Міністерство освіти і науки України
Український державний університет науки і технологій

Комп'ютерні технології і системи

Комп'ютерні інформаційні технології

Пояснювальна записка

до кваліфікаційної роботи
магістра

на тему: Дослідження впливу стандарту ES6 у програмуванні мовою JavaScript на можливості супроводу коду

за освітньою програмою: 12 Інженерія програмного забезпечення

зі спеціальності: 121 Інженерія програмного забезпечення

Виконав: студент групи ПЗ2422: _____ Михайло СВИРИДОВ

Керівник: _____ Віктор ШАРАВАРА

Нормоконтролер: _____ Світлана ВОЛКОВА

Засвідчую, що у цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____

Дніпро – 2026

Ministry of Education and Science of Ukraine
Ukrainian State University of Science and Technologies

Computer technologies and systems

Computer information technology

Explanatory Note
to Master's Thesis

on the topic: Research of the impact of the ES6 standard in JavaScript programming language on code maintainability

according to educational curriculum 12 Software engineering

in the Speciality: 121 Software engineering

Done by the student of the group: PZ2422 Mykhailo SVYRYDOV

Scientific Supervisor: Viktor SHARAVARA

Normative controller : Svitlana VOLKOVA

Dnipro – 2026

Міністерство освіти і науки України
Український державний університет науки і технологій

Факультет: Комп'ютерні технології і системи
Кафедра: Комп'ютерні інформаційні технології
Рівень вищої освіти: магістр
Освітня програма: «12 Інженерія програмного забезпечення»
Спеціальність:«121 Інженерія програмного забезпечення»
(шифр та назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри КІТ
_____ Вадим ГОРЯЧКІН
(підпис) (Ім'я ПРІЗВИЩЕ)
Дата _____

ЗАВДАННЯ

на кваліфікаційну роботу магістра
(ступінь вищої освіти)

студенту Свиридову Михайлу Олександровичу
(Прізвище, Ім'я, По батькові)

1. Тема роботи: Дослідження впливу стандарту ES6 у програмуванні мовою JavaScript на можливості супроводу коду
Керівник роботи: PhD, ст. викладач Шаравара Віктор Володимирович
(Прізвище, Ім'я, По батькові, науковий ступінь, вчене звання)

затверджені наказом від "02" жовтня 2025 р. №1401

2. Строк подання студентом роботи: 09.01.2026р.
3. Вихідні дані до роботи: розроблений веб-додаток, вихідний код JavaScript у стилях ES5 та ES6, парні приклади реалізацій для експериментального порівняння, результати статичного аналізу у вигляді показників складності та підтримуваності, а також згенеровані графічні звіти.
4. Зміст пояснювальної записки (перелік питань, які потрібно опрацювати):
4.1 Аналіз сучасних підходів до оцінювання підтримуваності javascript-коду
4.2 Обґрунтування методики дослідження впливу стандарту ES6 на підтримуваність javascript-коду
4.3 Проєктування й розробка інструментального забезпечення для дослідження впливу стандарту ES6 на підтримуваність javascript-коду
4.4 Дослідження впливу стандарту es6 на показники супроводжуваності коду:

2026

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень):

5.1 Презентація

5.2 Демонстраційне відео

2026

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Вступ	01.10.25 – 03.10.25	
2	Аналіз сучасного стану дослідження проблеми за науковими літературними джерелами	04.10.25 – 10.10.25	
3	Аналіз сучасного стану програмно-апаратного забезпечення	11.10.25 – 17.10.25	
4	Постановка задачі, технічне завдання	18.10.25 – 05.11.25	30%
5	Узгодження та затвердження ТЗ	06.11.25 – 10.11.25	
6	Виконання досліджень	11.11.25 – 20.12.25	60%
7	Оформлення пояснювальної записки	21.12.25 – 9.01.26	
8	Розробка демонстраційних матеріалів	9.01.26 – 15.01.26	100%
9	Подання кваліфікаційної роботи до кафедри	16.01.26	
10	Захист кваліфікаційної роботи на засіданні Екзаменаційної комісії	20.01.26	

Студент _____ Михайло СВИРИДОВ
(підпис) (Ім'я ПРІЗВИЩЕ)

Керівник роботи _____ Віктор ШАРАВАРА
(підпис) (Ім'я ПРІЗВИЩЕ)

2026

РЕФЕРАТ

Пояснювальна записка містить 169 сторінок основного тексту, 20 рисунків, 6 таблиць, 4 додатки, список використаних джерел із 8 найменувань; складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатків.

Об’єкт дослідження – процеси оцінювання складності та підтримуваності програмного забезпечення на рівні вихідного коду JavaScript.

