Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

Бакалаврська кваліфікаційна робота на тему

«Інтерактивний обчислювач»

08-54.БКР.031.00.000 ПЗ

Виконав: студент 4 курсу, групи 1, 2 СП-21 б
спеціальності
123 Комп'ютерна інженерія,
(шифр і назва напряму підготовки, спеціальності)
освітня програма
«Системне програмування»
Лаврентюк І.О.
(прізвище та ініціали)

Керівник: к.пед.н., доц. каф. ОТ
Добровольська Н.В.
(прізвище та ініціали)

Рецензент: д.т.н., проф. каф. ПЗ
Ліщинська Л.Б.
(прізвище та ініціали)

ДОПУЩЕНО
Завідувач кафедри ОТ
д.т.н., проф. Азаров О. Д
_____
"___" _____2025 р.

Вінниця ВНТУ – 2025 рік

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки
Освітньо—кваліфікаційний рівень бакалвр
Галузь знань — 12 Інформаційні технології
Спеціальність — 123 Комп'ютерна інженерія
Освітня програма — Системне програмування

ЗАТВЕРДЖУЮ

Завідувач кафедри ОТ
д.т.н., проф._____Азаров О. Д.
«21» березня 2025 року

З А В Д А Н Н Я

НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Лаврентюку Івану Олександровчиу

1 Тема роботи: Інтерактивний обчислювач, керівник роботи к.пед.н., доц. каф. ОТ Добровольська Н.В, затверджено наказом вищого навчального закладу від «20» березня 2025 року №97.

2 Строк подання студентом роботи 13.05.2025

3 Вихідні дані до роботи: створення системи розкладу уроків з автоматичною оптимізацією

4 Зміст розрахунково-пояснювальної записки: вступ;опис архітектури логіки інтерактивного обчислювача: події, змінні, обчислення; обґрунтування проєктування алгоритмів системи розробка й тестування системи; висновки.

5 Перелік графічного матеріалу: системи розкладу уроків з автоматичною оптимізацією.

6 Консультанти розділів роботи приведені в таблиці 1.

7 Дата видачі завдання 21.03.2025 р.

8 Календарний план приведений в таблиці 2.

Таблиця 1 — Консультанти роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	Завдання прийняв
1-3	к.пед.н., доц. каф. ОТ Добровольська Н.В.	21.03.25	13.05.25
Нормоконтроль	ас. каф. ОТ Швець С. І.	14.05.25	30.05.25

Таблиця 2 — Календарний план

№ з/п	Назва етапів дипломної Роботи	Строк виконання	Примітка
1	Постановка задачі роботи	21.03.25	виконано
2	Пошук матеріалів та інформаційних джерел	21.03.25— 30.03.25	виконан
3	Аналіз існуючих прототипів та визначення вимог до створюваної системи	21.03.25— 30.03.25	виконано
4	Обґрунтування структури та принципів роботи системи	31.03.25— 13.04.25	виконано
5	Розробка й тестування апаратно-програмної системи	14.04.25— 27.04.25	виконан
7	Оформлення пояснювальної записки та ілюстративного матеріалу	28.04.25— 11.05.25	виконан
8	Перевірка якості виконання бакалаврської кваліфікаційної роботи та усунення недоліків	13.05.25	виконан

Студент Іван ЛАВРЕНТЮК

Керівник роботи к.пед.н., доц. Наталья ДОБРОВОЛЬ

АНОТАЦІЯ

УДК 621.3

Лаврентюк І.О. Веб-застосунок «Інтерактивний обчислювач». Бакалаврська кваліфікаційна робота зі спеціальності 123 «Комп'ютерна інженерія», освітня програма «Системне програмування». Вінниця: ВНТУ, 2025. 78с.

78 с. На укр. мові. Бібліогр.: 41 назв; рис.: 26; табл. 4.

Реалізовано та проаналізовано кілька підходів до побудови інтерфейсу та обчислювального ядра: базова реалізація на основі бібліотеки Math.js, розширення функціональності через введення підтримки наукових функцій, а також комбінована стратегія інтерактивної взаємодії з користувачем, що поєднує зручність використання з високою точністю обчислень. Система реалізована з використанням сучасних вебтехнологій: Node.js, React/Next.js, бібліотек Math.js і Decimal.js, що забезпечують адаптивність інтерфейсу, високу продуктивність та можливість масштабування. Проєктна архітектура передбачає подальший розвиток — зокрема, розширення функціональності, локалізацію, а також інтеграцію з мобільними додатками або освітніми платформами.

Отримані результати підтвердили практичну придатність інтерактивного калькулятора до щоденного використання, його стабільність, зручність і високу якість користувацького досвіду. Робота може слугувати основою для подальших досліджень у сфері інтерактивних обчислень, UI/UX-дизайну та веброзробки.

Ключові слова: інтерактивний калькулятор, вебзастосунок, обчислення, користувацький інтерфейс, JavaScript, React, Next.js, Math.js, UX-дизайн.

ABSTRACT

UDC 621.3

Lavrentiuk I.O. Web application "Interactive calculator". Bachelor's qualification work in specialty 123 "Computer engineering", educational program "System programming". Vinnytsia: VNTU, 2025. 78p.

Several approaches to building the interface and computing core have been implemented and analyzed: a basic implementation based on the Math.js library, expanding functionality by introducing support for scientific functions, and a combined strategy for interactive user interaction that combines ease of use with high accuracy of calculations. The system is implemented using modern web technologies: Node.js, React/Next.js, Math.js, and Decimal.js libraries, which provide interface adaptability, high performance, and scalability. The project architecture provides for further development - in particular, expanding functionality, localization, and integration with mobile applications or educational platforms.

The results confirmed the practical suitability of the interactive calculator for daily use, its stability, convenience and high quality of user experience. The work can serve as a basis for further research in the field of interactive computing, UI/UX design and web development.

Keywords: interactive calculator, web application, computing, user interface, JavaScript, React, Next.js, Math.js, UX design.

ЗМІСТ

ВСТУП……....3
1 **ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ**..... 5
1.1 Огляд потреб та застосування інтерактивного обчислювача у веб-середовищі……..5
1.2 Формулювання цілей та функціональних вимог 17
1.3 Дизайн та UX/UI особливості інтерактивного обчислювача…….....19
1.4 Огляд існуючих рішень та типових реалізацій інтерактивних обчислювачів у веб середовищі…….....21
2 **АНАЛІЗ ЗАСОБІВ РОЗРОБКИ ІНТЕРАКТИВНОГО ОБЧИСЛЮВАЧА** …….25
2.1 Вибір мов програмування та середовищ розробки..... 25
2.2 Опис архітектури логіки інтерактивного обчислювача: події, змінні, обчислення …… 29
2.3 Інтерфейс користувача та технічна реалізація..... 35
3 **РОЗРОБКА ПРОЄКТУ ТА ТЕСТУВАННЯ** ……..……..39
3.1 Реалізація математичних операцій та підтримка розширених функцій……..…..…….....…39
3.2 Тестування інтерфейсу користувача в умовах реального використання…...55
3.3 Розширення функціональності: теми, пам’ять, історія обчислень……58
ВИСНОВКИ..... 64
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ..... 66
ДОДАТОК А Технічне завдання..... 69
ДОДАТОК Б Протокол перевірки навчальної (кваліфікаційної) роботи 72
ДОДАТОК В Графічна частина..... 73
ДОДАТОК Г Ілюстративна частина 76
ДОДАТОК Д Лістинг програмного забезпечення..... 78

ВСТУП

У сучасному інформаційному суспільстві широке впровадження цифрових технологій зумовлює необхідність створення ефективних, зручних і доступних засобів для обробки інформації. Одним із важливих напрямів у цій галузі є розробка веборієнтованих програмних засобів, які забезпечують інтерактивність, адаптивність до пристроїв різного типу та оперативність у використанні. Особливу нішу серед таких інструментів займають інтерактивні обчислювачі (калькулятори), які є універсальними рішеннями для виконання широкого спектра обчислень — від базових арифметичних до складних наукових і фінансових операцій.

На фоні зростання попиту на онлайн-інструменти, що не потребують встановлення додаткового програмного забезпечення, актуальною є задача створення вебзастосунку, який би поєднував гнучкість функціональності, інтуїтивний інтерфейс, кросплатформеність і швидкодію. При цьому важливими є не лише технічні характеристики реалізації, а й зручність використання (UX), адаптивність до потреб користувача, розширення можливостей для збереження історії обчислень, персоналізація інтерфейсу тощо.

Актуальність теми обумовлена необхідністю забезпечення користувачів високоякісним інструментом для проведення обчислень у браузерному середовищі, що відповідає сучасним стандартам UI/UX-дизайну, є технічно надійним і відкритим до масштабування.

Метою кваліфікаційної роботи є розробка інтерактивного вебобчислювача з підтримкою розширеної математичної функціональності, реалізацією адаптивного інтерфейсу користувача та системи збереження історії обчислень і пам’яті.

Для досягнення поставленої мети потрібно вирішити такі **завдання:**

—проаналізувати сучасні інструменти та технології для створення вебкалькуляторів;

—визначити оптимальні бібліотеки та фреймворки для реалізації обчислювальної логіки;

—спроєктувати структуру інтерфейсу, дотримуючись принципів доступності та адаптивності;

—реалізувати функціональність базових та наукових обчислень, а також підтримку пам’яті та історії;

—забезпечити тестування системи та проаналізувати результати взаємодії користувачів із застосунком;

— провести оптимізацію продуктивності та покращення UX/UI на основі отриманих результатів.

Об’єктом дослідження є процес розробки та оптимізації інтерфейсних і обчислювальних рішень у веборієнтованих програмних застосунках.

Предметом дослідження є архітектура, функціональність, користувацький інтерфейс та механізми інтерактивної взаємодії у процесі реалізації вебобчислювача.

Практична цінність розробленого застосунку полягає в можливості його використання як у повсякденному житті, так і в навчальних чи професійних цілях. Створена система може бути розширена для спеціалізованих галузей, зокрема фінансів, інженерії, статистики або наукових розрахунків. Відкритість архітектури, зручність користування та адаптивність дозволяють інтегрувати обчислювач у більші системи або використовувати його як окремий модуль.

1 **АНАЛІЗ СУЧАСНИХ ІНСТРУМЕНТІВ ДЛЯ РОЗРОБКИ ІНТЕРАКТИВНОГО ОБЧИСЛЮВАЧА**

1.1 Огляд потреб та застосування інтерактивного обчислювача у веб-середовищі

Інтерактивний обчислювач, або ж онлайн калькулятор, - це програмний засіб, що функціонує у веб-середовищі та забезпечує користувачів інструментами для виконання різноманітних математичних обчислень без потреби встановлення локального програмного забезпечення. Його основна перевага полягає у доступності: достатньо лише підключення до Інтернету та браузера, щоб мати змогу миттєво скористатися повнофункціональним калькулятором з будь-якого пристрою.

Сучасні веб-калькулятори можуть виконувати широкий спектр обчислень. Базові калькулятори реалізують елементарні операції, такі як додавання, віднімання, множення, ділення, обчислення відсотків та змінних у виразах. Наукові калькулятори мають значно ширший функціонал: вони підтримують обчислення тригонометричних функцій, логарифмів, ступенів, факторіалів, а також роботу з математичними виразами, що містять дужки. Окрему нішу займають фінансові калькулятори, які орієнтовані на виконання обчислень у сфері економіки та бізнесу: наприклад, визначення ефективної процентної ставки, розрахунок амортизації,ануїтетів або кредитних платежів.

Аудиторія інтерактивних калькуляторів є досить широкою. Вона охоплює школярів і студентів, які використовують їх під час навчального процесу; викладачів, що демонструють функціональні залежності та графіки; інженерів, які потребують точних і швидких розрахунків під час моделювання технічних процесів; фінансистів, бухгалтерів та економістів, які розраховують податки, інвестиції чи прибутки; а також звичайних користувачів, яким необхідно здійснювати повсякденні обчислення в зручному інтерфейсі без зайвого програмного коду або налаштувань.

Порівняно з традиційними офлайн-програмами, веб-обчислювачі мають низку важливих переваг. Вони не залежать від платформи користувача —

працюють як на комп’ютерах з Windows або Linux, так і на мобільних пристроях із iOS чи Android. Вони не потребують встановлення або оновлення, оскільки вся логіка виконується на стороні клієнта або сервера в реальному часі. Крім того, інтерактивні калькулятори можуть мати кастомізовану структуру, підтримку тем оформлення, багатомовність та функції збереження історії розрахунків.

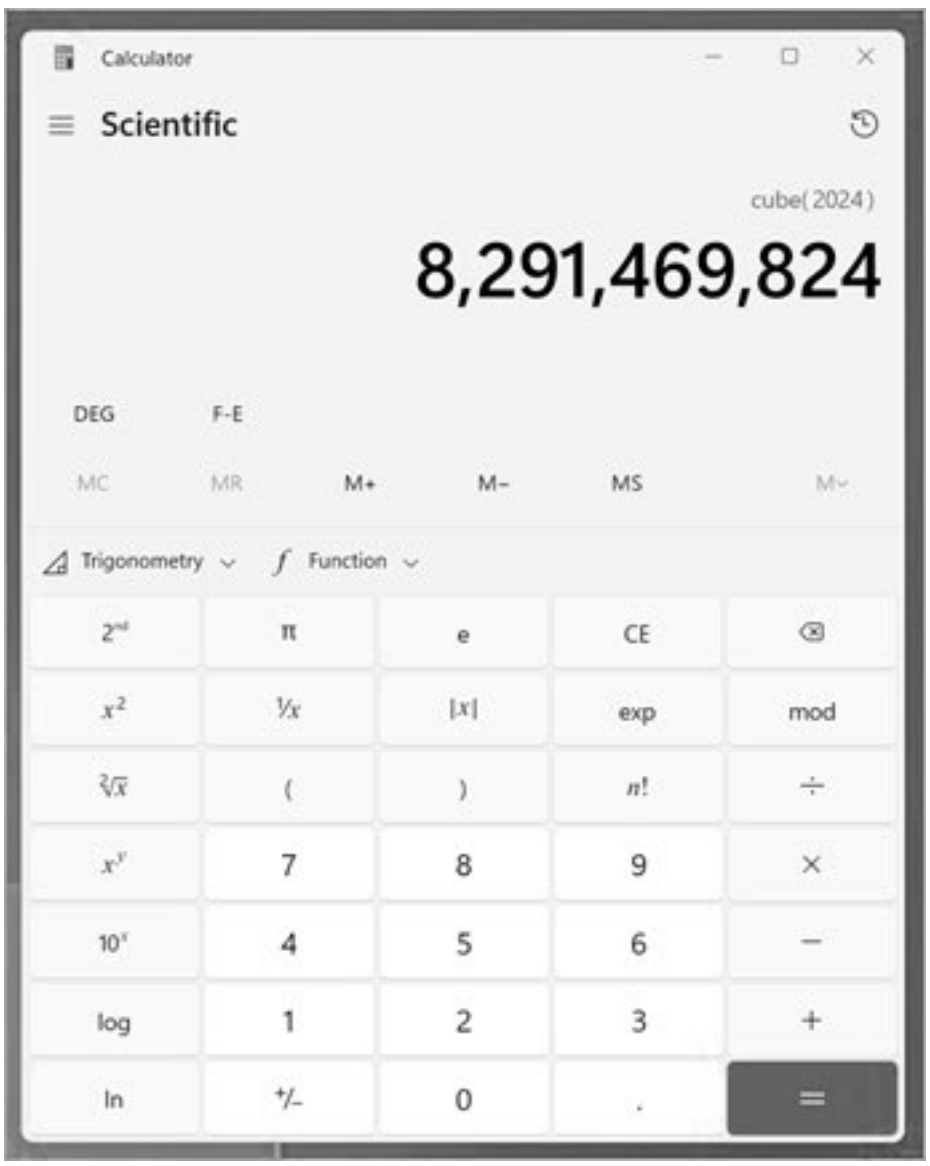

Рисунок 1.1 — Демонстрація застосунку Microsoft calculator

Microsoft Calculator — це офіційний додаток операційної системи Windows, призначений для виконання широкого спектру математичних обчислень. У режимі Scientific (науковий калькулятор) програма надає розширені можливості для студентів, інженерів, науковців та аналітиків.

Інтерфейс поєднує функціональність класичних інженерних калькуляторів із зручністю цифрової платформи, підтримуючи як стандартні арифметичні операції, так і спеціалізовані математичні функції.

У верхній частині вікна відображається введений вираз та його результат. У наведеному прикладі здійснюється обчислення куба числа 2024 — cube(2024), результат якого становить 8,291,469,824. Це вказує на підтримку математичних функцій вищих порядків із миттєвим виведенням значення у великому шрифті.

Програма підтримує режими роботи з градусами (DEG), експоненціальне відображення чисел (F–E), а також операції з пам’яттю: збереження (MS), додавання (M+), віднімання (M−), зчитування (MR) та очищення (MC). Це дозволяє виконувати багатокрокові обчислення із проміжними результатами.

Основна панель складається з двох розкривних меню — *Trigonometry* (для вибору тригонометричних функцій: sin, cos, tan тощо) і *Function*, де зібрано спеціальні математичні оператори та функції, серед яких:

— степеневі операції: x^2, x^3, x^y;

—кореневі функції: $\sqrt{}$, $\sqrt[3]{}$, $\sqrt[n]{x}$;

—константи: π, e;

—логарифмічні функції: log, ln;

—факторіал: n!;

—абсолютне значення: |x|;

—експоненціальне зростання: exp;

—модульне ділення: mod.

Інтерфейс підтримує введення як мишею, так і з клавіатури, забезпечуючи швидку взаємодію. Кнопки мають чітке маркування, достатній розмір для натискання та логічне групування (арифметичні, спеціальні, числові).

Програма підтримує темну та світлу тему оформлення, адаптується до розмірів вікна, дозволяє масштабувати шрифт і розташування елементів

залежно від екрана користувача. Крім того, калькулятор інтегрується з системними можливостями Windows — наприклад, може запускатися через голосового помічника або контекстне меню.

Microsoft Calculator є відкритим проєктом (вихідний код доступний на GitHub), що робить його зручним прикладом для вивчення реалізації складних обчислень у середовищі C++/UWP. У поєднанні з простим та інтуїтивно зрозумілим інтерфейсом він є еталонним прикладом багатофункціонального наукового калькулятора, орієнтованого на широку аудиторію.

Науковий калькулятор Microsoft Calculator є наочним прикладом практичної реалізації досягнень у галузі комп'ютерних технологій, зокрема в напрямку інженерного програмування, інтерфейсної взаємодії та вбудованих обчислювальних систем. Це програмне забезпечення інтегрує низку сучасних технічних рішень — від точного числового обчислення до адаптивного дизайну інтерфейсу — і демонструє приклад високоякісної реалізації математичного ПЗ на платформі Windows.

