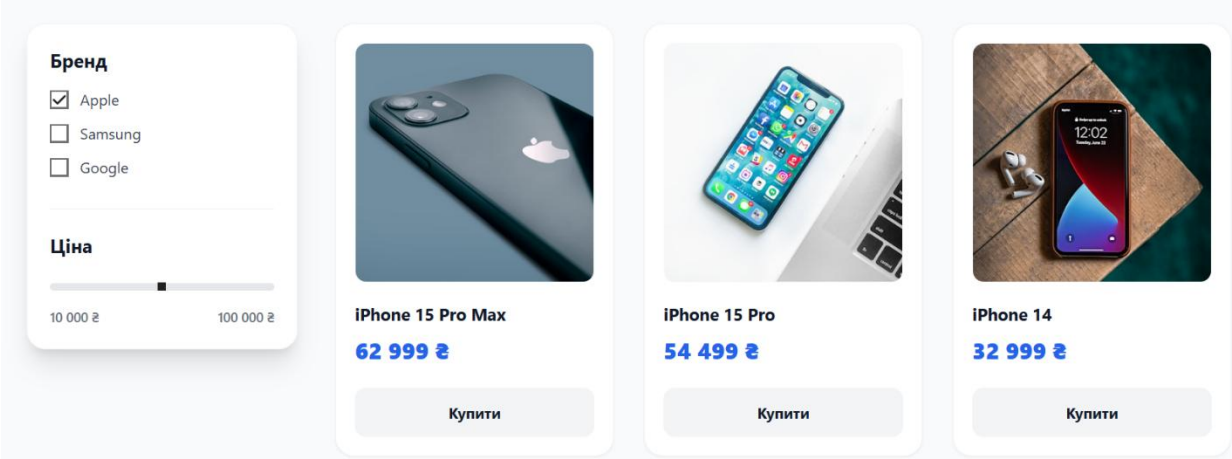
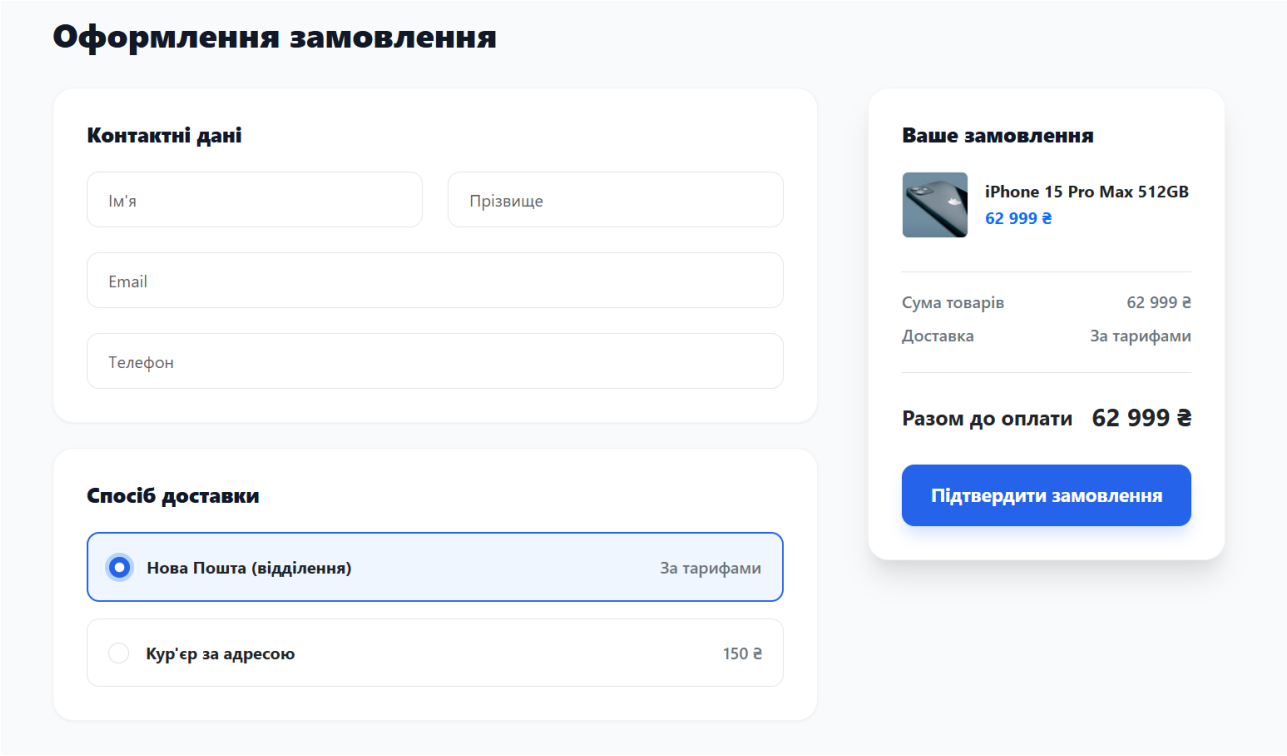