Мета роботи – дослідити вплив стандарту ES6 на підтримуваність JavaScript-коду шляхом кількісного порівняння показників складності ES5- та ES6-реалізацій за допомогою розробленого веб-інструменту «Code Complexity Lab».

Методи дослідження: аналіз публікацій з якості ПЗ та стандартів ECMAScript, адаптація показників Maintainability Index, когнітивної й цикломатичної складності та показників Гелстеда до JavaScript, статичний аналіз коду на основі AST (Babel), експериментальне порівняння парних реалізацій ES5/ES6, методи математичної статистики для перевірки значущості результатів.

Результати й новизна: розроблено односторінковий веб-застосунок на базі Vite та Pug, який за один прохід AST обчислює узгоджений набір показників підтримуваності JavaScript-коду й формує наочні звіти; на вибірці зі 100 парних прикладів показано кількісний ефект переходу від ES5 до ES6 (зростання Maintainability Index та зменшення когнітивної складності за незмінної функціональності).

Практичне значення роботи полягає у можливості використання «Code Complexity Lab» у навчальному процесі, під час code review та попереднього аудиту якості JavaScript-проєктів; застосунок працює локально в браузері та не потребує передачі коду на сервер.

Ключові слова: JavaScript, ES5, ES6, показники складності, Maintainability Index, когнітивна складність, цикломатична складність, показники Гелстеда.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАК, СКОРОЧЕНЬ І ТЕРМІНІВ

MI – Maintainability Index, індекс підтримуваності програмного коду (0–100)

CC – Cognitive Complexity, когнітивна складність за специфікацією SonarSource v1.7

G – Cyclomatic Complexity, цикломатична складність МакКейба

V – Halstead Volume, обсяг програми за Гелстедом

LOC – Lines of Code, кількість логічних рядків коду без коментарів

AST – Abstract Syntax Tree, абстрактне синтаксичне дерево

JS – JavaScript, мова програмування JavaScript

ES5 – ECMAScript 5, редакція стандарту мови JavaScript 2009 року

ES6 – ECMAScript 2015, шоста редакція стандарту мови JavaScript

IDE – Integrated Development Environment, інтегроване середовище розроблення

ПЗ – програмне забезпечення

ВСТУП

Стрімке поширення веб-технологій призвело до того, що JavaScript став фактично стандартною мовою для клієнтської логіки та значної частини серверних застосунків. При цьому величезні обсяги існуючого коду продовжують підтримуватися у стилі ECMAScript 5, тоді як сучасні проєкти дедалі активніше використовують можливості ECMAScript 6 та наступних редакцій стандарту. Співіснування двох стилів кодування в одному технологічному стеку ускладнює супровід, рефакторинг та оцінку технічного боргу, оскільки рішення щодо переходу на нові синтаксичні конструкції часто приймаються інтуїтивно, без кількісного обґрунтування.

У програмній інженерії проблема підтримуваності коду є ключовою для керування вартістю життєвого циклу програмного продукту. Зміни в кодовій базі, виправлення помилок та еволюційний розвиток функціональності безпосередньо залежать від зрозумілості та структурної складності програмних модулів. Для вимірювання цих характеристик застосовуються формалізовані показники, зокрема Maintainability Index, цикломатична складність, метрики Гелстеда та когнітивна складність, які дозволяють перетворити суб'єктивне враження від коду на кількісні показники. Однак існуючі інструменти аналізу складності коду, як правило, орієнтовані на серверну обробку або інтеграцію з великими платформами аналізу коду, що ускладнює їх використання для цілеспрямованих експериментальних досліджень впливу конкретних мовних конструкцій.

Особливий інтерес становить когнітивна складність як показник, що фокусується не на кількості можливих шляхів виконання, а на зусиллях, необхідних програмісту для розуміння коду. Вона покликана точніше відображати ментальне навантаження, пов'язане з вкладеністю коду, розгалуженнями та нетривіальними конструкціями керування потоком виконання, і тому є природним інструментом для аналізу читабельності та підтримуваності вихідного коду JavaScript.

Водночас у науковій та інженерній літературі недостатньо досліджені питання кількісного порівняння підтримуваності стилів ES5 та ES6 при збереженні однакової

алгоритмічної суті коду. Відсутні загальноприйняті методики побудови репрезентативних вибірок парних фрагментів коду, що відрізняються лише використовуваним стандартом.

Тому розробка веб-інструменту, який у браузері обчислює показники підтримуваності коду на основі синтаксичного дерева та дозволяє експериментально порівнювати варіанти реалізацій ES5 та ES6, а також дослідження впливу використання нових конструкцій стандарту ES6 на підтримуваність JavaScript-коду є актуальним завданням для програмної інженерії.