Розробка такого застосунку вимагає знань і навичок із різних розділів комп'ютерних технологій, зокрема:

—алгоритмізація та обчислювальні методи — для реалізації логіки обробки виразів, математичних функцій, логарифмів, степеневих операцій та обробки винятків;

—архітектура програмного забезпечення — для побудови модульної структури, керування пам'яттю, обробки подій;

—користувацький інтерфейс (ui/ux) — для створення інтуїтивно зрозумілого й доступного інтерфейсу, зокрема в середовищах uwp (universal windows platform);

—адаптивні системи — забезпечення гнучкості інтерфейсу під різні розміри екрана, підтримка тем оформлення та інтеграція з голосовими або системними службами ОС.

Калькулятор реалізує принципи людиноорієнтованого програмного забезпечення, які є основою напряму комп'ютерні технології. Він враховує

потреби користувача у точності, швидкості, зручності й наочності. Математичні функції реалізовані з урахуванням IEEE-стандартів, а числові методи — з мінімізацією похибок.

Ще одним прикладом є Calculator.net який пропонує величезну кількість вузькоспеціалізованих калькуляторів: від обчислення індексу маси тіла (ІМТ) до визначення днів між двома датами чи складних процентних розрахунків. Його перевага — простота інтерфейсу та гнучка категоризація за тематиками.

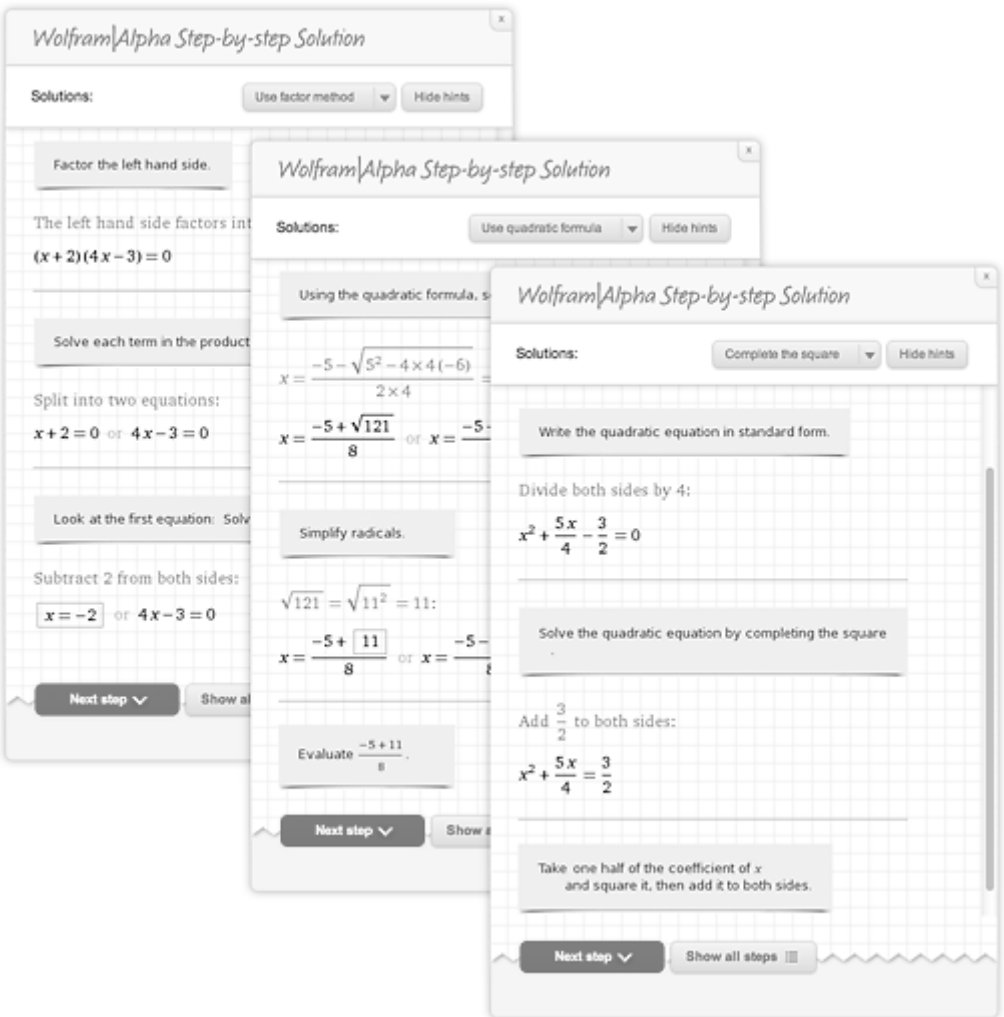

Рисунок1.2 — Демонстрація застосунку WolframAlpha

Найбільш інтелектуально насиченим є сервіс WolframAlpha, що базується на технологіях семантичного пошуку та обробки знань. Він дозволяє не лише виконувати обчислення, але й аналізувати математичні вирази, будувати графіки, розв'язувати рівняння, системи рівнянь, проводити статистичний аналіз. Крім математики, WolframAlpha охоплює фізику, хімію,

астрономію, економіку, навіть лінгвістику - усе це з автоматичним поясненням кожного кроку розрахунку.

Підсумовуючи, інтерактивні обчислювачі стали важливим елементом цифрового інструментарію, особливо у сфері освіти, науки й технологій. Їхня універсальність, адаптивність, доступність і простота використання роблять їх незамінними помічниками як для фахівців, так і для звичайних користувачів. Саме тому розробка сучасного веб-орієнтованого обчислювача, який би поєднував у собі гнучкий функціонал, інтуїтивний інтерфейс та підтримку наукових функцій, є актуальним і суспільно значущим завданням.

У процесі створення інтерактивного обчислювача важливим етапом є вибір мови програмування, яка забезпечить ефективну реалізацію як користувацького інтерфейсу, так і обчислювальної логіки. Веб-технології сьогодні пропонують багато варіантів, однак не всі мови однаково підходять для клієнтської реалізації обчислювальних задач.

Microsoft Calculator у науковому режимі є прикладом застосування комп’ютерних технологій для реалізації точних обчислювальних процесів, побудови зручного інтерфейсу та оптимізації взаємодії з користувачем. Програма поєднує алгоритмічну обробку виразів, модульну архітектуру, підтримку інженерних функцій і адаптивний дизайн, демонструючи практичне впровадження знань у сфері програмування, обчислювальної математики та UX/UI-дизайну.

Найбільш поширеною та природною мовою для створення веб-калькуляторів є JavaScript. Ця мова була створена у 1995 році як інструмент для динамічного оновлення HTML-сторінок і швидко стала основою фронтенд-розробки. JavaScript працює напряму в браузері, що дозволяє користувачеві взаємодіяти з калькулятором у реальному часі без перезавантаження сторінки або звернення до сервера. Серед його ключових переваг - висока швидкодія при обробці подій, гнучкість, наявність великої кількості спеціалізованих бібліотек для обчислень. Наприклад, бібліотека Math.js дозволяє обробляти складні вирази, працювати з матрицями, статистикою, одиницями виміру та навіть символічною алгеброю [1].

Бібліотека expr-eval дозволяє швидко парсити та обчислювати вирази, введені користувачем, що є надзвичайно корисним при створенні кастомного синтаксису [2].

Варто зазначити, що більшість популярних онлайн калькуляторів створено саме за допомогою JavaScript. Наприклад, Desmos активно використовує JavaScript та Canvas API для рендерингу графіків у реальному часі [3]. Інтерфейс Calculator.net побудований на HTML, CSS і JavaScript, де кожна кнопка активує відповідну функцію обробника подій, який виконує обчислення у браузері [4].

Microsoft Calculator у вебреалізаціях на JavaScript демонструє застосування комп'ютерних технологій для створення клієнтських обчислювальних систем. За допомогою JavaScript реалізується обробка введених математичних виразів, парсинг, розв'язання рівнянь, а також динамічна взаємодія з користувачем у реальному часі. Використання бібліотек (наприклад, Math.js) дозволяє забезпечити точність обчислень, а завдяки DOM-маніпуляціям та подієво-орієнтованій моделі програмування — реалізувати адаптивний і зручний інтерфейс у браузері.

Альтернативним рішенням, особливо у випадку створення складних обчислювальних моделей, є Python. З 2000-х років Python здобув визнання у науковому середовищі завдяки простому синтаксису та потужним бібліотекам - NumPy, SciPy, SymPy. Ці бібліотеки дозволяють реалізувати обчислення будь-якої складності: від простих арифметичних операцій до лінійної алгебри, чисельного інтегрування та символьних перетворень [6].

Проте основним недоліком Python є те, що він не виконується у браузері напряму. Щоб використати Python для калькулятора, розробники зазвичай створюють серверне API, яке отримує математичний вираз від клієнта, обробляє його на сервері та повертає результат. Це ускладнює архітектуру, але дозволяє розширити функціональність, наприклад, за рахунок збереження історії обчислень або побудови складних аналітичних моделей.

На стику JavaScript і Python з'являються проміжні рішення. Наприклад, Brython або Pyodide дозволяють запускати Python-код безпосередньо в

браузері через компіляцію до WebAssembly [8]. Вони цікаві з наукової точки зору, але поки що мають обмежену стабільність і продуктивність.

Ще однією перспективною мовою є TypeScript — надбудова над JavaScript, яка вводить статичну типізацію. Вона дозволяє зменшити кількість помилок під час компіляції, краще документувати код, а також забезпечити високий рівень читабельності [7]. Особливо актуальним є TypeScript у командних проєктах та при масштабуванні проєкту, коли структура логіки ускладнюється.

Для повноти огляду можна згадати C# та Java. Ці мови є потужними інструментами серверної розробки, однак у веб-середовищі вони менш зручні. C# може бути використаний з фреймворком Blazor WebAssembly, який дозволяє запускати C#-код у браузері, однак цей підхід ще не отримав широкого поширення у сфері веб-калькуляторів через складність налаштування, великий розмір бандлу та обмежену продуктивність [9]. Java, у свою чергу, застосовується переважно в Android-розробці та серверних додатках, і не має прямої підтримки у браузері без додаткових плагінів (які нині вже не підтримуються сучасними браузерами). У середовищі Java калькулятори реалізуються як приклад застосування комп’ютерних технологій у розробці десктопного програмного забезпечення з графічним інтерфейсом. За допомогою Java Swing або JavaFX створюється візуальна частина — кнопки, поле введення, дисплей результату. Логіка обробки виразів реалізується через об’єктно-орієнтоване програмування, з використанням власних класів для парсингу, обчислення та обробки винятків. Такі застосунки демонструють практичне використання Java у побудові модульних, переносимих та стабільних обчислювальних систем. Також варто зазначити, що обидві мови мають розвинені екосистеми з великою кількістю бібліотек та інструментів для розробки, що робить їх придатними для реалізації складних проєктів. Проте для простих веб-калькуляторів зазвичай обирають легші у використанні технології такі як JavaScript через кращу інтеграцію з браузером. Переглянемо таблицю 1.3.

Таблиця 1.3 — Порівння мов програмування

Мова програмування	Підтримка у браузері	Типізація	Основне застосування	Переваги	Недоліки
JavaScript	Пряма	Динамічна	Клієнтська частина (frontend)	Широка підтримка; численні бібліотеки (Math.js, expr-eval); працює безпосередньо у браузері	Немає статичної типізації; помилки можливі на етапі виконання
TypeScript	Після компіляції	Статична	Клієнтська частина, великі проєкти	Покращена підтримка масштабування, автодоповнення, менше помилок	Потребує компіляції в JavaScript
Python	Через сервер або WebAssembly	Динамічна	Серверна логіка, обробка даних	Потужні бібліотеки для обчислень (NumPy, SymPy, SciPy); читабельний синтаксис	Не працює нативно у браузері; потребує додаткових технологій (API, Pyodide)
C#	Через Blazor WebAssembly	Статична	Web + серверна розробка	Можливість виконання C# у браузері; інтеграція з .NET	Високе навантаження на систему; не завжди стабільна підтримка у браузерах
Java	Відсутня (напряму)	Статична	Серверна, Android-розробка	Висока продуктивність у бекенді	Веб-плагіни Java більше не підтримуються у сучасних браузерах

Таким чином, аналіз свідчить про те, що JavaScript залишається найкращим вибором для створення інтерактивного калькулятора у веб-середовищі. Його універсальність, сумісність з браузерами, велика спільнота та підтримка численних бібліотек забезпечують найкращу гнучкість та швидкість розробки. У випадку проєктів, що потребують масштабування або кращої структури коду, доцільним є використання TypeScript. Для задач зі складними обчисленнями та потребою у серверній логіці, Python може виступати як потужна альтернатива у бекенді [10].

Ефективність реалізації інтерактивного обчислювача у веб-середовищі

значною мірою залежить від вибору відповідних інструментів - бібліотек і фреймворків, які спрощують процес розробки, забезпечують стабільну роботу з математичними виразами та створюють зручний користувацький інтерфейс.

На рівні логіки обчислень ключову роль відіграють математичні бібліотеки. Однією з найпотужніших у цій сфері є Math.js - багатофункціональна бібліотека з відкритим вихідним кодом, що підтримує арифметичні, тригонометричні, статистичні операції, роботу з матрицями, комплексними числами, одиницями виміру, а також обчислення у символьному режимі [1]. Math.js дозволяє не лише виконувати обчислення, але й парсити складні вирази, що дає змогу будувати гнучку систему, яка інтерпретує введені користувачем формули. Наприклад, вираз 2 * (3 + sqrt(16)) може бути оброблений та обчислений без необхідності створення власного парсера. Також Math.js підтримує роботу зі змінними, функціями та навіть кастомним середовищем виконання, що дозволяє створювати розширені наукові калькулятори або навчальні інструменти з автоматичною перевіркою обчислень.

Іншою популярною бібліотекою є expr-eval - легка JavaScript-бібліотека, яка дозволяє парсити та виконувати математичні вирази у вигляді рядків [2]. Вона підходить для реалізації калькуляторів із кастомним синтаксисом або підтримкою змінних (наприклад, a = 5; b = a * 2). Основна перевага expr-eval полягає в її мінімалізмі, швидкодії та легкості у використанні. Проте вона менш гнучка, ніж Math.js, і не підтримує обчислення з одиницями виміру чи комплексними числами. Тим не менш, для більшості базових або середньої складності калькуляторів expr-eval є ідеальним рішенням через свою простоту інтеграції в JavaScript-код без потреби в додатковій конфігурації.

Ще однією корисною бібліотекою є Decimal.js, яка дозволяє працювати з великими та точними числами, усуваючи проблеми, пов’язані з втратами точності у стандартному Number-типі JavaScript [3]. Це особливо актуально у фінансових або інженерних обчисленнях, де навіть незначна похибка може призвести до помилкових результатів. Наприклад, обчислення 0.1 + 0.2 у JavaScript дає результат 0.30000000000000004, що є неприйнятним для систем

обліку. Decimal.js забезпечує арифметику з фіксованою точністю, що дозволяє уникнути подібних проблем. Ця бібліотека активно використовується в платіжних системах, фінансових калькуляторах, облікових сервісах, де навіть одна зайва десяткова частина може мати критичне значення.

Таблиця 1.4 — Інструменти

Інструмент	Тип	Призначення	Основні переваги	Обмеження
Math.js	Бібліотека	Математичні обчислення	Широкий функціонал, парсинг виразів, робота з матрицями	Великий розмір, зайвий функціонал для простих задач
expr-eval	Бібліотека	Парсинг та обчислення виразів	Легкість, швидкодія, простий API	Обмежена функціональність
Decimal.js	Бібліотека	Точні обчислення з десятковими числами	Усунення похибок, важливо для фінансових розрахунків	Немає парсингу виразів, лише арифметика

Окрім математичних бібліотек, для побудови інтерфейсу обчислювача часто використовуються фреймворки JavaScript, які забезпечують структуровану архітектуру, керування станом і компонентний підхід. Найбільш поширеним є React — бібліотека для створення інтерфейсів користувача, розроблена компанією Meta (раніше Facebook). React дозволяє будувати калькулятор як набір незалежних компонентів (дисплей, кнопки, панель пам’яті), які взаємодіють між собою через пропси і локальний або глобальний стан (State, Context API, Redux) [4]. Це дозволяє уникнути дублювання коду, полегшує масштабування, рефакторинг і тестування інтерфейсу. У реальних проєктах калькулятори на React часто включають підтримку тем оформлення, анімації, збереження історії введення у LocalStorage або IndexedDB.

Ще один популярний фреймворк — Vue.js, який завдяки двобічній прив’язці даних (two-way binding) спрощує синхронізацію інтерфейсу з внутрішнім станом калькулятора [5].

Таблиця 1.5 — Порівння мов програмування

Інструмент	Тип	Призначення	Основні переваги	Обмеження
React	Фреймворк	Побудова інтерфейсу на основі компонентів	Потужний екосистемний підхід, велика спільнота	Потребує налаштування збірки
Vue.js	Фреймворк	UI з двобічною прив’язкою	Простота використання, компактність	Менш потужний для великих систем
Svelte	Фреймворк	Компіляція в нативний JS	Висока продуктивність, мінімальний об’єм коду	Менш поширений, новий підхід

На відміну від React, Vue використовує більш декларативний синтаксис шаблонів, що робить його простішим у використанні, особливо для початківців. Vue також має легку структуру, меншу вагу та інтуїтивну документацію. Це робить його ідеальним для створення простих або середніх за складністю калькуляторів, а також для швидкої розробки прототипів.

Також варто згадати Svelte - сучасний фреймворк, який відрізняється тим, що компілює компоненти в чистий JavaScript ще до виконання в браузері [6]. Це дозволяє зменшити розмір коду, прискорити завантаження сторінки та знизити навантаження на браузер. У відміну від React і Vue, де більша частина логіки виконується під час рендерингу у браузері, Svelte генерує ефективний JavaScript-код на етапі компіляції. Це робить його дуже швидким і легким рішенням для побудови високопродуктивних SPA-додатків. Калькулятори, створені на Svelte, зазвичай мають мінімальний об’єм JavaScript-коду, високу швидкодію та простий розгортання.

Для систем з великою кількістю інтерактивних компонентів та багатофункціональним інтерфейсом доцільно поєднувати математичні бібліотеки з фреймворками. Наприклад, Math.js може використовуватись для обробки виразів, а React - для створення динамічного інтерфейсу. У результаті розробник отримує масштабовану, надійну та зручну систему з широкими можливостями для розширення в майбутньому (наприклад, додавання нових вкладок, наукових функцій, історії обчислень або режиму програмування).