Рисунок 3.2 — Прототип інтерфейсу каталогу товарі А) Bootstrap; Б) Tailwind CSS

А)



Б)

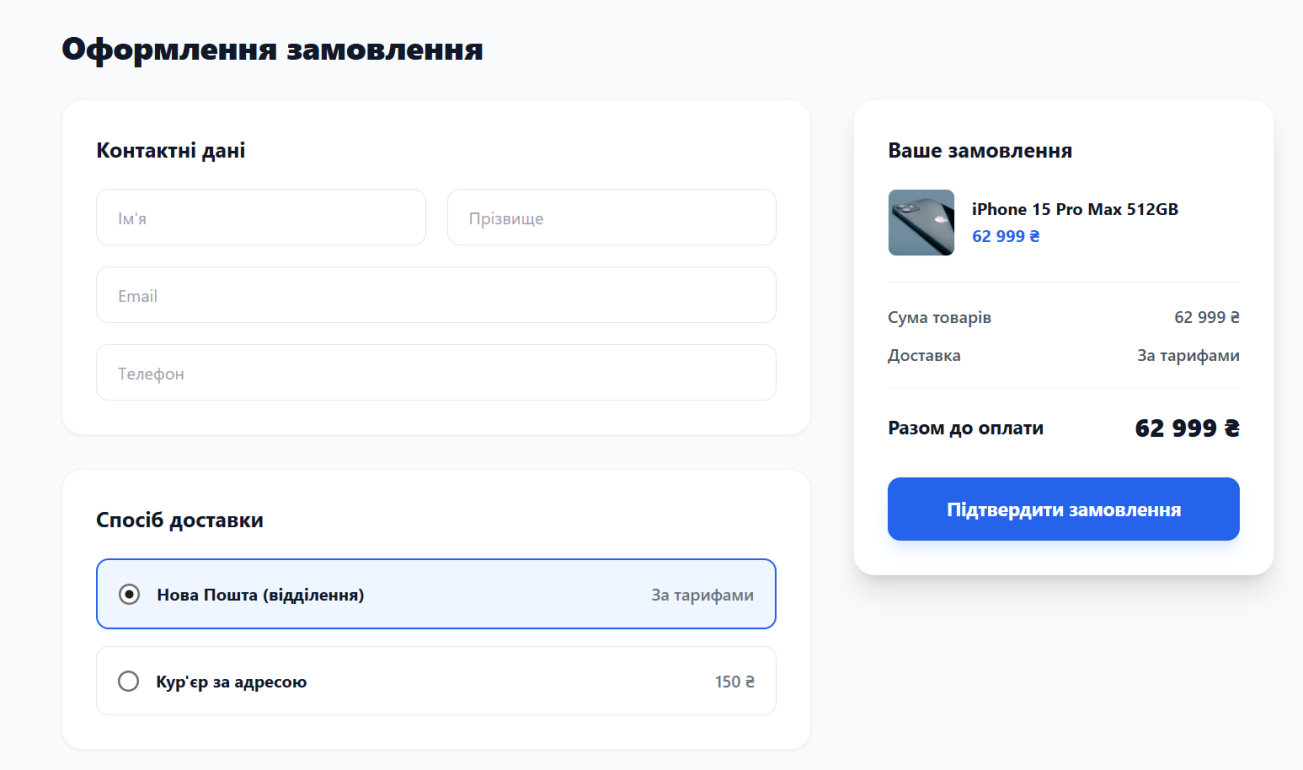


Рисунок 3.3 — Прототип інтерфейсу оформлення замовлення А) Bootstrap; Б) Tailwind CSS