Робота узгоджується з сучасними напрямами досліджень у галузі якості програмного забезпечення та автоматизованого аналізу коду.

Об'єкт дослідження – процеси оцінювання та забезпечення підтримуваності програмного забезпечення, реалізованого мовою JavaScript упродовж життєвого циклу.

Предмет дослідження – методи та інструментальні засоби кількісної оцінки впливу синтаксичних конструкцій стандарту ES6 порівняно з ES5 на показники підтримуваності та когнітивної складності JavaScript-коду.

Метою роботи є дослідження впливу стандарту ES6 на підтримуваність JavaScript-коду шляхом кількісного порівняння показників складності ES5- та ES6-реалізацій за допомогою розробленого веб-інструменту «Code Complexity Lab».

Методи дослідження ґрунтуються на аналізі наукових джерел з якості програмного забезпечення та стандартів ECMAScript, адаптації показників Maintainability Index, когнітивної й цикломатичної складності та показників Гелстеда до JavaScript, проєктуванні та реалізації веб-застосунку з використанням статичного аналізу абстрактного синтаксичного дерева JavaScript-програм, а також на експериментальному порівнянні парних прикладів коду ES5 та ES6 із застосуванням методів математичної статистики для перевірки значущості результатів.

Наукова новизна роботи полягає у поєднанні інтегрального показника Maintainability Index із когнітивною та цикломатичною складністю й показниками Гелстеда в єдиному браузерному інструменті аналізу підтримуваності JavaScript-коду, а

також в отриманні кількісних доказів покращення підтримуваності коду при використанні конструкцій стандарту ES6 порівняно з ES5 на репрезентативній вибірці парних реалізацій.

Практичне значення одержаних результатів полягає в тому, що веб-застосунок Code Complexity Lab може використовуватися розробниками для оперативної оцінки підтримуваності JavaScript-коду та планування рефакторингу, а також у навчальному процесі як наочний засіб демонстрації впливу різних синтаксичних конструкцій на показники складності та підтримуваності програм.

Основні положення та результати дослідження доповідалися та обговорювалися на XIX Міжнародній науково-практичній конференції «Сучасні інформаційні та комунікаційні технології на транспорті, в промисловості і освіті» (Дніпро, 18-19 грудня 2025 р.), де було опубліковано тези доповіді «Дослідження впливу стандарту ES6 у програмуванні мовою JavaScript на можливості супроводу коду».

РОЗДІЛ 1 АНАЛІЗ СУЧАСНИХ ПІДХОДІВ ДО ОЦІНЮВАННЯ ПІДТРИМУВАНОСТІ JAVASCRIPT-КОДУ

1.1 Підтримуваність і життєвий цикл програмного забезпечення

Однією з ключових характеристик якості програмного забезпечення є підтримуваність, тобто придатність програмної системи до ефективного внесення змін, виправлення помилок, адаптації до нових вимог та середовищ виконання програмістами та іншими фахівцями. Підтримуваність безпосередньо впливає на сумарні витрати протягом життєвого циклу програмного продукту: для систем із тривалим циклом життя витрати на супровід часто перевищують початкові витрати на розроблення.

У класичних моделях життєвого циклу програмного забезпечення (каскадна, спіральна, інкрементна, гнучкі методології) супровід розглядається як окремий етап, проте на практиці зміни вносяться протягом усього часу експлуатації системи. На етапі проєктування та реалізації закладаються властивості коду, які або спрощують подальший супровід (чітка структура, зрозумілі залежності, однозначна логіка), або, навпаки, призводять до накопичення технічного боргу. Відповідно, завдання вимірювання підтримуваності коду є важливою складовою забезпечення якості програмного забезпечення.

У сучасних підходах до оцінювання якості ПЗ підтримуваність розглядається не лише як організаційний аспект процесу розроблення, але й як характеристика вихідного коду, що може бути описана кількісними показниками. Це відкриває можливість порівнювати різні стилі програмування, технології та стандартні бібліотеки не тільки за функціональністю, а й за їх впливом на складність подальшого супроводу.

1.2 Показники складності та підтримуваності програмного коду

Для того щоб перейти від суб’єктивного враження про «зручність» коду до формалізованої оцінки, у програмній інженерії застосовують систему кількісних показників. Вони описують різні аспекти складності програмних модулів: структуру

графа управління, словниковий обсяг програми, вкладеність умов, розгалуженість тощо. На основі таких показників формуються інтегральні оцінки підтримуваності.

Одним із базових структурних показників є цикломатична складність [1]. Вона відображає кількість незалежних шляхів виконання через програмний модуль та обчислюється на основі графа управління. Зростання цикломатичної складності ускладнює тестування та розуміння коду, оскільки кожна додаткова розгалужена конструкція збільшує кількість можливих сценаріїв виконання.