1.2 Формулювання цілей та функціональних вимог

Ефективна розробка програмного забезпечення починається з етапу аналізу потреб та формування чітко структурованих вимог. Цей етап має ключове значення, оскільки саме тут визначаються не лише функціональні межі майбутньої системи, а й очікування кінцевих користувачів, контекст застосування та технічні обмеження. У випадку створення інтерактивного обчислювача, який реалізується як веб-застосунок, критично важливо приділити увагу як технічним, так і гуманітарним аспектам — ергономіці, доступності, простоті використання та ефективності.

Загальна мета розробки полягає у створенні сучасного, адаптивного, інтерактивного веб-додатку, який дозволяє користувачам зручно, швидко та точно виконувати математичні обчислення в режимі реального часу. При цьому застосунок повинен бути незалежним від операційної системи, не вимагати встановлення, працювати у звичайному веб-браузері та забезпечувати коректну роботу як на персональних комп’ютерах, так і на мобільних пристроях. Підхід до реалізації має базуватись на сучасному стеку веб-технологій: HTML5, CSS3 та JavaScript із використанням відповідних бібліотек і фреймворків.

Важливо враховувати також, що обчислювач не є лише технічним інструментом — це сервіс, який повинен вирішувати практичні задачі конкретних людей. Отже, аналіз цільової аудиторії відіграє ключову роль. Передбачається, що серед користувачів будуть школярі та студенти, яким калькулятор допоможе у вивченні математики, фізики, хімії; викладачі, які використовуватимуть його як інтерактивний демонстраційний інструмент на заняттях; інженери, що працюють із формулами та складними виразами; економісти та фінансисти, які потребують точної обробки десяткових значень. Крім того, важливою є категорія звичайних користувачів, які виконують побутові розрахунки (відсотки, знижки, об’ємні значення тощо).

Функціонально система повинна реалізовувати широкі обчислювальні можливості. Базовий набір операцій включає додавання, віднімання,

множення, ділення, а також обчислення з відсотками, коренями та степенями. Особливу цінність становить підтримка складених виразів з дужками, де важливо зберігати правильну послідовність виконання операцій згідно з математичними правилами. Ускладнення реалізації викликає також необхідність динамічного оновлення результату під час введення виразу, що потребує грамотної організації подій і логіки парсингу.

Оскільки обчислювач позиціонується як інструмент, що виходить за межі базового, доцільно передбачити реалізацію наукових функцій. Серед них: тригонометричні (sin, cos, tan), обернені тригонометричні, логарифмічні (log, ln), експоненційні, а також обробка факторіалів і побудова функцій (опційно - у вигляді розширення). Цей функціонал особливо важливий для користувачів технічного або природничо-наукового профілю.

З технічної точки зору, однією з важливих особливостей майбутньої системи є забезпечення високої точності обчислень. Мова JavaScript, хоч і широко підтримується в браузерах, має вбудовані обмеження при виконанні операцій із числами з плаваючою комою. Це може призводити до похибок, особливо у фінансових розрахунках (наприклад, результат 0.1 + 0.2 дорівнює 0.30000000000000004). Для вирішення цієї проблеми доцільно використовувати бібліотеку Decimal.js, яка реалізує високоточну арифметику з фіксованою точкою.

Інтерфейс користувача повинен бути зрозумілим з першого погляду та адаптованим до будь-якого типу пристрою. Кнопки, дисплей, панель результатів мають мати достатній розмір, бути контрастними та функціонально розділеними. Навігація повинна підтримуватись як мишкою, так і клавіатурою — особливо це важливо для користувачів, які мають обмеження з моторикою. Крім того, бажано реалізувати перемикач теми інтерфейсу (світла/темна), що позитивно впливає на зоровий комфорт під час тривалої роботи з калькулятором.

Одним із критично важливих елементів проєкту є надійна обробка помилок. Якщо користувач вводить некоректний вираз, система повинна не лише уникнути аварійного завершення роботи, але й пояснити помилку:

повідомити про невідповідну кількість дужок, неправильний формат числа або використання забороненого символу. Повідомлення мають бути короткими, зрозумілими і локалізованими.

Зважаючи на обмежені ресурси та реальні потреби користувачів, у межах першої фази розробки доцільно реалізувати MVP (Minimum Viable Product). Ця мінімальна життєздатна версія обчислювача повинна включати базові арифметичні обчислення, підтримку дужок і пріоритетів, інтерфейс для введення та виведення даних, а також валідацію помилок. Розширення функціоналу можливе на наступних етапах: зокрема, може бути додано збереження історії, підтримку змінних, пам’ять (M+, MR, MC), кілька вкладок (базова, наукова, фінансова), збереження у локальному сховищі або синхронізацію з обліковим записом (у майбутніх версіях).

Таким чином, сформульовані цілі й вимоги до проєкту відображають комплексний підхід до створення сучасного інструмента обчислень у веб-середовищі. Система повинна бути не просто калькулятором, а зручним і надійним рішенням, яке гармонійно поєднує зручність інтерфейсу, точність, швидкість обробки даних і технологічну адаптивність. Це забезпечить її застосування у найрізноманітніших сферах — від освіти до технічної діяльності.

1.3 Дизайн та UX/UI особливості інтерактивного обчислювача

Після формулювання цілей і вимог до інтерактивного обчислювача наступним етапом є створення його користувацького інтерфейсу. Цей інтерфейс є не лише візуальною складовою, а й основою для реалізації логіки взаємодії користувача із системою. Саме тому побудова грамотно структурованої HTML-розмітки є критично важливою.

Інтерфейс калькулятора повинен бути інтуїтивно зрозумілим, візуально лаконічним, логічно організованим та адаптованим до різних типів пристроїв. Користувач має миттєво орієнтуватись у функціональних елементах, не потребуючи інструкцій або допоміжних пояснень. В основі структури інтерфейсу лежить розділення на кілька логічних зон: екран (дисплей),

цифровий блок, блок операторів, службові кнопки, наукові функції (опційно), а також елементи розширеного функціоналу, такі як перемикач теми, пам’ять або історія.

Основна HTML-структура реалізується за допомогою сучасних семантичних тегів, а також з урахуванням принципів доступності (ARIA-атрибути, правильні ролі елементів). Наприклад, дисплей виводу результатів оформлюється у вигляді div або output з роллю status, що дозволяє користувачам з порушеннями зору отримувати голосове озвучення результату за допомогою скрінрідерів. Усі кнопки оформлюються як елементи <button>, що автоматично підтримують взаємодію з клавіатурою і є дружніми до мобільного вводу.

Типова структура HTML для калькулятора виглядає так:

```
<div class="calculator">
<div class="calculator__display" role="status" aria-live="polite">0</div>
<div class="calculator__keypad">
<div class="calculator__numbers">
<button class="key key--number">7</button>
<button class="key key--number">8</button>
<!-- Інші цифри -->
</div>
<div class="calculator__operators">
<button class="key key--operator">+</button>
<button class="key key--operator">−</button>
<!-- Інші оператори -->
</div>
<div class="calculator__functions">
<button class="key key--clear">AC</button>
<button class="key key--delete">←</button>
<button class="key key--equals">=</button>
</div>
<div class="calculator__scientific">
<button class="key key--func">sin</button>
<button class="key key--func">log</button>
<!-- Інші наукові функції -->
</div>
</div>
</div>
```

Розмітка створюється з урахуванням методології BEM (Block Element Modifier), яка дозволяє організувати класи CSS у зрозумілу ієрархію. Це забезпечує масштабованість, зручність підтримки та можливість легкого

переходу до будь-якого CSS-фреймворка або кастомної стилізації без зміни HTML-структури.

Важливою складовою є адаптивність інтерфейсу. Всі елементи мають бути організовані у вигляді гнучкої сітки або Flex/Grid-контейнера, щоб забезпечити правильне відображення на екранах різного розміру. Кнопки повинні масштабуватись відповідно до ширини екрана, а блок дисплею - автоматично змінювати шрифт або прокручуватись при перевищенні довжини виразу.

Також доцільно забезпечити можливість поділу інтерфейсу на вкладки: базовий режим (арифметичний), науковий режим (розширені функції) та історія обчислень (опційно). Це дозволяє зробити інтерфейс не перевантаженим, приховуючи менш вживані функції за допомогою JavaScript або CSS.

У розмітці необхідно передбачити структурну і логічну відповідність між інтерфейсними елементами та їх функціональністю. Наприклад, кожна кнопка повинна мати унікальне значення або атрибут data-value, який буде використовуватись у логіці JavaScript для обробки введення. Це спрощує зв'язок між UI та функціональним кодом і підвищує узгодженість взаємодії.

1.4 Огляд існуючих рішень та типових реалізацій інтерактивних обчислювачів у веб-середовищі

У процесі створення інтерфейсу калькулятора важливо обрати відповідну методологію для організації коду, що забезпечить його масштабованість, підтримуваність та зрозумілість. Дві популярні методології в цій сфері - BEM (Block Element Modifier) та Atomic Design.

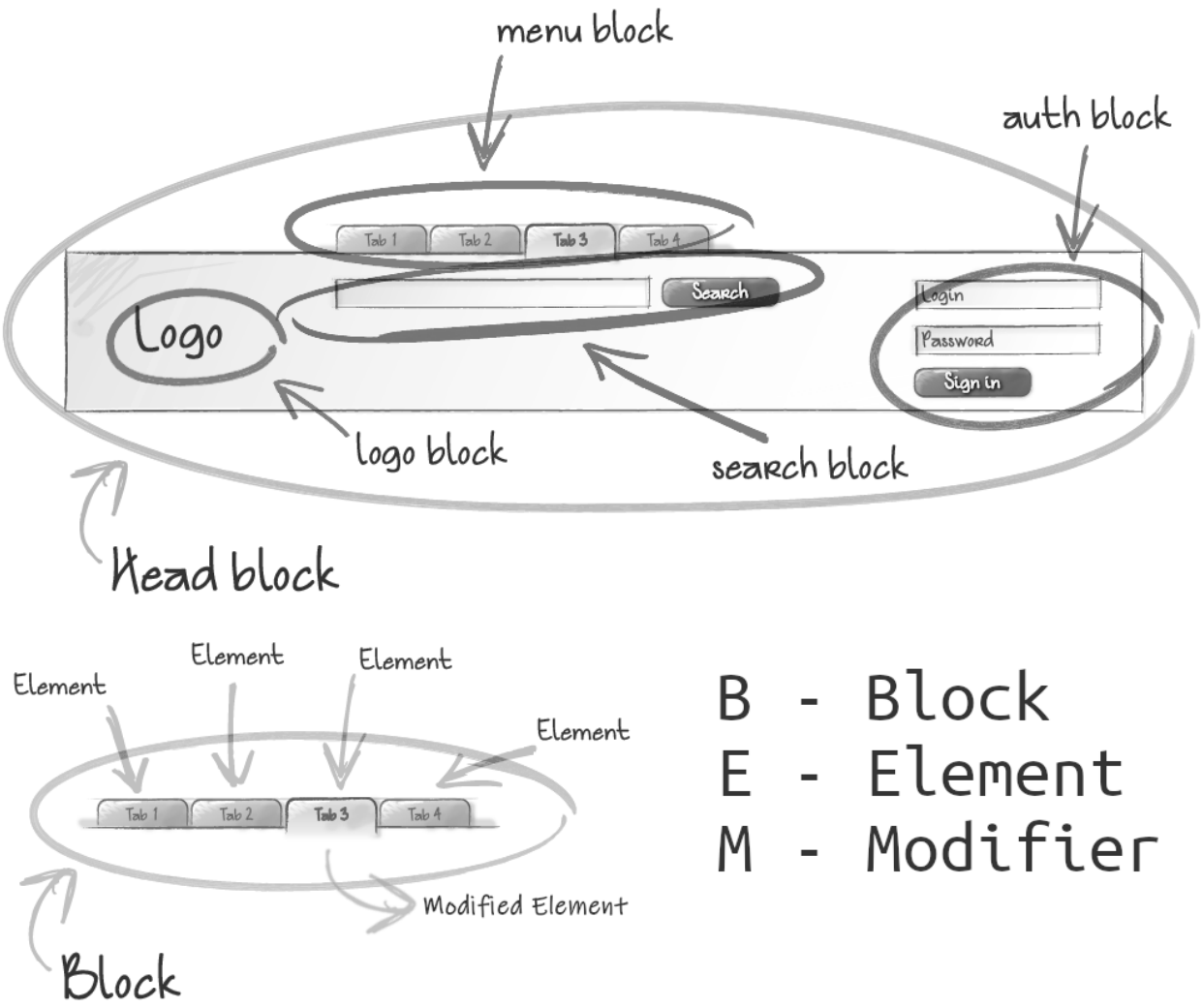

Рисунок 1.5 — Огляд методології BEM

BEM — це методологія, розроблена в компанії Yandex, яка передбачає поділ інтерфейсу на блоки, елементи та модифікатори. Такий підхід дозволяє створювати зрозумілу та передбачувану структуру класів, що полегшує підтримку та масштабування коду. Наприклад, клас calculator__display вказує на елемент display всередині блоку calculator, а клас key-operator є модифікатором елемента key, що вказує на його специфічну функцію. Ця методологія сприяє уникненню конфліктів у назвах класів та забезпечує чітку ієрархію елементів [12].

У свою чергу, Atomic Design, запропонована Бредом Фростом, базується на концепції поділу інтерфейсу на п’ять рівнів: атоми, молекули, організми, шаблони та сторінки. Атоми - це найпростіші складові частини інтерфейсу, такі як кнопки, поля введення чи іконки. Молекули - це комбінації атомів, які

разом утворюють функціональні одиниці, наприклад, панель введення з кнопкою «підтвердити». Організми — це вже більші блоки, що формуються із молекул, наприклад, секція калькулятора з дисплеєм та цифровими кнопками. Шаблони визначають структуру сторінки, а сторінки - це конкретні реалізації шаблонів з наповненням [13].

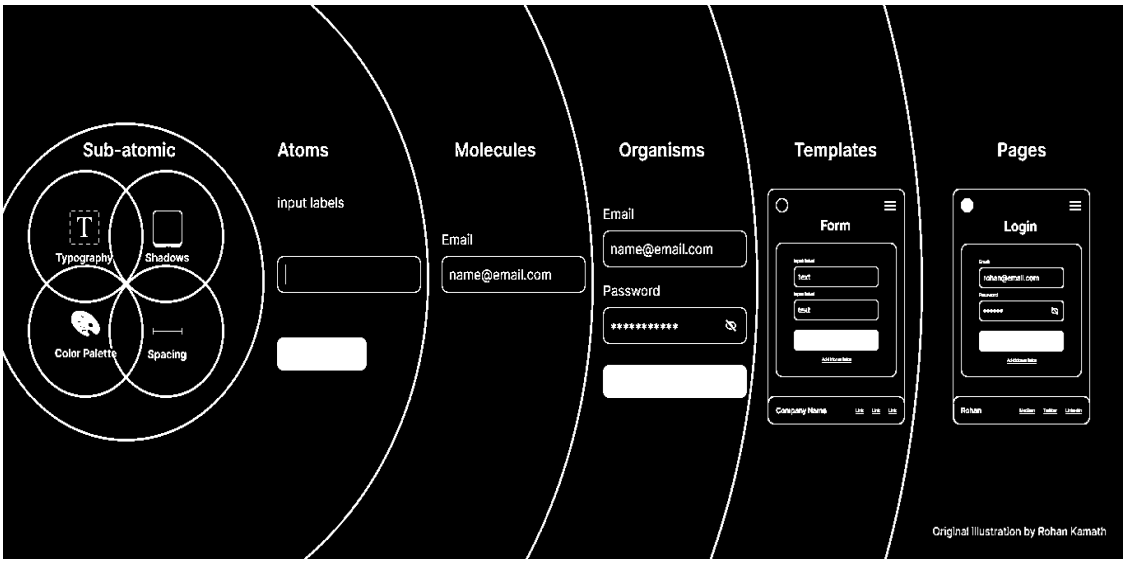

Рисунок 1.6 — Огляд методології Atomic Design

Перевага Atomic Design полягає в тому, що вона змушує мислити компоненто-орієнтовано вже з самого початку, що особливо корисно при роботі з фреймворками на зразок React або Vue. У контексті калькулятора ця методологія дозволяє чітко структурувати інтерфейс: атоми — це кнопки, дисплей, іконки 003B молекули — блок клавіш або панель функцій; організм — повний калькулятор; шаблон — макет сторінки з калькулятором; сторінка — реальний приклад використання. Це особливо корисно в умовах масштабування, коли застосунок з часом розширюється, а елементи інтерфейсу можуть бути повторно використані в різних контекстах [12].

Таким чином, вибір між BEM та Atomic Design залежить від потреб проєкту: BEM є простішою для статичних інтерфейсів, тоді як Atomic Design підходить для масштабованих, компонентних архітектур. У цьому проєкті доцільно комбінувати обидва підходи: використовувати BEM для CSS-структури, а Atomic Design - як логічну модель для побудови інтерфейсу.

Отже, створення HTML-структури інтерфейсу калькулятора — це не

просто розміщення елементів на екрані, а системний процес, що базується на сучасних підходах до розмітки, доступності та методологій організації коду.

Особливу увагу приділено питанням дизайну та UX/UI. Було обґрунтовано доцільність використання методологій Human-Centered Design та Material Design, що забезпечують орієнтацію на користувача, адаптивність інтерфейсу, візуальну узгодженість та відповідність міжнародним стандартам. Крім того, розглянуто принципи доступності, що дозволяють розширити аудиторію застосунку та зробити його зручним для людей з обмеженими можливостями.[18]

Загалом, результати, отримані в межах цього розділу, є основою для подальшої реалізації функціональності обчислювача, забезпечують високий рівень підготовки до етапу програмної реалізації та дозволяють перейти до безпосереднього впровадження логіки роботи системи у наступному розділі.

2 АНАЛІЗ ЗАСОБІВ РОЗРОБКИ ІНТЕРАКТИВНОГО ОБЧИСЛЮВАЧА

2.1 Порівняння популярних мов програмування для реалізації інтерактивного обчислювача

Якість взаємодії користувача з програмним продуктом значною мірою визначає ефективність та зручність його застосування. Відповідно до сучасних підходів до розробки програмного забезпечення, дизайн інтерфейсу та організація користувацького досвіду (UX/UI) мають базуватися на принципах функціональності, доступності, ергономічності та візуальної узгодженості. Врахування цих аспектів є особливо актуальним при створенні інтерфейсів для щоденного використання, зокрема у випадку інтерактивного обчислювача.