3.2 Проведення вимірювань та аналіз метрик продуктивності

Для об'єктивного підтвердження теоретичних висновків та оцінки реальної ефективності досліджуваних архітектурних підходів було проведено серію інструментальних вимірювань за розширеною системою з шести метрик. Процес дослідження вимагав використання спеціалізованого програмного забезпечення та врахування особливостей компіляції сучасних CSS-фреймворків у різних середовищах. Методологія проведення вимірювань базувалася на використанні вбудованого інструментарію розробника Google Chrome DevTools (рис.3.4) та системи автоматизованого аудиту Google Lighthouse (рис. 3.5), що дозволило зафіксувати реальну вагу файлів і показники Core Web Vitals у відповідності до індустріальних стандартів.

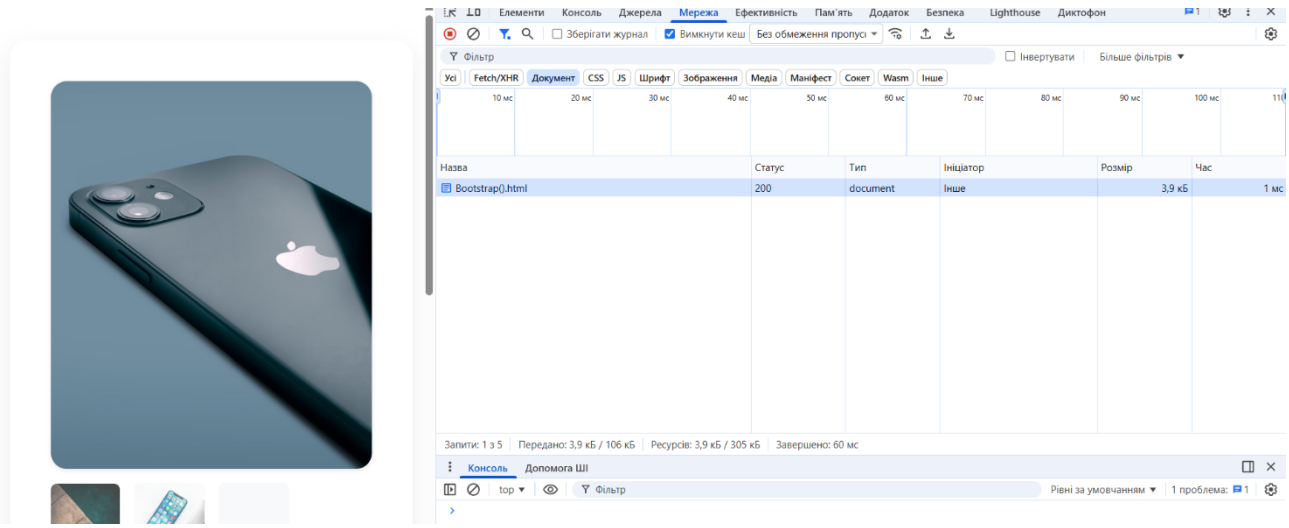


Рисунок 3.4 — Фіксація мережевого навантаження та обсягу розмітки засобами Chrome DevTools

Ефективність

Рисунок 3.5 — Звіт продуктивності інтерфейсу (компонентний підхід Bootstrap) за даними Google Lighthouse