Для опису обсягу та інформаційної насиченості програми широко застосовуються показники Гелстеда, що базуються на кількості унікальних операторів та операндів і загальній кількості їх входжень. Вони дозволяють оцінити умовний словниковий запас програми, її «розмір» з погляду логічних конструкцій, а також вивести похідні показники, пов’язані з трудомісткістю реалізації й потенційною помилковістю. Показник обсягу програми за Гелстедом визначається як:

$$V = (N_1 + N_2)\cdot log_2(n_1 + n_2), \quad (1.1)$$

де N_1 — загальна кількість операторів;

N_2 — загальна кількість операндів;

n_1 — кількість унікальних операторів;

n_2 — кількість унікальних операндів;

На основі структурних і об’ємних показників сформовано інтегральний індекс підтримуваності (Maintainability Index) [2], який агрегує інформацію про складність, обсяг коду та його логічну структуру в єдине числове значення з фіксованим діапазоном. Це дозволяє порівнювати між собою різні модулі та відстежувати зміну підтримуваності в процесі рефакторингу. Індекс підтримуваності обчислюється у

нормалізованому до діапазону [0; 100] вигляді за класичною формулою Оман–Гагемайстер у модифікації, що використовується в Microsoft Visual Studio:

$$MI = max(0, \frac{171-5{,}2\cdot ln(V)-0{,}23\cdot G-16{,}2\cdot ln(LOC)}{171} \cdot 100), \quad (1.2)$$

де V — показник обсягу за Гелстедом;

G — цикломатична складність модуля;

LOC — кількість логічних рядків коду.

Окремий напрям представляє когнітивна складність як показник, орієнтований на оцінювання зусиль, необхідних розробнику для розуміння коду. На відміну від цикломатичної складності, когнітивна складність враховує не лише кількість розгалужень, але й фактори, що збільшують ментальне навантаження: глибоку вкладеність коду, нетривіальні комбінації умов, розірваність логіки. Низька когнітивна складність відповідає більш читабельному та передбачуваному коду, а отже, кращій підтримуваності.

Водночас у літературі відзначаються й обмеження традиційних показників складності. Зокрема, цикломатична складність не враховує читабельність і стилістичні особливості коду, а показники Гелстеда чутливі до стилю найменування та вибору елементарних операцій. Індекс підтримуваності залежить від конкретної редакції формули та набору вхідних показників, а когнітивна складність, попри орієнтацію на ментальне навантаження, залишається емпіричною моделлю, що потребує додаткових експериментальних перевірок. Це підтверджує доцільність не лише використання окремих показників, а й порівняльних досліджень на реальних прикладах коду.

Попри зазначені обмеження, поєднання цих показників дозволяє отримати більш повне уявлення про підтримуваність коду, ніж використання лише одного критерію. У контексті JavaScript це особливо актуально через високу гнучкість мови,

різноманітність стилів програмування та наявність як застарілих, так і сучасних підходів у межах одного проєкту.

1.3 Підходи до статичного аналізу коду та обробки JavaScript

Для практичного обчислення показників складності та підтримуваності розроблено низку інструментів статичного аналізу, які автоматично обробляють вихідний код і визначають структурні й інтегральні показники. Частина з них є багатомовними платформами аналізу якості коду, інші орієнтовані безпосередньо на JavaScript та фронтенд-розробку.

Ключовим елементом статичного аналізу є побудова абстрактного синтаксичного дерева (AST), що відображає структуру програмного коду у вигляді ієрархії вузлів, які відповідають операторам, виразам, оголошенням змінних, функцій тощо. На основі AST можна:

— відновити структуру графа управління для розрахунку цикломатичної складності;

— підрахувати кількість операторів та операндів для обчислення показників Гелстеда;

— оцінити вкладеність і складність умов для визначення когнітивної складності;

— фільтрувати службові конструкції та підраховувати логічні рядки коду без коментарів.

У випадку JavaScript складність статичного аналізу підвищується через наявність декількох версій стандарту ECMAScript, підтримку розширень (JSX, TypeScript), а також специфічні синтаксичні конструкції (функції-стріли, деструктуризація, шаблонні рядки тощо). Для коректної обробки сучасного JavaScript коду широко застосовуються спеціалізовані парсери, які будують AST з урахуванням різних діалектів мови.

1.4 Аналіз сучасних інструментів оцінювання складності JavaScript-коду

Для практичного застосування показників складності та підтримуваності розроблено низку інструментів статичного аналізу, які автоматично обробляють вихідний код і обчислюють структурні й інтегральні показники. Частина з них є

багатомовними платформами аналізу якості коду, інші орієнтовані безпосередньо на JavaScript та фронтенд-розробку.