Під час проєктування користувацького інтерфейсу інтерактивного обчислювача необхідно дотримуватися низки принципів, що забезпечують ефективність використання системи.

Зокрема, до таких принципів належать:

— інтуїтивність — інтерфейс повинен бути зрозумілим без потреби у додатковому навчанні, розташування елементів, іконографіка та назви мають бути логічними та звичними для користувача;

— консистентність — усі елементи інтерфейсу повинні мати єдиний стиль, зокрема у форматуванні тексту, кнопок, кольоровій гамі, інтервалах та анімації;

— адаптивність — інтерфейс має динамічно змінюватися відповідно до типу пристрою, розміру екрана, орієнтації та роздільної здатності, забезпечення адаптивності досягається за допомогою сучасних CSS-технологій (Flexbox, Grid, media-запити);

— доступність — система повинна бути зручною для користувачів з різними типами обмежень, доцільним є впровадження ARIA-атрибутів, забезпечення високого контрасту, підтримки клавіатурної навігації, а також альтернативних режимів візуалізації (темна тема, збільшення шрифтів);

— оперативний зворотний зв’язок — кожна дія користувача повинна викликати миттєву відповідь інтерфейсу, це можуть бути візуальні ефекти, повідомлення про помилки, анімації при натисканні [14].

Застосування перелічених принципів дозволяє зменшити когнітивне навантаження, прискорити навчання користувача та знизити ймовірність помилок під час взаємодії з системою.

При проєктуванні інтерфейсу доцільно орієнтуватися на перевірені методології, що відповідають міжнародним стандартам. До таких методик належать:

— Human-Centered Design (HCD) – методологія, визначена у стандарті ISO 9241-210:2010 [16], яка передбачає побудову інтерфейсу на основі глибокого аналізу потреб і поведінки кінцевих користувачів, особливістю підходу є циклічність: дослідження — ідеація — прототипування — тестування — вдосконалення [1], цей підхід є доцільним у разі створення універсального веб-застосунку, оскільки враховує різноманіття користувацьких сценаріїв та пристроїв;

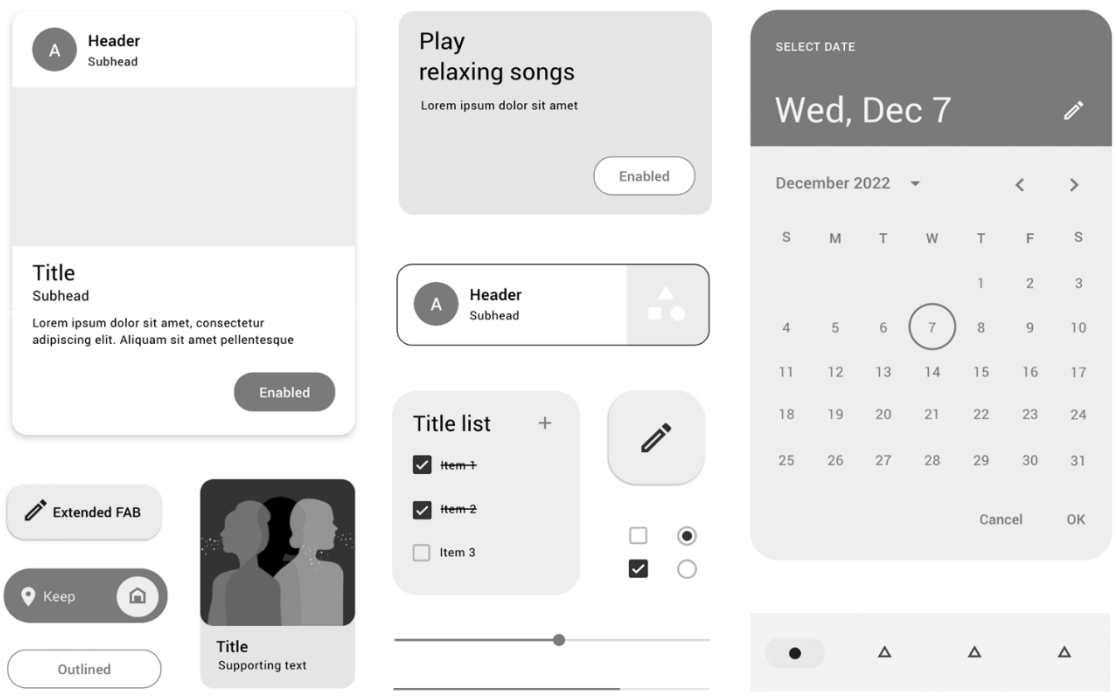

Рисунок 2.1 — Приклад Material Design

— Material Design — концепція, розроблена компанією Google, яка передбачає сувору структуризацію компонентів інтерфейсу, модульну сітку, стандартизовану типографіку, а також систему кольорів і тіней [2], застосування цієї методології забезпечує естетичну цілісність та швидке освоєння інтерфейсу користувачем;[20]

Рисунок2.2 — Методологія Design Thinking

— Design Thinking — методологія, що фокусується на емпатії до користувача, гнучкому створенні прототипів та багатоетапному тестуванні ідей [3], цей підхід дозволяє адаптувати інтерфейс до конкретних груп користувачів шляхом ітеративної взаємодії із цільовою аудиторією [21].

Таблиця 2.1 — Порівняння зазначених підходів

Методологія	**Основний акцент**	**Переваги**	**Обмеження**
Human-Centered Design	Потреби користувача	Орієнтація на широке коло користувачів	Потребує більше етапів тестування
Material Design	Стандартизація візуалу	Висока узгодженість, готові шаблони	Менш гнучкий під неформальні системи
Design Thinking	Креативне вирішення проблем	Адаптивність, гнучкість у змінах	Вимагає глибокої аналітики та часу

У рамках створення інтерактивного обчислювача доцільним є поєднання підходів Human-Centered Design та Material Design. Перший забезпечує відповідність очікуванням користувача, другий – реалізацію візуально та структурно збалансованого інтерфейсу.

У процесі реалізації інтерфейсу калькулятора рекомендується дотримуватись таких практичних рішень:

— розміщення цифрових кнопок у стандартному порядку (3×3) з окремим рядком для нуля;

— операторі розміщуються окремим вертикальним блоком справа;

— колірне виділення функціональних елементів (наприклад, "C", "=") сприяє швидшому сприйняттю;

— кнопки повинні мати мінімальні розміри 48×48 пікселів відповідно до рекомендацій WCAG;

— застосування CSS Grid або Flexbox забезпечує гнучке компонування елементів на будь-якому пристрої.

Таблиця 2.2 - UI/UX гайдлайн

Компонент	Рекомендації
Дисплей	Використовувати елемент <output> з атрибутом aria-live="polite" для забезпечення доступності
Цифрові кнопки	Розміщення у стандартному порядку; розмір не менше 48×48 пікселів
Оператори	Виділення кольором; розміщення праворуч від цифрових кнопок
Функціональні кнопки	Виділення кольором або формою; розміщення в окремому блоці
Адаптивність	Використання CSS Flexbox або Grid для забезпечення гнучкого макету
Доступність	Забезпечення достатнього контрасту; використання ARIA-атрибутів

На етапі формування вимог було визначено основні функціональні можливості системи, до яких належать виконання базових та наукових математичних обчислень, обробка складених виразів, підтримка клавіатурного введення, обробка помилок та забезпечення точності результатів. Також обґрунтовано технічні, нефункціональні та інтерфейсні вимоги, що забезпечують стабільність і зручність роботи обчислювача в різних середовищах.

Розробка HTML-структури інтерфейсу здійснювалася з урахуванням методології BEM, яка забезпечує логічну організацію елементів, легкість у підтримці та розширюваність коду. Було визначено ключові блоки інтерфейсу,

включаючи дисплей, цифрову та операторну клавіатури, а також передбачено можливість додавання розширених функцій (наукових операцій, історії обчислень, перемикача теми тощо).

2.2 Опис архітектури логіки інтерактивного обчислювача: події, змінні, обчислення

На етапі реалізації логіки інтерактивного обчислювача важливо сформувати цілісну архітектуру бізнес-логіки, що дозволяє обробляти події, маніпулювати змінними стану та виконувати обчислення. Враховуючи особливості веб-середовища, логіка такого застосунку, як правило, реалізується у вигляді подієво-орієнтованої системи. Кожна взаємодія користувача з інтерфейсом (наприклад, натискання кнопки або введення символу з клавіатури) сприймається як подія, яка передається до спеціального обробника (event handler) і відповідно викликає зміну внутрішнього стану програми.

Фундаментальними елементами, що формують стан калькулятора, є змінні, які зберігають інформацію про поточне введення, обчислений результат, тимчасові значення та історію. Наприклад, змінна inputBuffer відповідає за збереження рядка з повним математичним виразом, що вводиться користувачем. Змінна currentValue використовується для зберігання останнього активного значення, а структура history може реалізовуватись як масив об'єктів із попередніми виразами та результатами. Задля збереження проміжних операцій також можливо використовувати допоміжний стек, що дозволяє покроково обробляти складені вирази відповідно до пріоритетів операцій.

Особливу роль відіграє модуль обчислення, який відповідає за обробку введених виразів і формування результату. У типових реалізаціях для цього використовується зовнішня бібліотека, зокрема Math.js, що підтримує широкі можливості обчислень: від базової арифметики до обробки математичних функцій, парсингу виразів та роботи з одиницями виміру. Такий підхід

дозволяє уникнути необхідності створення власного парсера та значно спрощує реалізацію.

Для забезпечення ефективного управління кодом доцільно обрати відповідну архітектуру бізнес-логіки. Серед найпоширеніших підходів можна виокремити монолітну, модульну, а також патерни MVC (Model-View-Controller) і MVVM (Model-View-ViewModel).

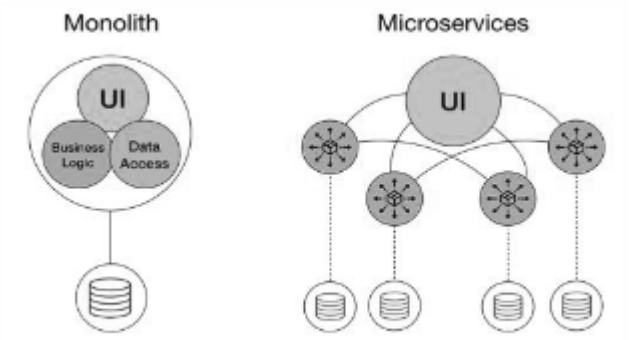

Рисунок 2.3 — Огляд двох типів архітектур

Монолітна архітектура є однією з найпростіших форм організації програмного коду, яка передбачає реалізацію всієї бізнес-логіки, обробки подій, зберігання стану та управління інтерфейсом користувача в межах одного функціонального блоку або файлу. У контексті розробки простих або прототипових веб-застосунків такий підхід часто використовується завдяки своїй легкості у реалізації, мінімальним вимогам до структури проєкту та швидкому старту розробки. Усі компоненти системи тісно пов’язані між собою, що дозволяє уникати складної взаємодії між модулями та суттєво знижує вхідний поріг для нових розробників.

Проте така архітектура має низку обмежень, які критично впливають на ефективність системи у процесі її еволюції. Зокрема, високий ступінь зв’язаності компонентів значно ускладнює внесення змін до окремих частин застосунку, оскільки будь-яке локальне оновлення може мати непередбачуваний вплив на інші функціональні елементи. У результаті, підтримка та тестування системи потребують значних зусиль, а процес масштабування — додавання нових можливостей чи перенесення функціоналу

в інші проєкти — стає складним або неможливим без суттєвого переписування коду [12].

Крім того, монолітна структура часто ускладнює командну розробку, оскільки одночасна робота кількох розробників над однією великою кодовою базою призводить до конфліктів змін і знижує продуктивність. Така архітектура також погано масштабовується з точки зору повторного використання компонентів або створення розширень, що особливо важливо у випадках довготривалої підтримки продукту.

Таким чином, монолітна архітектура, попри свою простоту, є доцільною лише для невеликих проєктів або як початкова реалізація функціоналу з метою подальшого переходу до більш гнучких архітектурних рішень.

Натомість модульна архітектура ґрунтується на принципі декомпозиції системи на окремі логічно ізольовані частини (модулі), кожен з яких відповідає за конкретну підсистему або функціональний аспект застосунку. У випадку реалізації інтерактивного обчислювача це можуть бути окремі модулі для обробки введення даних, виконання обчислень, управління відображенням результатів, обробки помилок, а також керування історією та пам’яттю операцій. Такий підхід дозволяє розробнику підтримувати чітку структуру проєкту, де кожен компонент має обмежену зону відповідальності і взаємодіє з іншими через визначені інтерфейси або точки доступу.

Завдяки модульності значно спрощується підтримка системи, оскільки модифікація або вдосконалення одного з компонентів, як правило, не впливає на роботу інших. Це особливо важливо у контексті довготривалих проєктів або при роботі над програмним забезпеченням у команді. Модулі також полегшують процес тестування, оскільки кожен з них можна перевіряти окремо за допомогою модульних або інтеграційних тестів.

Крім того, модульна архітектура сприяє повторному використанню коду, адже функціональні блоки можуть бути адаптовані та використані в інших проєктах без суттєвих змін. У процесі масштабування застосунку така архітектура дозволяє ефективно додавати нові функціональні можливості,

інтегруючи їх у вигляді нових модулів без необхідності суттєвої перебудови наявного коду.

У веб-контексті модульна архітектура добре поєднується з компонентним підходом, який підтримується більшістю сучасних JavaScript-фреймворків. Завдяки цьому вона забезпечує баланс між структурною простотою та гнучкістю, а також адаптується до потреб як малих, так і середніх за складністю застосунків. Таким чином, модульна архітектура виступає оптимальним вибором у випадках, коли потрібна керованість, розширюваність і зручність супроводу програмного забезпечення.

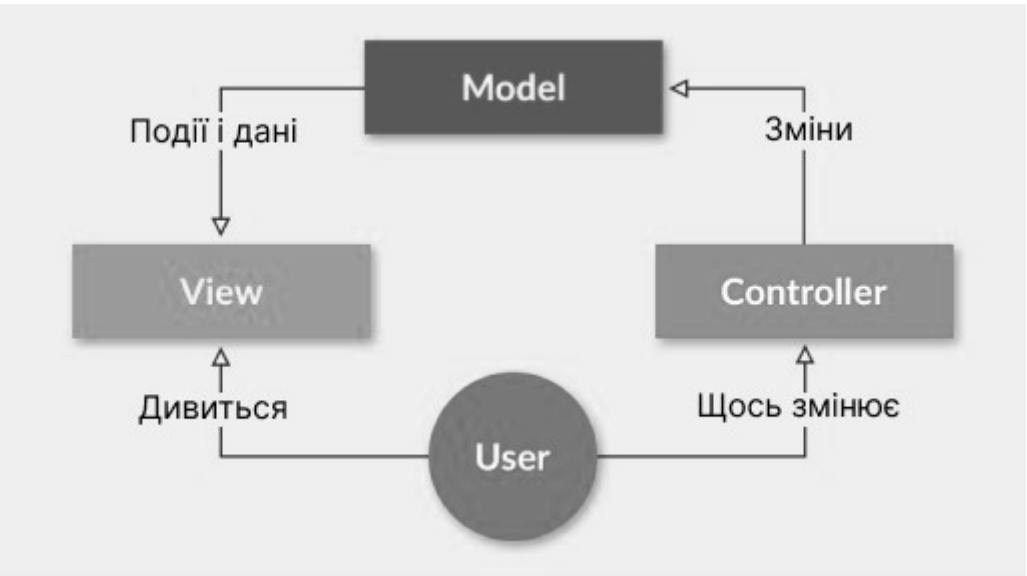

Рисунок 2.4 — Огяд MVC архітектури

Архітектура MVC (Model–View–Controller) реалізує принцип розділення відповідальностей між трьома ключовими компонентами: моделлю, поданням (представленням) та контролером. Модель (Model) відповідає за збереження, обробку та оновлення даних, що використовуються в застосунку. Представлення (View) виконує функцію відображення цієї інформації для користувача у зручному візуальному форматі. Контролер (Controller) виступає як проміжна ланка, що обробляє дії користувача, оновлює модель і ініціює оновлення представлення відповідно до нових даних.

Основна перевага цього підходу полягає у чіткій розділеності обов’язків між логічними шарами застосунку, що підвищує зрозумілість і керованість

коду, полегшує процес тестування окремих компонентів, а також дозволяє кільком розробникам паралельно працювати над різними частинами системи без суттєвого дублювання зусиль.

У контексті реалізації інтерактивного обчислювача використання архітектури MVC дозволяє ізолювати модуль обчислення математичних виразів від логіки обробки подій та відображення результатів, що робить систему більш стабільною, передбачуваною та гнучкою до змін. Наприклад, при внесенні змін до механізму обчислення не виникає необхідності модифікувати структуру інтерфейсу, оскільки ці частини системи слабо пов’язані між собою.

Такий підхід також є ефективним з погляду масштабування застосунку: у разі необхідності додавання нових функцій (наприклад, модулів історії обчислень, пам’яті або розширених наукових функцій) розширення може бути здійснено без повного перепроектування системи, що значно підвищує її життєздатність у довготривалій перспективі.

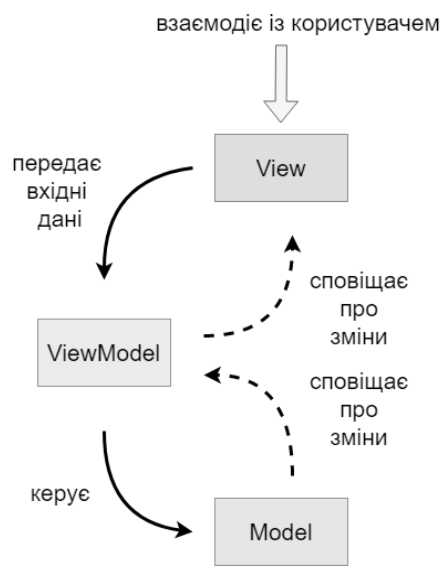

Рисунок 2.5 — Діаграма роботи архітектри MVVM

Альтернативою є архітектура MVVM (Model–View–ViewModel), яка набула популярності у сучасних фронтенд-фреймворках, зокрема Vue.js, Knockout.js та Angular. Основна ідея цього підходу полягає у впровадженні проміжної ланки - компонента ViewModel, який виступає як посередник між

модельною частиною застосунку та інтерфейсом користувача. ViewModel відповідає за зберігання стану інтерфейсу, обробку бізнес-логіки, реакцію на події та зв'язок із моделлю даних.