Особливої уваги під час експерименту заслужив етап компіляції utility-first проєкту в хмарному середовищі розробки StackBlitz. Під час роботи бандлера Vite було зафіксовано аварійну зупинку процесу через брак оперативної пам'яті браузерного контейнера, що відображено на рисунку 3.6.

```
Terminal
❯ npm run build

> vite-starter@0.0.0 build
> vite build

vite v3.0.7 building for production...
transforming (1) index.htmlBrowserslist: caniuse-lite is outdated. Please run:
  npx update-browserslist-db@latest
  Why you should do it regularly: https://github.com/browserslist/update-db#readme
✓ 6 modules transformed.
[vite:css-post] The service was stopped
error during build:
Error: The service was stopped
    at eval (/home/projects/vite-vanilajs-tailwind/node_modules/esbuild/lib/main.js:1400:29)
    at eval (/home/projects/vite-vanilajs-tailwind/node_modules/esbuild/lib/main.js:678:9)
    at Socket.afterClose (/home/projects/vite-vanilajs-tailwind/node_modules/esbuild/lib/main.js:
656:7)
    at Socket.emit (node:events:30:11640)
    at endReadableNT (node:internal/streams/readable:244:10626)
    at processTicksAndRejections (node:internal/process/task_queues:206:1157)
```

Рисунок 3.6 — Аварійна зупинка процесу компіляції (брак пам'яті хмарного контейнера)

Цей інцидент було ідентифіковано як апаратне обмеження віртуалізованого середовища при виконанні інтенсивних операцій JIT-компіляції. З метою отримання еталонних даних без впливу системних похибок було прийнято інженерне рішення щодо використання ізольованого середовища Tailwind Play (рис. 3.7).

Рисунок 3.7 — Генерація фінального CSS-файлу за допомогою Tailwind Play

Це дозволило отримати максимально точні показники ваги мініфікованого бандла, сгенерованого офіційним рушієм фреймворку, що є критично важливим для наукової достовірності результатів .

Кількісна оцінка швидкості прототипування та кастомізації була реалізована шляхом декомпозиції цієї якісної метрики на три вимірювані параметри: чистий час розробки, кількість перемикань контексту та алгоритмічну складність процесу. Експериментально встановлено, що реалізація унікального дизайну в межах компонентного підходу вимагає значних часових витрат через необхідність постійного перемикання між файлами розмітки та зовнішніми таблицями стилів для написання перевизначаючих правил . Натомість utility-first архітектура забезпечує лінійний процес розробки в межах єдиного документа, що не лише скорочує час створення інтерфейсу, а й мінімізує когнітивне навантаження на розробника завдяки відсутності потреби у формуванні складної ієрархії кастомних селекторів . У таблиці 3.1 наведено порівняння метрик продуктивності компонентного та utility-first підходів .

Таблиця 3.1 – Порівняльний аналіз метрик продуктивності компонентного та utility-first підходів

Тип інтерфейсу	**Досліджувана метрика**	**Компонентний підхід (Bootstrap)**	**Utility-first підхід (Tailwind CSS)**
1. Сторінка товару (Складни UI)	Розмір CSS (мініфікований)	34 КБ (мережа) / ~170 КБ (розпакований)	2,68 КБ
	Розмір HTML-розмітк	3,9 КБ	3,2 КБ
	Сумарне мережеве навантаження	106КБ	221 КБ
	Продуктивність (Lighthouse)	98	98

Продовження таблиці 3.1

2. Каталог товарів (Повторювані елементи)	**Розмір CSS (мініфікований)**	**34 КБ (мережа) / ~170 КБ (розпакований)**	**2,51 КБ**
	Розмір HTML-розмітк	5,4 КБ	5,8 КБ
	Сумарне мережеве навантаження	140 КБ	256 КБ
	Продуктивність (Lighthouse)	98	94
3. Оформлення замовлення (Інтерактивні форми)	Розмір CSS (мініфікований)	34 КБ (мережа) / ~170 КБ (розпакований)	2,62 КБ
	Розмір HTML-розмітк	6,1 КБ	6,6 КБ
	Сумарне мережеве навантаження	45,2 КБ	160 КБ
	Продуктивність (Lighthouse)	91	96
Загальні показники	Кастомний CSS-код (рядки)	~ 140 рядків ручного коду	0 рядків
	Швидкість кастомізації	Низька (Context Switching)	Висока (Робота в 1 файлі)

Фіксація сумарного мережевого навантаження для фреймворку Tailwind CSS (221 Кб) проводилася в режимі розробки (Development Mode) через обмеження віртуального середовища. Цей обсяг включає повний JavaScript-компілятор, що працює через CDN. Як було доведено метрикою генерації фінального бандлу, у реальному продакшен-середовищі загальне мережеве

навантаження складатиме лише близько ~17-20 Кб (3,2 Кб розмітки плюс 14 Кб мініфікованих стилів), що є вагомим доказом ефективності utility-first підходу в контексті оптимізації мережевого навантаження, хоча й потребує комплексного порівняння з іншими критеріями для остаточного вибору.