Однією з найпоширеніших платформ є SonarQube [3], яка підтримує аналіз JavaScript та інших мов і обчислює, зокрема, цикломатичну складність, когнітивну складність, обсяг коду, дублювання та пов’язані з ними показники якості. У межах цієї платформи було запропоновано показник когнітивної складності, специфікація якого доступна в окремому документі SonarSource [4] і орієнтована на оцінювання зрозумілості структури коду для розробника. SonarQube зазвичай розгортається як серверний сервіс, інтегрується з системами безперервної інтеграції та використовується для контролю якості великих репозиторіїв, але не надає простого браузерного інтерфейсу для порівняння окремих фрагментів коду в альтернативних стилях запису.

Для повсякденного контролю якості JavaScript-коду широко застосовується ESLint [5] – розширюваний лінтер, який виявляє порушення стилю, потенційні помилки й підвищену складність коду. Зокрема, правило complexity дозволяє обмежувати допустиму цикломатичну складність функцій, попереджаючи про перевищення заданого порогу, що прямо пов’язано з підтримуваністю та тестованістю коду. ESLint легко інтегрується у процес збирання (Vite, Webpack тощо) та в редактори коду, проте його основна роль полягає у виявленні проблемних конструкцій у поточному проєкті, а не в наданні експериментального середовища для систематичного порівняння варіантів ES5/ES6 з кількісною візуалізацією показників.

Також існують хмарні платформи контролю якості коду, наприклад Code Climate [6], які агрегують результати різних аналізаторів і надають інтегровані показники підтримуваності, складності, дублювання та технічного боргу для репозиторіїв з JavaScript-кодом. Вони ефективні в командній розробці, проте вимагають налаштування інфраструктури та орієнтовані радше на довготривалий моніторинг стану кодової бази, ніж на точкові експерименти з порівнянням різних стилів реалізації задач.

Ще один клас засобів становлять статичні аналізатори, інтегровані в середовища розробки (IDE) та редактори коду. Вони використовують власні механізми аналізу для

виявлення «code smells», надмірної вкладеності коду та інших ознак підвищеної складності, доповнюючи їх локальними показниками. Такі засоби добре підтримують повсякденну роботу розробника, але зазвичай не надають прозорого доступу до формул і повного набору числових показників, необхідних для академічного дослідження впливу синтаксичних особливостей мови на підтримуваність.

Таблиця 1.1 – Порівняння функціональних можливостей інструментів оцінювання складності JavaScript-коду

Функціональна можливість	**SonarQube**	**ESLint**	**Code Climate**
Обчислення цикломатичної складності	+	+	+
Обчислення когнітивної складності для JavaScript	+	–	частково
Обчислення індексу підтримуваності (MI)	+	–	частково
Підтримка сучасного синтаксису ES6+	+	+	+
Аналіз окремих фрагментів коду без репозиторію	–	+	–
Робота без серверної інфраструктури безпосередньо в браузері	–	частково	–

У табл. 1.1 використано такі умовні позначення: «+» – функціональна можливість наявна, «частково» – підтримується опосередковано або в обмеженому вигляді, «–» – функціональна можливість відсутня.

Жоден із наведених вище інструментів повною мірою не задовольняє вимоги до дослідницького середовища для порівняння підтримуваності варіантів ES5 та ES6.

1.5 Стандарти ES5 та ES6 і їх вплив на підтримуваність JavaScript-коду

Розвиток стандарту ECMAScript від версії ES5 до ES6 супроводжувався суттєвими змінами у синтаксисі та засобах мови [7]. ES5 закріпив традиційний імперативний стиль із функціональними виразами, об’єктами-прототипами та відсутністю вбудованих механізмів модульності. У цьому стилі широко використовуються анонімні функції, ланцюжки зворотних викликів, допоміжні об’єкти для імітації структур даних.

ES6 приніс низку конструкцій, які безпосередньо впливають на підтримуваність коду:

— стрілкові функції та скорочений синтаксис для колбеків і функцій вищого порядку;

— let/const замість var для більш передбачуваної області видимості змінних;

— деструктуризація масивів та об’єктів, що зменшує кількість допоміжних операцій доступу;

— шаблонні рядки, які спрощують роботу з форматованими текстами;

— вбудовані колекції (Map, Set), що замінюють частину неявних патернів з об’єктами;

— синтаксис класів, який уніфікує опис об’єктно-орієнтованих структур.

У багатьох випадках ці конструкції дозволяють скоротити обсяг коду, усунути зайву вкладеність коду, зробити явними залежності та структуру даних. Очікується, що це позитивно впливає на показники підтримуваності та когнітивної складності. Водночас некоректне або надмірне використання сучасних можливостей мови може дати протилежний ефект, ускладнюючи розуміння коду через надлишкову лаконічність або нетривіальні вирази.