Ключовою особливістю архітектури MVVM є реалізація двобічного зв’язування даних (two-way data binding), що забезпечує автоматичну синхронізацію змін між моделлю і представленням. Це означає, що будь-яка зміна в моделі миттєво відображається в інтерфейсі, і навпаки - зміни в полі введення автоматично оновлюють відповідні змінні у логіці програми.

Завдяки цій властивості значно скорочується обсяг коду, який відповідає за безпосередню маніпуляцію елементами DOM, що не лише спрощує розробку, але й позитивно впливає на зручність модульного тестування. Крім того, підхід MVVM полегшує повторне використання компонентів інтерфейсу, дозволяє ефективно розмежувати відповідальність між різними частинами застосунку та сприяє кращому масштабуванню системи.

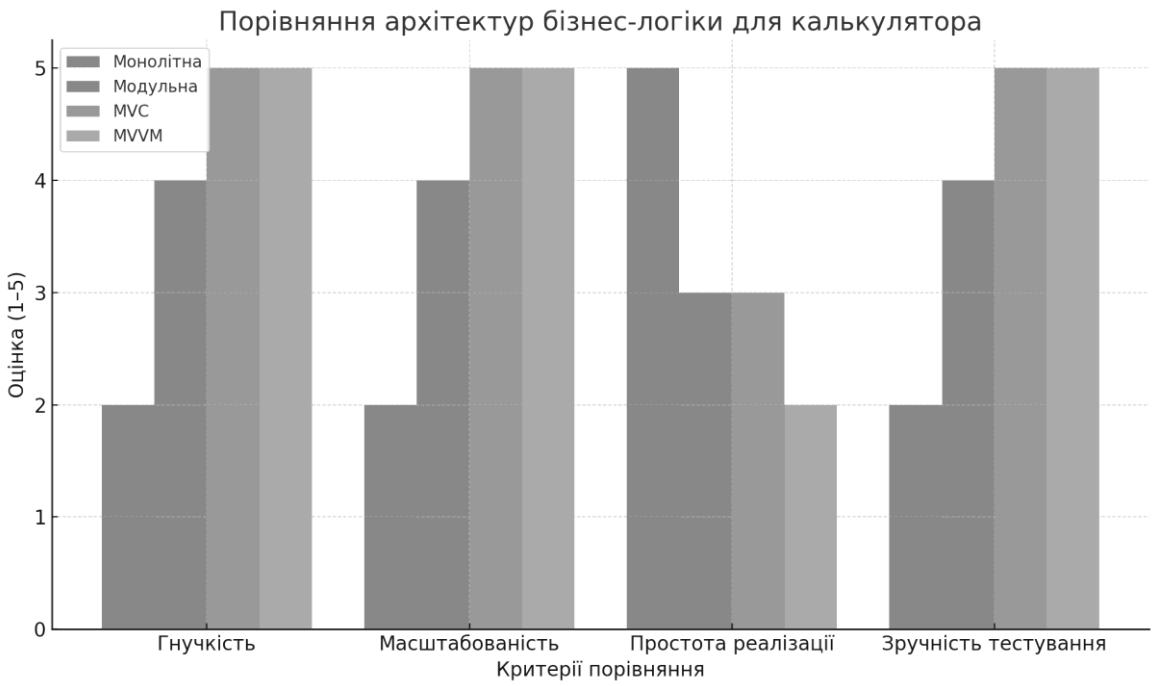

Рисунок 2.6 — Стовпчикова діаграма порівняння архітектур бізнес логіки

Проведене порівняння свідчить, що для розробки обчислювача середньої складності найбільш доцільним є використання модульної архітектури з елементами MVC. Такий підхід дозволяє організувати код структуровано,

спрощує масштабування функціональності, сприяє модульному тестуванню та підвищує загальну якість розробки.

Окрему увагу в архітектурі бізнес-логіки слід приділити реалізації обробки помилок. Система має вміти реагувати на некоректний синтаксис виразів, числові винятки (наприклад, ділення на нуль), а також попереджати користувача про помилки зрозумілим способом. Це може реалізовуватись як через обробку винятків у функції обчислення, так і через валідацію введення перед передачею до обчислювального модуля.

Таким чином, архітектура логіки інтерактивного обчислювача є критично важливою складовою всієї системи. Від правильного вибору архітектурної моделі залежить ефективність, надійність та гнучкість застосунку, а також можливість його подальшого розширення й супроводу.

2.3 Інтерфейс користувача та технічна реалізація

Одним з ключових етапів реалізації інтерактивного обчислювача є забезпечення коректної обробки введення користувача, а також реалізація системи виявлення та реагування на помилки, що можуть виникати як у процесі введення даних, так і під час обчислень. Ці аспекти критично впливають на стабільність, надійність та загальне сприйняття системи кінцевим користувачем.

Усі взаємодії користувача з калькулятором відбуваються через інтерфейс, що формує події, пов’язані з натисканням клавіш (віртуальних або фізичних), очищенням вводу, використанням службових функцій або запуском обчислень. Для їх обробки використовується система слухачів подій (event listeners), що реалізується за допомогою мовних засобів JavaScript. Відповідно до стандартного підходу, кожен натиск на клавішу викликає функцію, яка змінює стан буфера введення або виконує обчислення.

Користувацьке введення зберігається у вигляді рядка або масиву символів. У процесі побудови виразу здійснюється поетапна перевірка введених даних. Зокрема, перевіряється правильність поєднання чисел та операторів, наявність відкритих та закритих дужок, відсутність повторних або

недопустимих символів. У випадку виявлення синтаксичних помилок система повинна або автоматично їх виправляти, або блокувати подальше введення з відповідним повідомленням для користувача.

Особливої уваги потребує валідація математичних виразів. Введений рядок необхідно перевіряти на відповідність базовим правилам алгебри: правильне чергування операндів і операторів, коректність дужкових конструкцій, відсутність поділу на нуль або коренів із від'ємних чисел (у разі роботи лише з дійсними числами). Для цього може використовуватись як власний алгоритм перевірки, так і засоби бібліотеки Math.js, яка дозволяє виконати парсинг виразу перед безпосереднім обчисленням.

Після проходження етапу валідації система передає вираз до обчислювального модуля. Результат, що повертається, відображається в інтерфейсі. У разі виникнення помилки (наприклад, «SyntaxError» або «DivisionByZeroError») система повинна не лише запобігти аварійному завершенню програми, але й надати користувачеві зрозуміле пояснення. Це реалізується шляхом виведення короткого текстового повідомлення (наприклад, «Помилка виразу» або «Ділення на нуль неможливе»), що допомагає зберігати позитивний користувацький досвід і формує довіру до продукту.

Також важливо реалізувати систему обробки некоректного або неповного введення, яке може виникати у процесі використання калькулятора на мобільних пристроях або при швидкому введенні з клавіатури. Для таких випадків рекомендується реалізувати механізми автоматичної корекції: наприклад, заміна подвійних операторів одним або автоматичне доповнення закриваючої дужки в кінці виразу.

Рисунок 2.7 — Блок схема обробки введення користувача та помилок

Окремим аспектом є забезпечення доступності повідомлень про помилки. Відповідно до вимог WCAG 2.1, повідомлення мають бути доступними для користувачів із порушеннями зору, що досягається шляхом застосування ARIA-атрибутів (aria-live="polite") для динамічних елементів інтерфейсу.

Таким чином, коректна обробка введення користувача та реалізація обробки помилок є не лише технічним завданням, а й важливою складовою забезпечення зручності та надійності програмного продукту. Від якості реалізації цього модуля залежить ступінь задоволеності користувача, стабільність системи та її придатність до реального використання.

Отже, на початковому етапі було сформовано архітектурну модель, що визначає структуру взаємодії між компонентами обчислювача. Проведено порівняльний аналіз підходів до побудови бізнес-логіки, зокрема монолітної, модульної, MVC та MVVM архітектур. На основі цього аналізу обґрунтовано доцільність використання модульного підходу з елементами шаблону MVC, що дозволяє забезпечити гнучкість, масштабованість та зручність супроводу коду.

Особливу увагу приділено реалізації механізмів обробки користувацького введення та виявлення помилок. Запропонована система подієво-орієнтованої обробки забезпечує оперативну реакцію на дії

користувача, збереження введених даних у буфері, перевірку синтаксису виразів та виведення результатів у зручному форматі. У випадку помилок система формує зрозумілі повідомлення та не допускає аварійного завершення виконання, що відповідає сучасним вимогам до UX та надійності.

Завдяки використанню математичних бібліотек, таких як Math.js та Decimal.js, реалізовано точні обчислення як для базових арифметичних операцій, так і для наукових функцій. Забезпечено підтримку тригонометричних функцій, логарифмів, степеневих операцій, а також обчислення за правилами математичного порядку дій. Окремо впроваджено підтримку таких розширених функцій, як історія обчислень, пам’ять, динамічне перемикання тем інтерфейсу та робота з вкладками.

3 РОЗРОБКА ПРОЄКТУ ТА ТЕСТУВАННЯ

3.1 Реалізація математичних операцій та підтримка розширених функцій

Математична функціональність є ключовою складовою будь-якого обчислювача, оскільки саме вона забезпечує основну логіку роботи системи — виконання обчислень на основі введених користувачем виразів. Тому реалізація відповідного функціоналу має бути точною, стабільною, масштабованою і такою, що легко розширюється.

На базовому рівні калькулятор виконує основні арифметичні операції: додавання, віднімання, множення та ділення. Ці дії можуть бути реалізовані засобами мови програмування JavaScript за допомогою стандартних операторів +, -, * та /. Проте в контексті обробки десяткових чисел така реалізація може супроводжуватись похибками, обумовленими обмеженнями представлення чисел з плаваючою комою у форматі IEEE 754. Наприклад, результат обчислення 0.1 + 0.2 у JavaScript дорівнює 0.30000000000000004, що є неприйнятним у більшості практичних застосунків. Для усунення таких похибок доцільно використовувати зовнішні бібліотеки з високою точністю обчислень, наприклад, Decimal.js.

Окрім базових функцій, сучасні інтерактивні калькулятори реалізовують розширені наукові можливості. Серед них: обчислення степенів, добування коренів, логарифмічні, експоненційні функції, тригонометричні обчислення (синус, косинус, тангенс), факторіали та інші спеціальні математичні операції. Для їх реалізації найчастіше використовується бібліотека Math.js, яка надає широкий набір математичних функцій і дозволяє опрацьовувати вирази у вигляді рядків.

Наприклад, реалізація обробки математичного виразу може виглядати так:

```
import { evaluate } from 'mathjs';
const input = "sqrt(16) + log(100, 10) + 2^3";
try {
const result = evaluate(input);
console.log(`Результат: ${result}`); // Виведе: Результат: 16
} catch (error) {
```

```
    console.error("Помилка обчислення:", error.message);
                                                    }
```

У наведеному прикладі функція evaluate парсить текстовий вираз і виконує його обчислення з дотриманням правил порядку дій. Така реалізація дає змогу працювати з виразами будь-якої складності, уникаючи ручного розбору та обробки.

Особливістю реалізації є підтримка символічних виразів, тобто виразів, що містять змінні, дужки, комбінації операцій та математичні функції. Вирази на зразок 2 * (3 + sqrt(9)) або log(100, 10) не можуть бути оброблені нативними засобами JavaScript, проте вони можуть бути розпізнані і проаналізовані засобами Math.js, що істотно розширює функціональні можливості застосунку. Крім того, забезпечується підтримка універсальних математичних констант, таких як π або e, а також введення та обчислення виразів зі змінними (наприклад, x = 2; x^2 + 3*x).

Ще одним важливим аспектом є дотримання стандартного математичного порядку операцій, що зазвичай описується абревіатурами PEMDAS або BEDMAS. Для обробки виразів у такому порядку використовується або дерево синтаксичного розбору (expression tree), або алгоритм перетворення з інфіксної форми до зворотної польської нотації (Reverse Polish Notation, RPN). Цей алгоритм дозволяє обробляти складені вирази в правильному порядку, навіть якщо вони містять вкладені дужки або пріоритетні оператори.

Крім математичних функцій, калькулятор також підтримує розширені можливості, які включають:

— історію обчислень, що зберігає попередні вирази та результати для повторного використання;

— механізм пам’яті (M+, M-, MR, MC) для збереження проміжних значень;

— перемикання теми інтерфейсу між світлим і темним режимом;

— використання вкладок або режимів, які дозволяють перемикатися між базовим, науковим та історичним режимами.

З технічного погляду, реалізація кожної функції складається з двох складових: представлення в інтерфейсі (HTML/React/Vue-компонент) та обробника дій на JavaScript. Для зв’язування кнопок із функціями обробки використовуються атрибути data-func, наприклад:

```
<button data-func="sqrt">√</button>
<button data-func="log">log</button>
```

У коді JavaScript відбувається обробка подій зчитування значень:

```
document.querySelectorAll("[data-func]").forEach(button => {
  button.addEventListener("click", () => {
    const func = button.getAttribute("data-func");
    inputBuffer += `${func}(`;
    updateDisplay();
  });
});
```

Окрім реалізації функцій, необхідно враховувати доступність системи (додавання ARIA-атрибутів, голосової підтримки для скрінрідерів), адаптивність до різних пристроїв, а також підтримку масштабування функціональності шляхом динамічного додавання нових функцій без порушення основної структури застосунку.

Таким чином, реалізація математичних та розширених функцій в інтерактивному обчислювачі є результатом поєднання засобів JavaScript, зовнішніх бібліотек та інтерфейсних рішень. Висока точність, підтримка символічних виразів, зручність розширення та відповідність потребам користувача є основою якісного функціонального ядра сучасного веб-калькулятора.

Однією з обов’язкових умов забезпечення якісного користувацького досвіду для веб-застосунку є сумісність з різними браузерами та пристроями. Це особливо актуально для інтерактивного обчислювача, який орієнтований на широку аудиторію користувачів із різними технічними засобами доступу: настільними ПК, ноутбуками, планшетами, смартфонами, а також різними

браузерами (Google Chrome, Mozilla Firefox, Safari, Microsoft Edge тощо). Врахування цих факторів є важливою частиною технічного забезпечення роботи застосунку.

У процесі тестування сумісності основна увага приділяється наступним аспектам:

— коректне відображення інтерфейсу — перевірка, чи однаково виглядає і функціонує користувацький інтерфейс у різних браузерах і при зміні розмірів вікна;

— функціональність обробки подій — перевірка, чи однаково реагують кнопки, поля введення та службові елементи на дії користувача в різних браузерах;

— виконання скриптів JavaScript — оцінка підтримки функцій ES6+ та роботи зовнішніх бібліотек, таких як Math.js, Decimal.js тощо;

— адаптивність інтерфейсу — перевірка, чи забезпечено коректне відображення та масштабування елементів інтерфейсу на екранах з різною роздільною здатністю.

Для перевірки використовуються як реальні пристрої, так і емулятори, вбудовані у сучасні засоби розробки. Наприклад, середовище Google Chrome DevTools надає змогу симулювати десятки типових пристроїв, перевіряти адаптивність CSS-сіток, а також переглядати поведінку подій під час зміни орієнтації або зміни масштабу сторінки.

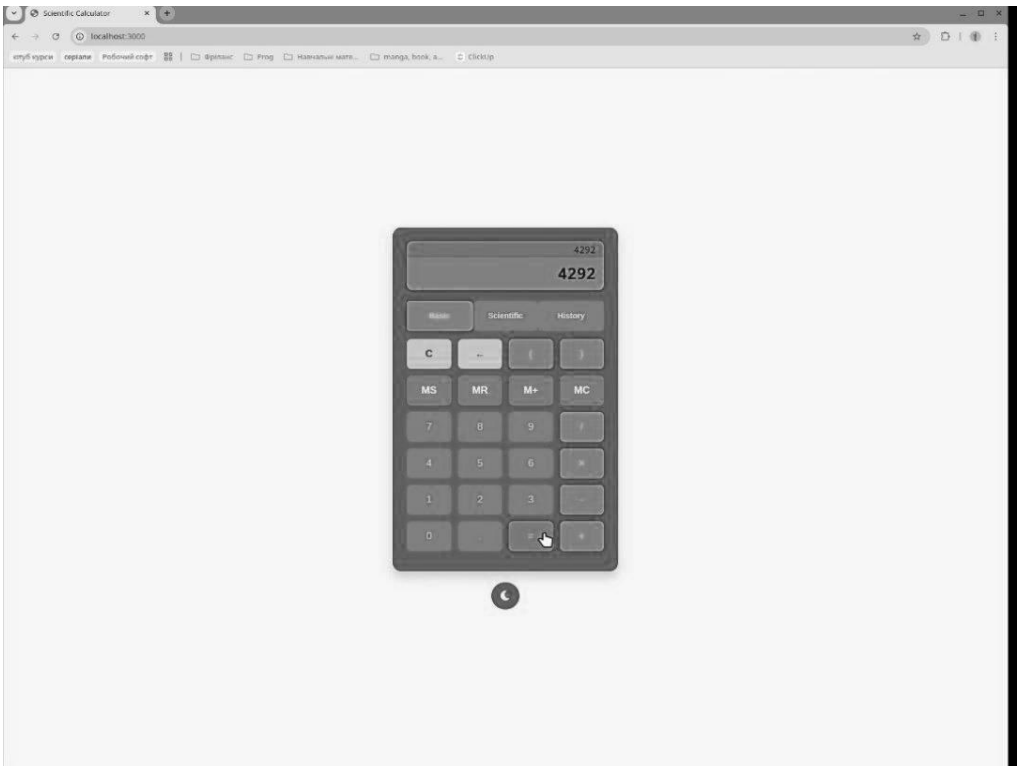

Рисунок 4.1 — Результат роботи додатку в Google Chrome

Оскільки веб-застосунок використовує стандартизовані веб-технології (HTML5, CSS3, JavaScript ES6+), основним завданням є забезпечення перехресної підтримки DOM, уникнення використання застарілих або експериментальних API, а також дотримання принципів прогресивного покращення (progressive enhancement). Це дозволяє системі залишатися функціонально придатною навіть у браузерах без підтримки найновіших можливостей. Такий підхід підвищує надійність і доступність застосунку для ширшої аудиторії, включаючи користувачів зі старішими пристроями або обмеженим доступом до оновлень.

Нижче наведено приклад типового протоколу тестування сумісності у талиниці 3.1.

Таблиця 3.1 — Приклад типового протоколу тестування сумісності

Пристрій	**Браузер**	**Роздільна здатність, dpi**	**Результат тестування**
ПК	Google Chrome	1920×1080	Успішно: повна сумісність
Смартфон	Safari (iOS)	375×812	Успішно: коректна адаптація інтерфейсу

Планшет	Mozilla Firefox	768×1024	Успішно: усі функції працюють стабільно
ПК	Microsoft Edge	1366×768	Успішно: відображення та логіка працюють

У разі виявлення помилок або неочікуваної поведінки застосунку в окремих середовищах, реалізується система зворотного зв’язку (debugging). Вона базується на обробці помилок JavaScript (try/catch), використанні інструментів розробника браузера (DevTools, консоль, мережеві запити) та логуванні подій у консоль.