Аналіз результатів інструментальних вимірювань, зведених у таблиці 3.1, дозволяє зробити комплексний висновок щодо ефективності застосування компонентного та utility-first підходів при проєктуванні інтерфейсів інформаційних систем.

Головною технічною перевагою архітектури Tailwind CSS, що емпірично підтверджується на всіх трьох типах досліджуваних сторінок, є радикальне зменшення обсягу каскадних таблиць стилів. Завдяки механізму JIT-компіляції, фреймворк генерує мініфікований CSS-файл розміром близько 2,5–2,6 Кб, що містить виключно задіяні у розмітці класи. Натомість компонентний підхід на базі Bootstrap вимагає завантаження монолітного ядра, яке навіть у стиснутому для передачі по мережі вигляді займає 34 Кб, а після розпакування браузером споживає близько 170 Кб оперативної пам'яті клієнтського пристрою незалежно від функціональної складності інтерфейсу.

Варто відзначити, що прогнозований недолік utility-first підходу у вигляді суттєвого збільшення обсягу HTML-розмітки (ефект «class soup») на практиці виявився некритичним. Згідно з отриманими даними, відхилення розміру HTML-документів між двома підходами становить у середньому менше 1 Кб (наприклад, 6,1 Кб проти 6,6 Кб для форми оформлення замовлення), що не має жодного негативного впливу на швидкість парсингу структури документа браузером.

У розрізі процесу розробки та супроводу програмного продукту utility-first архітектура продемонструвала вищу інженерну гнучкість. Досягнення повної візуальної відповідності унікальному макету за допомогою Bootstrap потребувало написання близько 140 рядків кастомного коду, що нівелює головну ідею використання готових компонентів та підвищує ризик виникнення конфліктів специфічності. Tailwind CSS дозволив реалізувати ідентичний

візуальний результат без використання зовнішніх CSS-правил, спираючись виключно на маніпуляції з утилітами. Цей фактор мінімізує когнітивне навантаження на розробника та повністю усуває проблему перемикання контексту (Context Switching) між робочими файлами.

Комплексні показники продуктивності за методологією Google Lighthouse для обох підходів знаходяться на високому рівні (понад 90 балів), проте рішення на базі Tailwind CSS стабільно демонструють маргінальну перевагу. Підсумовуючи математичні та інструментальні дані, можна стверджувати про доцільність використання utility-first підходу для проєктування сучасних масштабованих вебзастосунків, де ключовими інженерними вимогами є мінімізація мережевого навантаження, висока швидкість рендерингу та модульна чистота кодової бази.

3.3 Багатокритеріальна оцінка результатів експерименту.

Для об'єктивного та математично обґрунтованого вибору оптимального архітектурного підходу до стилізації інтерфейсів застосуємо метод узагальненого критерію, описаний формулою (2.2). Порівняння здійснюється між двома сформованими альтернативами: x_1(компонентний підхід на базі Bootstrap) та x_2 (utility-first підхід на базі Tailwind CSS).

На основі інструментальних вимірювань, зафіксованих у ході експерименту, було розраховано середні арифметичні значення технічних метрик для трьох типів еталонних сторінок. Для проведення розрахунків сформовано систему з чотирьох ключових критеріїв (k_1, k_2, k_3, k_4), призначено їхні вагові коефіцієнти (a_i), що відповідають специфіці високонавантажених e-commerce платформ, та виконано процедуру нормування початкових значень до безрозмірної шкали $k_i^н \epsilon [0; 1]$.

Зведені дані та результати проміжних розрахунків представлені у таблиці 3.2.