У наукових і технічних публікаціях питання підтримуваності JavaScript-коду розглядається переважно у контексті великих промислових проєктів, архітектурних стилів (наприклад, використання фреймворків) або загального технічного боргу. Спеціалізовані дослідження, присвячені саме впливу переходу від ES5 до ES6 на показники підтримуваності, зустрічаються значно рідше, а методики побудови

репрезентативних парних прикладів, що відрізняються лише стилем кодування, залишаються недостатньо формалізованими.

Проведений аналіз показав, що, попри наявність формалізованих показників складності та підтримуваності, а також розвинених інструментів статичного аналізу, залишаються невирішеними такі питання:

— відсутні загальноприйняті методики кількісного порівняння підтримуваності стилів ES5 та ES6 за однакової алгоритмічної суті коду;

— наявні інструменти не забезпечують відтворюваного браузерного середовища для аналізу окремих фрагментів коду та побудови вибірок парних прикладів;

— розрахунок показників підтримуваності часто «заховано» у великих платформах без прозорого доступу до результатів для подальшої обробки.

З огляду на це постає завдання створити інструментальне середовище, яке дозволяє:

— порівнювати реалізації ES5 та ES6 з однаковою функціональністю;

— обчислювати узгоджений набір показників підтримуваності безпосередньо у браузері;

— отримувати відтворювані статистичні результати для подальшого аналізу.

Висновки до розділу 1

У першому розділі проаналізовано сучасні підходи до оцінювання підтримуваності програмного забезпечення та роль вихідного коду в загальній якості програмних систем. Розглянуто основні показники складності та підтримуваності програмного коду, зокрема цикломатичну складність, показники Гелстеда, індекс підтримуваності та когнітивну складність, і показано доцільність їх поєднання для комплексної оцінки стану JavaScript-модулів.

Проаналізовано принципи статичного аналізу на основі абстрактного синтаксичного дерева та особливості застосування цього підходу до JavaScript, що підтримує декілька версій стандарту ECMAScript і розширення синтаксису. Показано, що наявні інструменти оцінювання складності JavaScript-коду або орієнтовані на важкі серверні рішення, або не забезпечують прозорого та відтворюваного експериментального середовища для порівняння стилів ES5 та ES6.

Розглянуто розвиток стандартів ES5 та ES6 і окреслено групи синтаксичних конструкцій, які потенційно впливають на підтримуваність коду: механізми оголошення змінних, стрілкові функції, деструктуризація, шаблонні рядки, вбудовані колекції, синтаксис класів. Показано, що попри широке використання ES6 у практиці веб-розробки, питання кількісного порівняння підтримуваності коду, написаного у стилях ES5 та ES6 з однаковою функціональністю, залишаються недостатньо вивченими.

Отримані висновки обґрунтовують доцільність розроблення веб-застосунку для відтворюваного обчислення показників підтримуваності JavaScript-коду безпосередньо в браузері та проведення експериментального дослідження впливу використання стандарту ES6 на підтримуваність програмних модулів. Саме ці завдання деталізуються та реалізуються у наступних розділах роботи. Сформовані в розділі висновки безпосередньо обґрунтовують мету роботи, наведеної у вступі, та визначають основні завдання дослідження: розроблення інструменту для обчислення показників

підтримуваності JavaScript-коду в браузері та проведення експериментального порівняння стилів ES5 і ES6.

РОЗДІЛ 2 ОБҐРУНТУВАННЯ МЕТОДИКИ ДОСЛІДЖЕННЯ ВПЛИВУ СТАНДАРТУ ES6 НА ПІДТРИМУВАНІСТЬ JAVASCRIPT-КОДУ

2.1 Формулювання задачі та дослідницької гіпотези

У першому розділі було показано, що підтримуваність програмного забезпечення суттєво залежить від структурної та когнітивної складності вихідного коду, а стандарт ECMAScript 2015 (ES6) пропонує низку конструкцій, які потенційно спрощують читання та супровід JavaScript-програм. Водночас у науковій та прикладній літературі відсутні загальноприйняті методики кількісного порівняння підтримуваності реалізацій у стилях ES5 та ES6 за умови збереження однакової алгоритмічної суті.

У цьому контексті дослідницьку задачу можна сформулювати так:
порівняти показники підтримуваності JavaScript-коду, реалізованого в стилях ES5 та ES6, на парних прикладах з однаковою функціональністю, використовуючи формалізовані показники складності та непараметричні статистичні критерії для оцінювання значущості спостережуваних відмінностей.