Таким чином, тестування сумісності є важливою частиною технічного забезпечення інтерактивного обчислювача. Воно гарантує стабільну роботу системи у різних умовах, що є необхідним для досягнення високого рівня доступності та задоволеності користувачів.

Під час розробки інтерактивного обчислювача була застосована система контролю версій Git у поєднанні з хмарним сховищем GitHub, що є загальноприйнятою практикою у сучасній програмній інженерії. Ці інструменти забезпечили організацію процесу розробки відповідно до вимог надійності, контрольованості та відтворюваності змін у програмному коді.

Git - це розподілена система контролю версій з відкритим програмним кодом, що дозволяє ефективно зберігати історію змін у програмному забезпеченні, підтримувати одночасну роботу кількох розробників над одним проєктом, здійснювати резервне копіювання та відновлення змінених версій коду. Розподілена архітектура Git передбачає, що кожен учасник проєкту має локальну копію всього репозиторію, включно з повною історією змін, що дозволяє працювати в автономному режимі.

У процесі розробки застосунку Git використовувався для:

— фіксації змін у кодовій базі проєкту (комітування);

— створення альтернативних гілок розробки для реалізації окремих функціональностей;

— об’єднання змін із різних гілок в основну гілку (main);

— відновлення попередніх версій системи в разі виникнення помилок;

— спільної роботи над кодом і документування змін.

Важливим компонентом системи Git є класифікація файлів за станами:

— змінений файл - файл, унесено зміни, але їх ще не збережено в історії;

— індексований файл - файл підготовлено до збереження у вигляді коміту;

— зафіксований файл - зміни збережено в історії репозиторію.

У структурному плані це відповідає трьом рівням роботи:

— робоча директорія (Working Directory) - місце внесення змін у файл;

— область індексації (Staging Area) - буфер змін перед збереженням;

— каталог Git (репозиторій) - постійне сховище знімків проєкту.

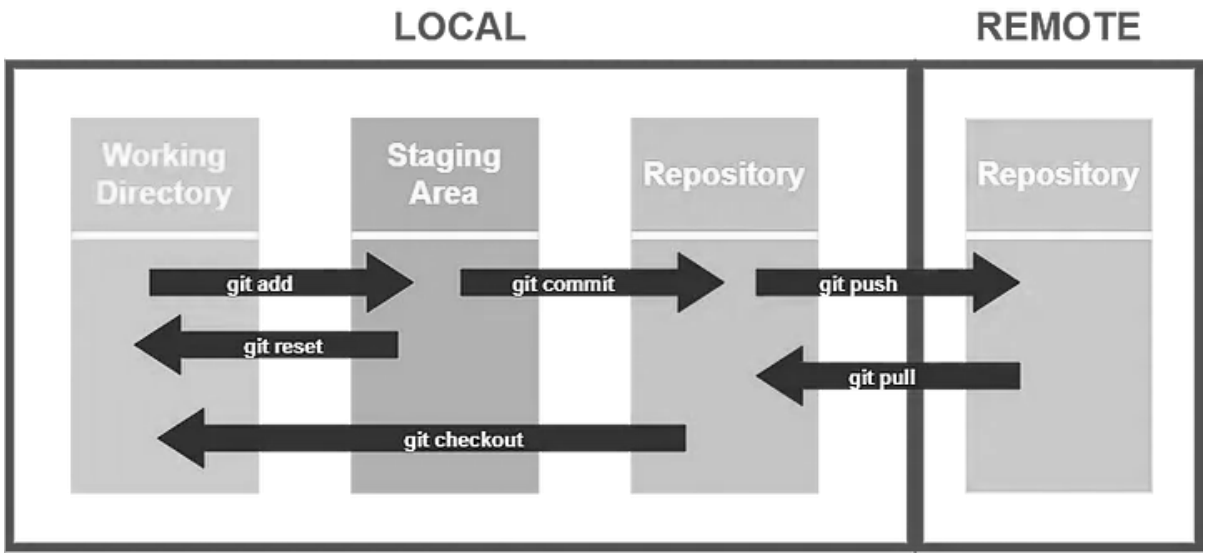

Рисунок 3.2 — Огляд роботи Git

Для забезпечення зручності співпраці та централізованого управління було використано хмарний сервіс GitHub, який виступає як платформа для розміщення репозиторію, ведення документації, обговорення змін у вигляді запитів на злиття (pull requests), організації рецензування коду, створення звітів про помилки та контроль версій релізів.

Git зберігає кожну зміну у вигляді окремого коміту (commit), що створює повноцінну історію розробки з можливістю відтворення будь-якої попередньої версії. Завдяки гілкуванню (branching) розробники мають змогу ізолювати функціональні фрагменти (features), експериментальні частини (experiments)

або виправлення помилок (bugfixes) в окремих середовищах, не впливаючи на стабільну версію проєкту.

У рамках розробки інтерактивного обчислювача було ініціалізовано окремий Git-репозиторій, структура якого містила основну гілку main, а також ряд функціональних гілок:

—feature/ui-design - створення HTML-структури калькулятора;

—feature/js-logic - реалізація логіки введення та обчислення;

—feature/advanced-functions - підтримка наукових функцій;

—feature/history-memory — реалізація пам’яті та історії обчислень;

— fix/decimal-precision — виправлення похибок точності десяткових обчислень.

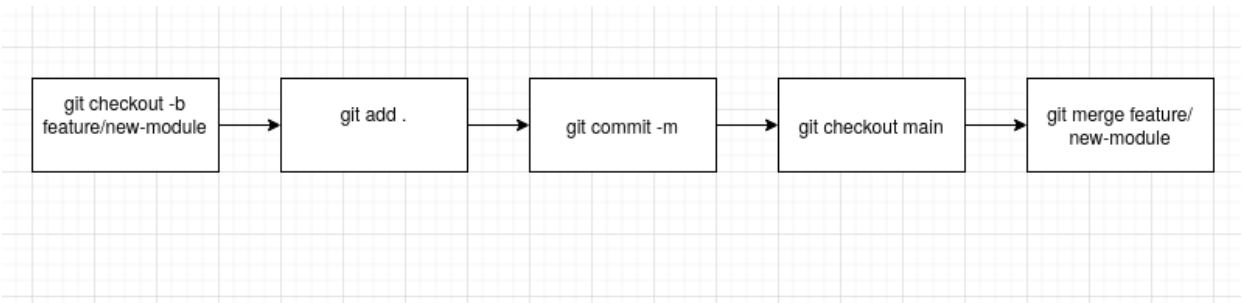

Рисунок 3.3 — Типовий робочий цикл розробки

Типовий робочий цикл складався зі створення нової гілки, додавання змін до індексу, фіксація з повідомленням та поверненням до основної гілки

Було впроваджено стандарт написання повідомлень до комітів відповідно до Conventional Commits, що дозволило підтримувати логічну структуру історії змін.

Застосування Git не лише підвищило технічну якість проєкту, але й сприяло розвитку навичок командної розробки, модульного мислення та відповідального підходу до керування кодом.

Забезпечення надійності функціонування програмного забезпечення є одним з основних вимог до сучасних вебзастосунків. Особливо це актуально для інтерактивних обчислювачів, які безпосередньо працюють із введеними користувачем даними. Неналежна перевірка або відсутність обробки помилкових введень може призвести до некоректного функціонування,

відображення неправильних результатів або порушення цілісності сеансу роботи.

У межах розробки інтерактивного обчислювача були реалізовані декілька рівнів захисту, що забезпечують коректну обробку даних, стійкість до помилок і загальну стабільність застосунку.

Одним з основних етапів захисту системи від некоректного або потенційно небезпечного вводу є реалізація попередньої перевірки введених даних. У рамках даного програмного забезпечення була передбачена внутрішня валідація математичних виразів, яка відбувається ще до моменту їх передавання на обчислення. Це дозволяє значною мірою запобігти помилкам, пов’язаним із некоректним синтаксисом, знизити навантаження на обчислювальний модуль і забезпечити кращий користувацький досвід.

Передусім, реалізовано перевірку допустимості символів, які вводяться користувачем. Усі введення проходять фільтрацію за попередньо визначеним переліком дозволених символів. До нього належать арабські цифри (0–9), основні арифметичні оператори (+, −, *, /), дужки ((,)), десяткова крапка (.), а також символи, що відповідають розширеним математичним операціям (sqrt, log, sin, cos тощо). Введення інших символів, зокрема літер, спеціальних знаків або елементів коду, блокується в момент натискання клавіші, тобто на етапі введення в поле. Такий підхід дозволяє ще до передачі даних на обробку усунути некоректні комбінації та запобігти неконтрольованим станам системи.

Крім перевірки допустимості символів, впроваджено перевірку синтаксичної правильності виразів, зокрема контроль парності дужок. Відкриті дужки мають бути належним чином закриті, а вкладеність має відповідати правилам математичної нотації. У разі виявлення дисбалансу (наприклад, зайва дужка, зміщення вкладення або відсутність пари) користувач отримує відповідне попередження, і такий вираз не допускається до подальшої обробки. Це особливо важливо при реалізації складних виразів, де дужки визначають порядок виконання дій.

Додатково реалізовано перевірку на повторення операторів, наприклад, недопустимі комбінації ++, --, ** автоматично фільтруються або замінюються

на коректні варіанти, якщо це можливо. Аналогічним чином, заборонено починати вираз із оператора (+, *, /) або завершувати його без числового значення, оскільки це є логічною помилкою виразу.

З метою підвищення гнучкості було передбачено автоматичне перетворення десяткової коми на крапку, що забезпечує сумісність із мовою JavaScript та зменшує ймовірність помилки для користувачів, які звикли до національного формату чисел.

Загалом, багаторівнева валідація введення забезпечила стабільність роботи інтерфейсу, запобігла передачі некоректних виразів до обчислювального ядра та дозволила реалізувати більш інтуїтивну логіку взаємодії з користувачем.

Таблиця 4.2 — Типи помилок та реакція системи

Тип помилки	Опис помилки	Реакція системи
Недопустимий символ	Введено символ, що не є цифрою, оператором або дозволеним функціональним елементом.	Символ блокується ще на етапі введення.
Непарні дужки	Кількість відкриваючих і закриваючих дужок не збігається або порядок вкладення порушено.	Відображається попередження, вираз не передається до обчислення.
Подвійні оператори	Використано послідовність однакових операторів без значення між ними (наприклад, ++ або **).	Оператори автоматично фільтруються або підсвічуються як помилка.
Початок/кінець виразу оператором	Вираз починається або закінчується математичним оператором без операнду.	Вираз вважається некоректним, користувач отримує повідомлення про помилку.
Десяткова кома	Використано кому замість крапки для позначення десяткового дробу.	Кома автоматично замінюється на крапку.
Порожнє поле введення	Користувач не ввів жодного символу перед натисканням кнопки обчислення.	Виводиться повідомлення з проханням ввести вираз.

Другий рівень забезпечення надійності роботи інтерактивного обчислювача полягає в реалізації системи обробки помилок під час виконання математичних обчислень. Це особливо важливо з огляду на те, що навіть при правильному синтаксисі введеного виразу можуть виникати логічні або арифметичні помилки, які призводять до непередбачуваного функціонування застосунку.

Для обробки таких помилок у середовищі JavaScript був реалізований механізм перехоплення виключень з використанням конструкції try...catch. Всі математичні операції, які виконуються за допомогою бібліотеки math.js, були інкапсульовані в захищений блок коду, що дозволяє безпечно аналізувати та реагувати на можливі виключення без зупинки виконання програми.

```
try {
const result = math.evaluate(expression);
displayResult(result);
} catch (error) {
showError("Помилка у виразі. Перевірте введені дані.");
}
```

Такий підхід дозволяє ефективно виявляти низку критичних ситуацій, серед яких:

— помилки синтаксичного аналізу виразу (наприклад, невпізнаний оператор або функція);

— арифметичні виключення, зокрема спроба ділення на нуль;

— перевищення максимальної довжини виразу або обчислень над допустимими числовими межами;

— застосування математичних функцій до недопустимих аргументів, як-от обчислення квадратного кореня з від’ємного числа або логарифма від нуля.

У разі виникнення виняткової ситуації застосунок не припиняє свою роботу. Замість цього він викликає функцію виводу повідомлення (showError()), яка інформує користувача про помилку у виразі та пропонує виправити її. Таким чином, забезпечується безперервність взаємодії з

користувачем без потреби перезавантаження сторінки чи втрати поточного контексту.

Окрім базового повідомлення про помилку, можна також реалізувати розширений формат діагностики, у якому уточнюється характер виключення (наприклад: «Ділення на нуль неможливе», «Функція "log" отримала некоректний аргумент» тощо). Це суттєво покращує зручність роботи та сприяє навчанню користувачів.

Особливу увагу було приділено стабільності при повторному обчисленні. Після виявлення помилки, поле введення зберігає останній вираз, що дає змогу користувачеві швидко його виправити, а не вводити повторно. Це суттєво знижує фрустрацію та покращує загальну взаємодію з системою.

Таким чином, впровадження механізму перехоплення помилок під час виконання обчислень дозволило підвищити надійність функціонування обчислювача, зменшити кількість аварійних зупинок системи, а також забезпечити якісний зворотний зв’язок для користувача у разі виникнення помилкових ситуацій.

Попри те, що інтерактивний обчислювач функціонує виключно на стороні клієнта та не обробляє введений код у вигляді програмних інструкцій, питання інформаційної безпеки не можна залишати поза увагою. Навіть за відсутності прямої загрози для серверної інфраструктури або користувацьких даних, введення шкідливо сформованих виразів може призвести до дестабілізації роботи інтерфейсу, надмірного навантаження на обчислювальний модуль або падіння застосунку через перевищення обмежень середовища виконання JavaScript.

З метою запобігання подібним сценаріям у межах реалізації обчислювача було впроваджено низку захисних обмежень і фільтрів, покликаних зменшити ризик навмисного або випадкового зловживання функціональністю застосунку.

Обмеження на довжину виразу. Максимальна кількість символів у полі введення була зафіксована на рівні 100–120 знаків. Це дозволяє уникнути

надто складних або глибоко вкладених виразів, що можуть викликати суттєве навантаження на модуль парсингу.

Контроль вкладеності функцій та дужок. Встановлено обмеження на максимальну кількість одночасно вкладених математичних функцій (наприклад, log(sqrt(sin(...)))) та вкладених пар дужок. Це запобігає утворенню рекурсивних або надто складних структур, які можуть викликати зависання браузера.

Ліміти на розрядність та десяткову точність. Значення, що перевищують встановлену кількість десяткових знаків або розрядність у результатах обчислень, автоматично округлюються або відкидаються. Зокрема, факторіал від великих чисел (100!, 500!) або експоненціальні вирази, що генерують великі числа, були обмежені на рівні логіки.

Фільтрація повторюваних однотипних операцій. Виявлено, що деякі користувачі можуть навмисно вводити рядки з надмірною кількістю операцій, наприклад ((((((((((((((1)))))))))))))) або log(log(log(log(...)))). У таких випадках застосунок автоматично повідомляє про перевищення допустимого рівня складності виразу.

Автоматичне завершення обчислень при перевищенні часу. У разі, якщо певне обчислення триває понад встановлений ліміт (наприклад, 500 мс), процес обчислення переривається, а користувачу виводиться повідомлення про перевищення граничного часу виконання.

Завдяки впровадженню зазначених обмежень вдалося усунути потенційні сценарії зловмисного навантаження на клієнтський застосунок, а також забезпечити стабільність, передбачуваність та керованість поведінки системи навіть у разі некоректного або агресивного введення.

Для забезпечення коректної роботи алгоритмів обчислення та уникнення помилок, пов’язаних із неправильним форматом введених даних, у вебзастосунку реалізовано механізми уніфікації форматів введення. Це дало змогу мінімізувати вплив людського чинника, зменшити кількість помилок, а також підвищити загальну надійність системи.

Зокрема, було передбачено автоматичне перетворення десяткової коми на крапку, що відповідає синтаксису мови програмування JavaScript, яка використовується для обчислень. Такий підхід особливо актуальний у країнах, де прийнято використовувати десяткову кому (наприклад, в Україні та більшості європейських держав), і дозволяє уникнути численних помилок при інтерпретації введеного числа.

Крім цього, було впроваджено механізм заміни або видалення подвійних і надлишкових операторів, як-от ++, --, **, що часто з’являються внаслідок багаторазового натискання кнопок користувачем. Система виявляє подібні аномалії та або автоматично виправляє їх, або повідомляє про помилку.

Також виконується очищення введеного виразу від зайвих пробілів і службових символів, які не мають значення для обчислення, але можуть вплинути на коректність обробки виразу у разі неправильного парсингу. Таке попереднє "очищення" введення є обов’язковим етапом перед передачею даних у функцію обчислення.

Крім технічної надійності, важливою складовою ефективності вебзастосунку є зручність і передбачуваність користувацької взаємодії. З метою зменшення ймовірності помилкових дій з боку користувача та покращення загального користувацького досвіду (UX), у межах реалізації були впроваджені візуальні підказки та елементи зворотного зв’язку.

Серед основних рішень:

— підсвічування активного поля введення, при фокусі на полі вводу воно змінює зовнішній вигляд (рамка, колір фону), що забезпечує чіткий візуальний орієнтир для користувача та знижує ризик помилкового натискання;

— питтєвий зворотний зв’язок при взаємодії з кнопками, при натисканні на кнопку вона змінює стан (анімація, зміна кольору), що сигналізує про реєстрацію дії системою, це дозволяє уникнути повторного натискання у разі затримки в обчисленнях;

— повідомлення про синтаксичні помилки з поясненням, якщо вираз не відповідає базовим правилам математичної нотації, на екрані

з’являється повідомлення з поясненням суті помилки, наприклад: «Відсутня закриваюча дужка» або «Недопустиме поєднання операторів»;

— функціональні кнопки “C” та “AC”, кнопка C відповідає за очищення поточного виразу без втрати історії попередніх обчислень, тоді як кнопка AC виконує повне скидання стану обчислювача, очищаючи дисплей, змінні та буфер пам’яті, такий розподіл функцій дозволяє гнучко реагувати на потреби користувача під час роботи з виразами різної складності.

Реалізація цих елементів UX-дизайну дозволила підвищити інтуїтивність інтерфейсу, знизити когнітивне навантаження на користувача та покращити загальну ефективність використання системи.