Таблиця 3.2 — Параметри багатокритеріального оцінювання альтернатив розробки UI

Критерій (k_i)	Назва критерію та одиниці виміру метрики	Вага (a_i)	Абсолютне значення (x_1, Bootstrap)	Абсолютне значення (x_2, Tailwind)	Нормоване значення (x_1, Bootstrap)	Нормоване значення (x_2, Tailwind)
k_1	Обсяг фінального CSS-файлу (Кб) →min	0,4	170	2,6	0	1
k_2	Розмір HTML-розмітки сторінки (Кб) →min	0,1	5,1	5,2	1	0
k_3	Обсяг кастомного CSS-коду (рядків) →min	0,2	140	0	0	1
k_4	Продуктивність Google Lighthouse (балів) →max	0,3	95	96	0	1
$\sum$	Умова нормування ваг за формулою (2.3)	1	—	—	—	—

Використовуючи визначені нормовані показники $k_i^{\text{н}}$ та вагові коефіцієнти a_i з таблиці 3.2, проведемо обчислення інтегрального значення узагальненого критерію для кожної архітектурної альтернативи за формулою (2.2):

$$P(x_1) = (0 * 0.4) + (1 * 0.1) + (0 * 0.2) + (0 * 0.3) = 0.1$$

$$P(x_2) = (1 * 0.4) + (0 * 0.1) + (1 * 0.2) + (1 * 0.3) = 0.9$$

Таким чином, математичні розрахунки на основі узагальненого адитивного критерію оптимізації чітко демонструють, що узагальнений показник ефективності для utility-first підходу ($P(x_2) = 0.9$) суттєво перевищує аналогічний показник для компонентного підходу ($P(x_1) = 0.1$).

Отриманий результат є суворим математичним підтвердженням доцільності та інженерної переваги вибору фреймворку Tailwind CSS для розробки інтерфейсних підсистем електронної комерції, де критично важливими

є мінімальна вага стилів, відсутність кастомного технічного боргу та максимальна швидкість відмальовування сторінок користувачеві.

ВИСНОВКИ

У магістерській кваліфікаційні роботі вирішено актуальне науково-практичне завдання, яке полягає в оптимізації процесу розробки клієнтської частини вебзастосунків для сфери електронної комерції шляхом обґрунтованого вибору архітектурного підходу до стилізації інтерфейсів. Проведений комплексний теоретичний та інструментальний аналіз дозволив сформувати низку ключових висновків.

Дослідження еволюції методологій проєктування інтерфейсів підтвердило, що класичний компонентний підхід, яскравим представником якого є фреймворк Bootstrap, забезпечує високу швидкість створення стандартизованих макетів, проте втрачає свою ефективність в умовах жорстких вимог до унікального брендингу. Практична реалізація експериментальних прототипів інтернет-магазину довела, що спроба кастомізації компонентної архітектури неминуче призводить до проблеми високої специфічності та фрагментації кодової бази. Для досягнення повної візуальної відповідності дизайн-макету розробнику довелося створити значний масив перевизначаючих CSS-правил, що структурно ускладнює подальшу підтримку інформаційної системи.

Натомість застосування utility-first підходу на базі Tailwind CSS продемонструвало принципово вищий рівень інженерної гнучкості. Перенесення процесу стилізації безпосередньо в HTML-розмітку шляхом використання низькорівневих утилітарних класів дозволило цілком відмовитися від написання зовнішніх каскадних таблиць стилів. Це не лише мінімізувало когнітивне навантаження на розробника через усунення необхідності перемикання контексту (Context Switching) між файлами, а й забезпечило лінійний та передбачуваний алгоритм створення унікальних інтерфейсних рішень.

Багатокритеріальне оцінювання досліджуваних підходів до стилізації інформаційних систем за допомогою інструментальних метрик продуктивності підтвердило архітектурні переваги utility-first підходу. Завдяки використанню алгоритмів JIT-компіляції вдалося досягти значного скорочення обсягу

фінального CSS-бандлу — з понад 170 КБ (базове ядро Bootstrap) до мікроскопічних 2,5–2,6 КБ. При цьому зафіксоване незначне збільшення обсягу HTML-розмітки виявилося статистично несуттєвим і не вплинуло на швидкість парсингу документа. Як наслідок, загальне мережеве навантаження на підсистему стилізації у продакшен-середовищі зменшилося в десятки разів, що є критично важливим фактором для систем електронної комерції, орієнтованих на користувачів із нестабільним мобільним зв'язком.