Основною гіпотезою дослідження є твердження, що використання конструкцій ES6 (стрілкові функції, let/const, деструктуризація, шаблонні рядки, колекції Map/Set, синтаксис класів) за умови збереження алгоритмічної семантики коду призводить до покращення показників підтримуваності у порівнянні з еквівалентними реалізаціями у стилі ES5.

Очікується, що це покращення проявиться у:

— зростанні індексу підтримуваності (Maintainability Index, MI);

— зменшенні когнітивної складності (Cognitive Complexity, CC);

— зменшенні цикломатичної складності (G) та кількості логічних рядків коду (LOC) як підтримувальних показників.

2.2 Вибір показників підтримуваності та критерії порівняння

Для кількісного порівняння стилів ES5 та ES6 обрано узгоджений набір показників, розглянутих у першому розділі:

Таблиця 2.1 – Порівняння показників підтримуваності коду

Показник	Переваги	Недоліки	Обґрунтування вибору
Maintainability Index (MI)	Інтегральний, нормалізований діапазон 0–100, агрегує структурну складність і обсяг	Залежить від конкретної редакції формули	Єдиний показник, що поєднує структурну складність, обсяг коду та словниковий запас
Cognitive Complexity (CC)	Враховує ментальне навантаження, вкладеність та нетривіальні конструкції	Емпірична модель, потребує валідації	Краще відображає читабельність коду порівняно з G
Cyclomatic Complexity (G)	Формалізований, широко використовується, легко обчислюється	Не враховує читабельність та стилістичні особливості	Класичний показник складності, необхідний для розрахунку MI
Halstead Volume (V)	Описує словниковий обсяг програми	Чутливий до стилю найменування	Підтримувальний показник для MI та інтерпретації структури
Lines of Code (LOC)	Простий, інтуїтивно зрозумілий	Не враховує складність логіки	Характеризує компактність реалізації

У рамках дослідження для кожної пари реалізацій (ES5, ES6) обчислюються вказані показники. Для порівняння стилів приймається така система критеріїв:

— MI: більше значення вважається кращим;

— CC, G, LOC: менше значення вважається кращим;

— V: розглядається як контекстний показник, що допомагає інтерпретувати зміни MI та CC (зменшення V за незмінної функціональності зазвичай свідчить про більш лаконічний і зрозумілий код).

Таким чином, кожна пара ES5/ES6 описується вектором показників, і завдання зводиться до аналізу розподілу різниць між ними.

2.3 Побудова експериментальної вибірки парних прикладів ES5/ES6

Для експериментального дослідження сформовано вибірку парних прикладів коду, яка відповідає таким вимогам:

— еквівалентність функціональності: кожна пара складається з двох файлів (ES5 та ES6), що реалізують однакову задачу;

— типовість завдань: приклади моделюють поширені фрагменти прикладного JavaScript-коду, характерні для веб-застосунків (обробка масивів, фільтрація/трансформація даних, форматування рядків, валідація, прості алгоритми тощо);

— контроль структури: в ES6-варіантах збережено загальну логіку, але використано сучасні конструкції замість класичних патернів ES5 (анонімні функції замінені на стрілкові, var замінено на let/const, введено деструктуризацію та шаблонні рядки, де це доцільно);

— обмеження домену: не розглядаються фреймворк-специфічні конструкції та складна асинхронність (callback hell, проміси, async/await), щоб зосередитися на «чистій» різниці між стилями ES5 та ES6.

Експериментальна вибірка складається з 100 парних прикладів, що дозволяє виконати детальний аналіз структури коду та інтерпретацію показників.

2.4 Методика обчислення показників підтримуваності

Для обчислення показників підтримуваності JavaScript-коду використовується метод статичного аналізу вихідного коду без його виконання. Статичний аналіз дозволяє отримати об'єктивні кількісні оцінки структури та складності програми на основі її текстового представлення.

Обчислення показників виконується у такій послідовності:

1. Парсинг коду та побудова внутрішнього представлення структури програми. На цьому етапі вихідний текст JavaScript-коду перетворюється на структуроване представлення, що відображає синтаксичні конструкції мови: оголошення змінних, функції, умовні оператори, цикли, вирази тощо. Це дозволяє автоматизовано аналізувати структуру коду незалежно від стилю форматування.

2. Підрахунок операторів та операндів для показників Гелстеда. Операторами вважаються ключові слова мови (if, for, while, function тощо), знаки операцій (+, -, *, /, = тощо) та виклики функцій. Операндами є ідентифікатори змінних, літерали (числа, рядки, булеві значення) та аргументи функцій. Підрахунок виконується з урахуванням як загальної кількості входжень (N_1 для операторів, N_2 для операндів), так і кількості унікальних елементів (n_1 та n_2 відповідно). На основі цих даних обчислюється показник обсягу програми V за формулою (1.1).