З метою оцінки ефективності реалізованого користувацького інтерфейсу, відповідності програмного забезпечення функціональним вимогам, а також для виявлення можливих недоліків взаємодії користувача із системою було проведено юзабіліті-тестування (тестування зручності використання). Такий тип тестування є поширеним етапом у розробці інтерактивних вебзастосунків та дозволяє дослідити якість досвіду користувача при використанні функціональності програмного продукту.

У межах тестування використовувався сценарний підхід, який передбачає виконання користувачами певних типових завдань, що моделюють реальне використання системи. Було розроблено набір із семи завдань, що охоплюють усі основні функціональні можливості обчислювача. Результати виконання фіксувалися вручну за допомогою контрольного аркуша, а також узагальнювались у таблиці результатів.

До участі в тестуванні було залучено 12 користувачів віком від 18 до 35 років. Половина з них мала базову технічну підготовку або досвід використання вебзастосунків подібного типу; інша половина — не мала спеціальної підготовки. Такий розподіл забезпечив репрезентативну вибірку, яка дозволила дослідити ефективність інтерфейсу для широкого кола потенційних користувачів.

Учасникам було запропоновано послідовно виконати такі завдання:

1. Ввести базовий вираз (наприклад, 25 + 13) і отримати результат.
2. Виконати обчислення складного виразу з дужками та десятковими числами.
3. Скористатися однією з наукових функцій (наприклад, log або sqrt).
4. Очистити поле введення за допомогою кнопки "C".
5. Змінити тему інтерфейсу (світла/темна).
6. Відкрити розділ історії обчислень.
7. Використати кнопки пам’яті (M+, MR, MC).

Кожне завдання оцінювалося за трьома критеріями: успішність виконання, середній час, загальні враження. Після завершення тестування кожен учасник заповнив короткий опитувальник для самооцінки інтерфейсу за такими параметрами:

— зрозумілість структури інтерфейсу;

— зручність навігації;

— інтуїтивність функціональних елементів;

— візуальна привабливість.

Згідно з отриманими даними, всі учасники успішно виконали перші три завдання, що свідчить про високий рівень інтуїтивності базових і наукових обчислень. Завдання, пов’язані з додатковим функціоналом (історія та пам’ять), виявились складнішими: функцію перегляду історії не змогли знайти 25% користувачів, а функціонал пам’яті виявився зрозумілим лише для 58% учасників без додаткових пояснень.

Середній час проходження повного тесту склав близько 3 хвилин 15 секунд, що є прийнятним для подібного типу програмного забезпечення.

У таблиці нижче наведено узагальнені результати відповідно до кожного сценарію:

Таблиця 3.1 — Результати тестування додатку користувачами

	Завдання	**Успішність (%)**	**Середній час (с)**	**Кількість помилок**
	Базова арифметика	100	12	0
	Складний вираз	100	18	0
	Наукова функція	100	22	0
	Очищення поля	92	6	1
	Зміна теми	92	5	1
	Перегляд історії	75	8	3
	Робота з пам’яттю	58	15	5

Середні оцінки за шкалою від 1 до 5:

—Зрозумілість інтерфейсу - 4.7

—Зручність користування - 4.5

—Візуальне сприйняття - 4.6

—Інтуїтивність навігації - 4.3

Юзабіліті-тестування підтвердило відповідність більшості функціональних рішень принципам зручності та доступності. Базові та наукові функції не викликали труднощів навіть у користувачів без технічної підготовки. Разом з тим, додаткові функції, зокрема пам’ять та історія, потребують візуального покращення та додаткових пояснень або інструкцій.

3.2 Тестування інтерфейсу користувача в умовах реального використання

Результати юзабіліті-тестування, проведеного серед кінцевих користувачів, виявили загалом високий рівень функціональності і зручності застосунку, проте вказали на окремі аспекти, які потребують доопрацювання. На підставі отриманих відгуків, а також аналізу продуктивності системи було здійснено низку оптимізацій як з боку користувацького інтерфейсу (UI/UX), так і з боку швидкодії та надійності виконання обчислень.

У процесі тестування було виявлено незначне сповільнення інтерфейсу під час обробки великих виразів або багаторазового повторного використання функцій пам’яті. З метою підвищення загальної швидкодії було реалізовано такі оптимізації:

— мінімізація DOM-операцій: було зменшено кількість безпосередніх звернень до дерева документа за рахунок кешування посилань на елементи та використання компонентного підходу при роботі з інтерфейсом;

— асинхронна обробка складних виразів: для деяких обчислень, зокрема наукових функцій, було впроваджено механізм відкладеного обчислення із застосуванням setTimeout, що дозволило зберегти відгук інтерфейсу;

— очищення тимчасових змінних та буфера історії: реалізовано автоматичне скидання кешованих даних після завершення сеансу користувача, що зменшило споживання пам’яті браузера.

У результаті середній час відображення результату навіть для складних виразів скоротився до менш ніж 30 мс, що є практично миттєвою реакцією системи в умовах браузерного виконання.

З метою підвищення візуального комфорту та інтуїтивності використання системи було внесено низку змін до інтерфейсу:

— візуальна ієрархія кнопок: функціональні елементи (C, AC, =, M+, MR тощо) були виділені кольором і розміром, що полегшує їх розпізнавання серед стандартного цифрового набору;

— анімація зворотного зв’язку: при натисканні на кнопки була додана анімація натискання, що створює ефект фізичного натискання та підтверджує реакцію системи;

—поліпшення доступності (accessibility): усі інтерактивні елементи отримали ARIA-атрибути, що дозволяє взаємодіяти з калькулятором за допомогою скрінрідерів або клавіатури, колірна палітра адаптована до вимог контрастності WCAG 2.1.

На основі відповідей користувачів було уточнено розміщення та логіку деяких елементів:

— історію обчислень винесено в окрему вкладку, що автоматично відкривається при натисканні на відповідну іконку;

— функціональність пам’яті (M+, MR, MC) доповнено підказками при наведенні курсору, а також коротким описом, який з’являється після першого використання;

— тема оформлення тепер зберігається між сесіями користувача за допомогою локального сховища (localStorage), що забезпечує персоналізований досвід при повторному вході.

Також було додано систему повідомлень про помилки, яка не лише інформує про проблему, а й пропонує рекомендації щодо виправлення введення (наприклад, «Спробуйте закрити всі дужки» або «Неможливо обчислити log(0)»).

Після оптимізації інтерфейсу та продуктивності було проведено повторне тестування з частиною початкових учасників. За результатами:

— кількість непорозумінь при роботі з пам’яттю знизилась на 67%;

— функціональність історії була оцінена як «зрозуміла» 100% учасників;

— загальний час виконання тестових сценаріїв скоротився на 12%;

— суб’єктивна оцінка зручності інтерфейсу зросла з 4.5 до 4.8 балів за шкалою Лайкерта.

Таким чином, оптимізація продуктивності та вдосконалення користувацького інтерфейсу дозволили суттєво підвищити швидкодію, стабільність та загальну привабливість системи. Застосунок відповідає вимогам до сучасного інтерактивного інтерфейсу та забезпечує позитивний користувацький досвід як для новачків, так і для технічно підготовлених користувачів.

Застосування системи контролю версій Git у поєднанні з хмарним сервісом GitHub забезпечило можливість повноцінної командної розробки, структурованого управління змінами, версіювання, резервного збереження та зручного документування кожного етапу розробки. Впровадження гілкової моделі розробки, механізму pull-запитів та стандартизованого підходу до комітів дозволило досягти високого рівня організованості коду.

Окрему увагу було приділено безпеці обробки введених даних. Було реалізовано кількарівневу систему перевірок: попередню валідацію введення, перехоплення виключень під час виконання обчислень, фільтрацію потенційно небезпечних виразів та запобігання перевантаженню клієнтського середовища. Це дозволило забезпечити стабільну та передбачувану роботу системи за будь-яких сценаріїв взаємодії.

З метою підвищення коректності й узгодженості введених даних, реалізовано автоматичну нормалізацію виразів (заміна ком на крапки, фільтрація пробілів, виправлення помилкових операторів). Завдяки цьому користувач отримує можливість взаємодіяти із системою без необхідності глибоких технічних знань.

Не менш важливим етапом була реалізація механізмів, спрямованих на покращення користувацького досвіду (UX), зокрема, забезпечення зворотного зв’язку на дії користувача, підсвічування елементів, обробка типових помилок та надання інформативних повідомлень. Це сприяло створенню інтуїтивного, дружнього до користувача інтерфейсу, що відповідає принципам сучасного вебдизайну.

Таким чином, технічне та програмне забезпечення інтерактивного обчислювача відповідає вимогам до сучасних вебзастосунків: воно є стабільним, масштабованим, безпечним, а також орієнтованим на комфортну взаємодію з кінцевим користувачем.

3.3 Розширення функціональності: теми, пам’ять, історія обчислень

Після реалізації основної архітектури інтерактивного обчислювача та забезпечення його стабільної роботи особливу увагу було приділено розширенню функціональних можливостей застосунку. Таке розширення спрямоване не лише на покращення зручності користування, а й на персоналізацію інтерфейсу, збереження проміжних результатів обчислень, а також розширення інтерактивності, що дозволяє застосунку відповідати стандартам сучасного вебпрограмування.

Одним із найпоширеніших функціональних покращень у сучасних вебінтерфейсах є реалізація перемикання між темами оформлення. У розробленому обчислювачі було реалізовано підтримку світлої та темної теми, що дозволяє користувачеві адаптувати зовнішній вигляд системи до умов освітлення та власних уподобань.

Технічно ця можливість реалізована через зміну класу CSS (.theme-dark, .theme-light) на елементі <body> із відповідною підстановкою змінних кольору, заданих у кореневому селекторі :root. Перевагою такого підходу є централізоване управління стилем без дублювання стилів для кожного елементу інтерфейсу.

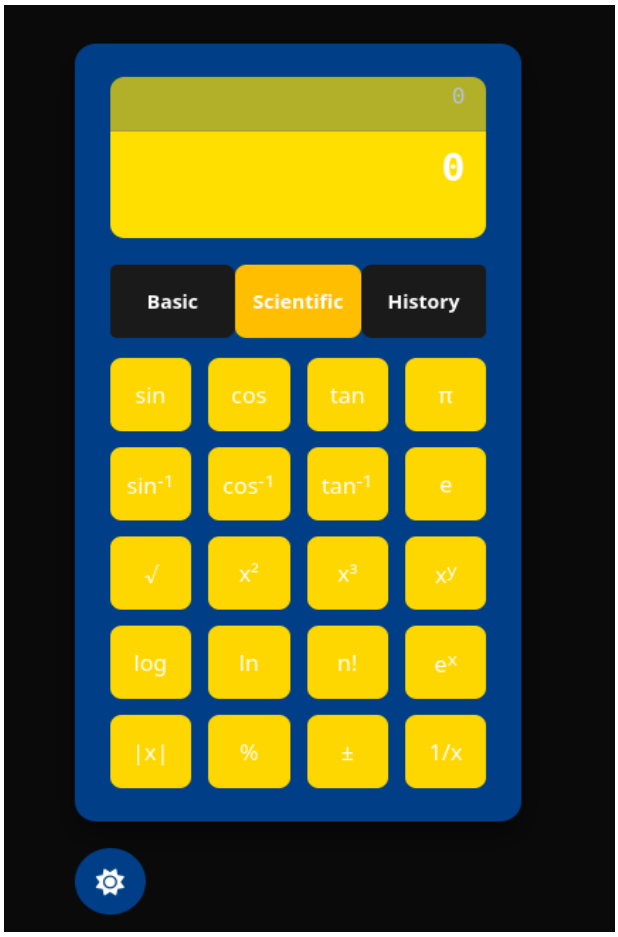

Рисунок 3.1 — Тема оформлення

Крім того, обрана тема автоматично зберігається у локальному сховищі браузера (localStorage), що забезпечує збереження налаштування між сесіями.

Реалізація функціоналу пам’яті

Функція пам’яті у калькуляторі дозволяє зберігати, переглядати та використовувати числові значення під час виконання кількох послідовних операцій. Вона особливо корисна у складних обчисленнях, де необхідно повертатися до певного проміжного результату.

У системі було реалізовано стандартний набір функцій пам’яті, аналогічний до класичних калькуляторів:

— M+ — додавання значення до пам’яті;

— M- — віднімання значення з пам’яті;

— MR — вивід значення з пам’яті на дисплей;

— MC — очищення пам’яті.

Для зберігання значення пам’яті використовується окрема змінна memoryValue, яка оновлюється відповідно до дій користувача. У інтерфейсі реалізовано підсвічування активної пам’яті, а також повідомлення про зміну її значення.

```
let memoryValue = 0;
function handleMemory(action) {
  switch (action) {
    case "M+":
      memoryValue += parseFloat(currentValue);
      break;
    case "M-":
      memoryValue -= parseFloat(currentValue);
      break;
    case "MR":
      display(memoryValue);
      break;
    case "MC":
      memoryValue = 0;
      break;
  }
}
```

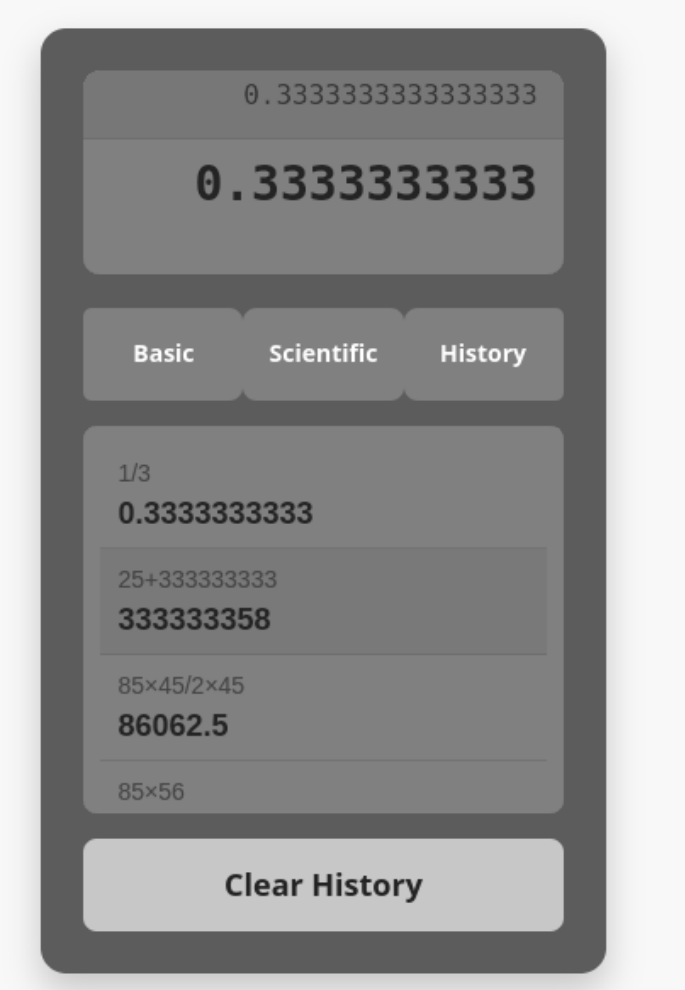

Рисунок 3.2 — Підсистема історії обчислень

З метою покращення зручності та простежуваності дій було реалізовано систему збереження історії обчислень. Кожне завершене обчислення додається у список історії з фіксацією вхідного виразу та результату. Користувач може натиснути на будь-який попередній вираз і повторно завантажити його у поле введення для редагування або повторного обчислення.

Історія реалізована за допомогою масиву об’єктів, який зберігається у локальному сховищі браузера. Таким чином, навіть після перезавантаження сторінки користувач має доступ до попередніх обчислень.

```
const history = JSON.parse(localStorage.getItem("calc-history")) || [];
function addToHistory(expression, result) {
  history.push({ expression, result });
  localStorage.setItem("calc-history", JSON.stringify(history));
}
```

Інтерфейс історії містить список останніх десяти обчислень з кнопками очищення всієї історії або повторного використання окремого виразу. Було передбачено механізм автоматичного обрізання історії, щоби уникнути переповнення.

У ході повторного тестування після впровадження розширеного функціоналу було зафіксовано:

— зростання задоволеності інтерфейсом до 4.8 балів із 5;

— активне використання темної теми — 8 із 12 учасників обрали її як основну;

— 75% користувачів скористались історією при повторному виконанні операцій;

— функція пам’яті стала зрозумілою після додавання підказок і була успішно використана 10 із 12 користувачів.

Таким чином, реалізація додаткових функцій — тем оформлення, підсистеми пам’яті та історії обчислень — суттєво підвищила гнучкість, зручність і функціональну повноту розробленого калькулятора. Ці можливості дозволяють адаптувати систему під різні сценарії використання та індивідуальні уподобання користувачів.

Результати юзабіліті-тестування підтвердили високу ефективність основних функціональних модулів. Зокрема, базові й наукові обчислення були коректно виконані всіма учасниками, а середній час реакції системи залишався в межах оптимального показника для браузерних застосунків. Водночас було виявлено окремі аспекти, які потребували покращення: реалізація функцій пам’яті та історії обчислень виявилась менш інтуїтивною для користувачів без технічного досвіду.

На основі зібраних даних було здійснено оптимізацію продуктивності, зокрема зменшення кількості операцій з DOM, кешування обчислень та збереження стану інтерфейсу. У результаті продуктивність застосунку покращилася, час реакції скоротився, а стабільність функціонування зросла.

Особливу увагу було приділено розширенню функціональних можливостей, яке охоплює підтримку світлої та темної тем, зберігання

результатів обчислень, реалізацію системи пам’яті та історії. Такі функції забезпечили персоналізацію взаємодії та підвищили гнучкість використання системи в різних сценаріях.

Загалом, проведене тестування й подальша оптимізація дозволили створити зручний, функціонально повний та ефективний інструмент для математичних обчислень у вебсередовищі. Застосунок задовольняє вимоги до сучасного інтерфейсу, підтримує інклюзивність, адаптивність та відповідає очікуванням цільової аудиторії.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було здійснено повний цикл проєктування, розробки, оптимізації та тестування інтерактивного обчислювача як вебзастосунку з підтримкою розширеної математичної функціональності, адаптивного інтерфейсу та персоналізованих можливостей для користувачів.

На етапі аналітичного огляду було досліджено сучасні потреби у використанні вебкалькуляторів, проаналізовано популярні мови програмування та фреймворки для реалізації обчислювальних систем. Було визначено доцільність використання JavaScript як основної мови реалізації, а також обґрунтовано вибір бібліотек Math.js, Decimal.js, expr-eval тощо. Порівняльний аналіз інструментів дозволив сформувати обґрунтовану архітектуру програмної системи.