З позиції управління IT-проєктами, інтеграція Tailwind CSS у технологічний стек дозволяє оптимізувати життєвий цикл розробки програмного продукту та зменшити накопичення технічного боргу. Метрики інтегральної продуктивності за стандартами Core Web Vitals (Google Lighthouse), які стабільно наближаються до максимальних 100 балів, гарантують високу швидкість рендерингу сторінок. Це прямо корелює з покращенням користувацького досвіду (UX), зниженням показника відмов (Bounce Rate) та підвищенням ефективності пошукової оптимізації (SEO), що в кінцевому підсумку позитивно впливає на комерційні показники продукту.

Таким чином, результати дослідження дають повні підстави стверджувати, що використання архітектури Tailwind CSS є найбільш раціональним, масштабованим та технічно обґрунтованим рішенням для проєктування сучасних високонавантажених вебзастосунків, що робить його впровадження доцільним для широкого спектра комерційних IT-проєктів.

.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. ISO 9241-210:2019. Ергономіка взаємодії людина-система. Частина 210. Людино-орієнтоване проєктування інтерактивних систем [Електронний ресурс]. – Режим доступу: https://www.iso.org/standard/77520.html (дата звернення: 01.04.2026).

2. Duckett J. HTML and CSS: Design and Build Websites. – Indianapolis: Wiley, 2011. – 490 p.

3. Web Vitals [Електронний ресурс] / web.dev. – Режим доступу: https://web.dev/vitals/ (дата звернення: 01.04.2026).

4. BEM Methodology [Електронний ресурс]. – Режим доступу: https://en.bem.info/methodology/ (дата звернення: 01.04.2026).

5. Bootstrap Documentation [Електронний ресурс]. – Режим доступу: https://getbootstrap.com/docs/5.3/ (дата звернення: 01.04.2026).

6. Tailwind CSS Documentation [Електронний ресурс]. – Режим доступу: https://tailwindcss.com/docs/ (дата звернення: 01.04.2026).

7. Frain B. Responsive Web Design with HTML5 and CSS. – Packt Publishing, 2020. – 334 p.

8. Krug S. Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability. – New Riders, 2014. – 200 p.

9. W3C. Cascading Style Sheets (CSS) [Електронний ресурс]. – Режим доступу: https://www.w3.org/Style/CSS/ (дата звернення: 01.04.2026).

10. Marcotte E. Responsive Web Design. – New York: A Book Apart, 2011. – 143 p.

11. Sass: Syntactically Awesome Style Sheets [Електронний ресурс]. – Режим доступу: https://sass-lang.com/documentation (дата звернення: 01.04.2026).

12. Coyier C. A Complete Guide to Flexbox [Електронний ресурс] / CSS-Tricks. – Режим доступу: https://css-tricks.com/snippets/css/a-guide-to-flexbox/ (дата звернення: 01.04.2026).

13. MDN Web Docs. CSS: Cascading Style Sheets [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/CSS (дата звернення: 01.04.2026).

14. Osmani A. Learning JavaScript Design Patterns. – O'Reilly Media, 2012. – 254 p.

15. Node.js Documentation [Електронний ресурс]. – Режим доступу: https://nodejs.org/en/docs/ (дата звернення: 01.04.2026).

16. Vite. Next Generation Frontend Tooling [Електронний ресурс]. – Режим доступу: https://vitejs.dev/guide/ (дата звернення: 01.04.2026).

17. House C. A Complete Guide to Grid [Електронний ресурс] / CSS-Tricks. – Режим доступу: https://css-tricks.com/snippets/css/complete-guide-grid/ (дата звернення: 01.04.2026)

18. Foundation Documentation [Електронний ресурс] / ZURB. – Режим доступу: https://get.foundation/sites/docs/ (дата звернення: 01.04.2026)

19. Bulma: the modern CSS framework [Електронний ресурс]. – Режим доступу: https://bulma.io/documentation/ (дата звернення: 01.04.2026)

20. Петров Е. Г., Новожилова М. В., Гребеннік І. В. Методи і засоби прийняття рішень у соціально-економічних системах : навч. посіб. / за ред. Е. Г. Петрова. – Київ : Техніка, 2004. – 256 с.