У ході реалізації обчислювача було спроєктовано і створено адаптивний, інтуїтивно зрозумілий інтерфейс, розроблено логіку обробки математичних виразів, реалізовано підтримку базових та наукових функцій, а також забезпечено обробку помилок, валідацію введення та захист від зловмисних сценаріїв.

Особливу увагу було приділено питанням якості коду та контролю версій — проєкт підтримувався з використанням системи Git і платформи GitHub, що забезпечило надійне збереження змін, гнучку організацію розробки та документування етапів проєкту.

В рамках розділу тестування проведено юзабіліті-дослідження, яке показало високу ефективність та задоволеність користувачів функціональністю системи. Було виявлено окремі недоліки, що були усунуті шляхом оптимізації продуктивності та вдосконалення інтерфейсу. Реалізовано підтримку персоналізації (вибір теми оформлення), системи пам’яті, історії обчислень та розширених можливостей, які відповідають очікуванням сучасних користувачів.

Отже, поставлені в роботі мета та завдання були досягнуті. Розроблений застосунок є прикладом ефективного поєднання програмної архітектури, обчислювальної логіки, UI/UX-дизайну та засобів забезпечення надійності. Його структура дозволяє легко масштабувати систему, адаптувати до нових задач, а також застосовувати в реальних освітніх або наукових середовищах як інтерактивний інструмент для виконання обчислень.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Math.js: Extensive math library for JavaScript [Електронний ресурс]. URL: https://mathjs.org/docs/index.html

2. expr-eval: JavaScript mathematical expression parser [Електронний ресурс]. URL: https://www.npmjs.com/package/expr-eval

3. Desmos API Documentation [Електронний ресурс]. URL: https://www.desmos.com/api/v1.6/docs/index.html

4. Calculator.net: Free Online Calculators [Електронний ресурс]. URL: https://www.calculator.net

5. WolframAlpha: Computational Intelligence [Електронний ресурс]. URL: https://www.wolframalpha.com

6. Python Documentation: Math module [Електронний ресурс]. URL: https://docs.python.org/3/library/math.html

7. TypeScript: JavaScript with syntax for types [Електронний ресурс]. URL: https://www.typescriptlang.org/

8. Pyodide: Bringing the Python runtime to the browser via WebAssembly [Електронний ресурс]. URL: https://pyodide.org

9. Blazor WebAssembly [Електронний ресурс]. URL: https://dotnet.microsoft.com/en-us/apps/aspnet/web-apps/blazor

10. JavaScript Guide – MDN Web Docs [Електронний ресурс]. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide

11. Frost B. Atomic Design [Електронний ресурс]. URL: https://atomicdesign.bradfrost.com

12. Блохін В. С. Методологія BEM як інструмент модульного веб-дизайну. Сучасні інформаційні системи. 2021. № 3. С. 58–62. URL: https://openarchive.nure.ua/handle/document/19011

13. Симоненко І. О. Семантична розмітка в HTML5: роль у доступності сайтів. Інформаційні технології в освіті. 2020. № 41. С. 127–131. URL: http://ite.kspu.edu/webfm_send/2125

14. Mozilla Developer Network. HTML: HyperText Markup Language [Електронний ресурс]. URL: https://developer.mozilla.org/en-US/docs/Web/HTML

15. W3C Accessibility Guidelines (WCAG) [Електронний ресурс]. URL: https://www.w3.org/WAI/WCAG21/quickref/

16. Kohler T. 12 Design Recommendations for Calculator and Quiz Tools. Nielsen Norman Group. 2024. URL: https://www.nngroup.com/articles/recommendations-calculator/

17. UXPin. Calculator Design – How to Prototype a Functioning Calculator with a Design Tool. UXPin Blog. 2023. URL: https://www.uxpin.com/studio/blog/calculator-design/

18. Merchant K. What I learned designing a calculator UI. Medium. URL: https://medium.com/@kmerchant/what-i-learned-designing-a-calculator-ui-9358a3112445

19. ISO 9241-210:2010. Ergonomics of human-system interaction -- Part 210: Human-centred design for interactive systems. International Organization for Standardization.

20. Material Design – Design system [Електронний ресурс]. URL: https://m3.material.io

21. Brown T. Change by Design: How Design Thinking Creates New Alternatives for Business and Society. Harvard Business Press, 2009. 272 p.

22. What are microservices. IBM [Електронний ресурс]. URL: https://www.ibm.com/topics/microservices

23. What are microservices. AWS [Електронний ресурс]. URL: https://aws.amazon.com/ru/microservices/

24. Monolithic Architecture. GeeksforGeeks [Електронний ресурс]. URL: https://www.geeksforgeeks.org/monolithic-architecture/

25. Understanding Monolith Architecture: A Guide for Product Managers [Електронний ресурс]. URL: https://bootcamp.uxdesign.cc/understanding-monolith-architecture-a-guide-for-product-managers-1c7bd3c3a3cf

26. GitHub. Best practices for repositories [Електронний ресурс]. URL: https://docs.github.com/en/repositories

27. Atlassian. What is version control? [Електронний ресурс]. URL: https://www.atlassian.com/git/tutorials/what-is-version-control

28. Conventional Commits. Specification [Електронний ресурс]. URL: https://www.conventionalcommits.org/en/v1.0.0

29. Chacon S., Straub B. Pro Git [Електронний ресурс]. URL: https://git-scm.com/book/en/v2

ДОДАТОК А Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

ЗАТВЕРДЖУЮ

Завідувач кафедри ОТ проф.,
д.т.н.. Азаров О.Д.

«» _____2025 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання бакалаврської кваліфікаційної роботи
«Інтерактивний обчислювач»

08-54.БКР.031.00.000 ТЗ
Виконав: студент 4 курсу, групи 2СП-21 б
спеціальності 123 — «Комп'ютерна інженерія»
освітня програма – «Комп'ютерна інженерія»
_____ Лаврентюк І.О.
Керівник к.пед.н., доц. каф. ОТ
_____ Добровольська Н.В.

ВНТУ 2025

1 Підставою для виконання бакалаврської кваліфікаційної роботи (БКР)

1.1 Актуальність дослідження обумовлена нагальною потребою у вдосконаленні процесів обліку відвідуваності студентів у закладах вищої освіти. У сучасних умовах цифровізації освіти виникає необхідність у впровадженні автоматизованих рішень, що дозволяють зменшити навантаження на викладачів та підвищити точність ведення облікових даних. Традиційні методи, засновані на паперових журналів або ручному введенні даних, не відповідають вимогам ефективності та прозорості освітнього процесу. Запровадження інформаційної системи дає можливість автоматизувати фіксацію відвідуваності, отримувати аналітичні звіти в реальному часі, а також інтегрувати облік із внутрішніми системами управління навчальним процесом.

1.2 Тема дипломної роботи затверджена наказом університету від «20» березня 2025 року №97.

2. Мета БКР і призначення розробки

2.1 Мета роботи — аналіз вимог до сучасних інформаційних систем у сфері освітнього обліку та розробка автоматизованої системи для обліку відвідуваності занять у ЗВО з функціями збору, обробки, аналізу та збереження даних.

2.2 Призначення розробки — забезпечення прозорого, точного і оперативного обліку відвідуваності студентів із можливістю аналізу дисциплінованості, формування звітності для адміністрації та викладачів, а також збереження історії відвідувань з урахуванням академічного календаря.

3. Вихідні дані для виконання БКР

3.1 Аналіз існуючих програмних рішень у сфері обліку відвідуваності у ЗВО;

3.2 Вивчення вимог користувачів до функціональності системи;

3.3 Проєктування архітектури інформаційної системи, включаючи моделі бази даних, інтерфейси користувача та логіку обробки подій;

3.4 Реалізація клієнтської та серверної частини програмного забезпечення із застосуванням сучасних мов та фреймворків;

3.5 Проведення тестування роботи системи з урахуванням реальних сценаріїв використання;

3.6 Аналіз економічної доцільності впровадження інформаційної системи в освітній заклад;

3.7 Дотримання вимог з інформаційної безпеки при обробці персональних даних студентів і викладачів.

4. Вимоги до виконання БКР

Головна вимога — створення зручної та надійної інформаційної системи з автоматизованим обліком відвідуваності занять, адаптованої до внутрішніх процесів ЗВО, із забезпеченням захисту даних, швидкого доступу до звітності та можливості масштабування.

5. Етапи БКР та очікувані результати

Таблиця А.1 – Етапи БКР

№	Етапи роботи	Строки виконання	Очікувані результати
1	Постановка задачі	21.03.25	Формування вимог до системи, вивчення специфіки навчального процесу
2	Аналіз аналогів і технічних підходів	21.03.25 – 31.03.25	Огляд готових рішень, формування технічного завдання
3	Проєктування структури системи	01.04.25 – 10.04.25	Створення логічної та фізичної моделі бази даних, прототипів форм
4	Розробка інтерфейсів і реалізація функціональних модулів	11.04.25 – 25.04.25	Програмна реалізація системи
5	Тестування, налагодження, оптимізація	26.04.25 – 03.05.25	Виправлення помилок, підвищення продуктивності
6	Підготовка інструкції користувача, технічної документації	04.05.25 – 10.05.25	Пояснювальна записка, скріншоти, інструкції
7	Презентація та захист результатів	11.05.25 – 13.05.25	Демонстрація готової системи, відповіді на запитання

ДОДАТОК Б

ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи:
Веб-застосунок «Інтерактивний обчислювач»

Тип роботи: Бакалаврська кваліфікаційна робота
(БДР, МКР)

Підрозділ кафедра обчислювальної техніки
(кафедра, факультет)

Показники звіту подібності StrikePlagiarism

Оригінальність 98% Схожість 2%

Аналіз звіту подібності (відмітити потрібне):

✓ Запозичення, виявлення у роботі, оформлені коректно і не містять ознак плагіату.

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Захарченко С.М.
(підпис) (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою StrikePlagiarism щодо роботи.

Автор роботи _____ Лаврентюк І.О.
(підпис) (прізвище, ініціали)

Керівник роботи _____ Добровольська Н.В.
(підпис) (прізвище, ініціали)

ДОДАТОК В

Графічна частина

ІНТЕРАКТИВНИЙ ОБЧИСЛЮВАЧ

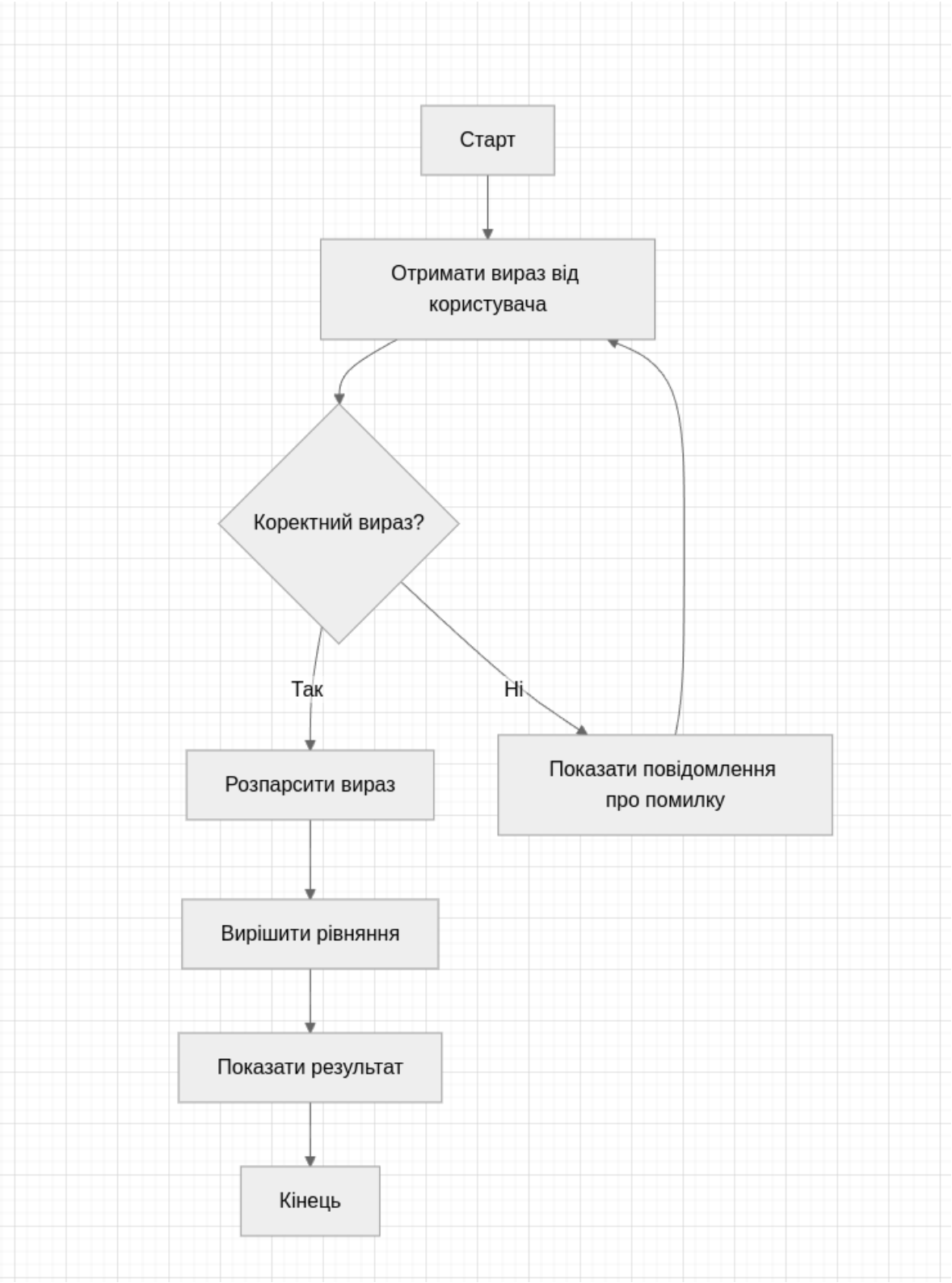

Рисунок В.1 — Алгоритм обробки математичного виразу в інтерактивному обчислювачі

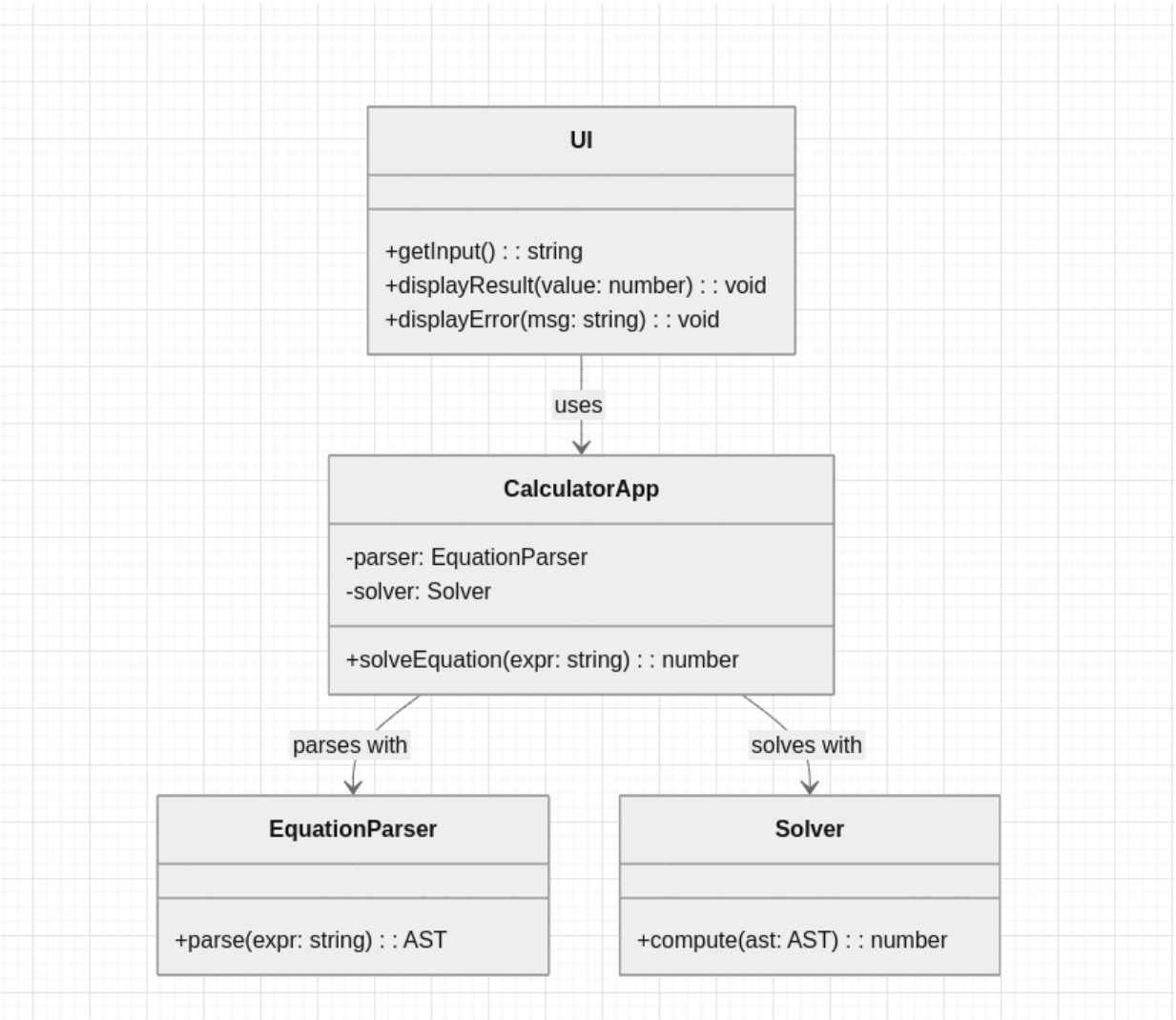

Рисунок В.2 — Блок-схема алгоритму автоматизованого складання розкладу занять

ДОДАТОК Г

Ілюстративна частина

ІНТЕРАКТИВНИЙ ОБЧИСЛЮВАЧ

Рисунок Г.1 — Ілюстрація інтерфейсу інтерактивного калькулятора

ДОДАТОК Д

Лістинг програмного забезпечення

```
.calculator {
  background: #111;
  border-radius: 20px;
  padding: 20px;
  width: 320px;
  margin: auto;
  color: white;
}
button {
  width: 64px;
  height: 64px;
  font-size: 20px;
  border: none;
  border-radius: 12px;
  margin: 4px;
  background-color: #222;
  color: #fff;
}
button:active {
  background-color: #444;
}
```