3. Виявлення точок прийняття рішень для цикломатичної складності. Цикломатична складність G визначається як кількість незалежних шляхів виконання програми. Для її обчислення виявляються всі конструкції, що створюють розгалуження потоку виконання: умовні оператори (if, else if), оператори вибору (switch-case), тернарні оператори (?:), логічні оператори короткого замикання (&&, ||) та цикли (for, while, do-while). Цикломатична складність обчислюється як кількість виявлених точок рішень плюс одиниця.

4. Оцінка вкладеності конструкцій для когнітивної складності. Когнітивна складність CC розраховується за специфікацією SonarSource v1.7, яка враховує не лише

кількість розгалужень, але й глибину їх вкладеності та порушення лінійності виконання. Кожна конструкція керування потоком (if, else, switch, цикли) отримує базовий штраф, який збільшується пропорційно до рівня вкладеності. Додаткові штрафи застосовуються для конструкцій break, continue та складних логічних виразів. Такий підхід дозволяє оцінити ментальне навантаження на розробника при читанні коду.

5. Підрахунок логічних рядків коду (LOC). При визначенні кількості логічних рядків коду враховуються лише рядки з виконуваними інструкціями. Порожні рядки, коментарі (однорядкові // та багаторядкові /* */) та рядки, що містять лише синтаксичні елементи (фігурні дужки, крапки з комою), виключаються з підрахунку. Багаторядкові конструкції, такі як об'єктні літерали або масиви, підраховуються коректно з урахуванням фактичної кількості логічних операцій.

6. Після отримання базових показників V, G та LOC обчислюється індекс підтримуваності MI згідно з формулою (1.2). Нормалізація до діапазону [0; 100] забезпечує зручність інтерпретації результатів: значення MI понад 20 традиційно вважаються показником доброї підтримуваності, значення 10-20 — середньої, а значення нижче 10 — поганої підтримуваності коду.

Для забезпечення відтворюваності результатів та виключення систематичної похибки обидва стилі коду (ES5 та ES6) аналізуються єдиним методом з використанням однакових алгоритмів підрахунку показників. Це дозволяє гарантувати, що виявлені відмінності в показниках підтримуваності зумовлені саме різницею в синтаксичних конструкціях, а не особливостями інструментів аналізу.

Метод статичного аналізу має низку переваг порівняно з динамічними підходами, що базуються на запуску програм: він не вимагає виконання коду, не залежить від тестових даних, дозволяє аналізувати неповні фрагменти програм та забезпечує детерміновані результати незалежно від середовища виконання. Водночас статичний аналіз має обмеження щодо виявлення деяких типів складності, пов'язаних з

динамічною поведінкою програми під час виконання, проте для порівняння синтаксичних стилів ES5 та ES6 ці обмеження не є критичними.

2.5 Статистична обробка результатів та застосування критерію Вілкоксона

Оскільки для кожної задачі в роботі побудовано парні реалізації у стилях ES5 та ES6, природно розглядати результати як дві пов’язані вибірки. Для кожного показника X (Maintainability Index, когнітивна складність, цикломатична складність, LOC) для i-ої пари обчислюється різниця:

$$d_i = X_i^{(ES6)} - X_i^{(ES5)}, \quad (2.1)$$

де X – обраний показник підтримуванності

Для індексу підтримуваності позитивні значення d_i інтерпретуються як перевага реалізації ES6 над ES5. Для когнітивної та цикломатичної складності, а також LOC, навпаки, цікавими є від’ємні значення d_i, оскільки вони означають зменшення складності або обсягу коду в ES6-варіанті за незмінної функціональності.

Розподіл різниць d_i для реального програмного коду, як правило, є далеким від нормального:

— значення показників дискретні;

— у частини пар різниця може дорівнювати нулю;

— можливі поодинокі великі відхилення.

Тому застосування класичних параметричних тестів (наприклад, t-тесту для парних вибірок) було б методично сумнівним. У роботі використовується непараметричний критерій Вілкоксона для парних вибірок [8], який:

— не вимагає припущення про нормальний розподіл різниць;

— враховує як знак, так і відносну величину відхилень d_i;

— придатний для вибірок порядку 100 спостережень, характерних для даного дослідження.

Процедура застосування критерію Вілкоксона для кожного показника включає такі кроки:

1. Виключення пар, для яких $d_i = 0$ (відсутність відмінностей між ES5 та ES6 для даного показника).

2. Упорядкування ненульових $|d_i|$ у зростаючому порядку та присвоєння їм рангів із урахуванням можливих зв'язок.

3. Розділення рангів за знаком різниці та обчислення сум рангів для додатних і від'ємних відхилень (T^+ та T^-).

4. Вибір статистики критерію як: $T = min(T^+, T^-)$, та подальше обчислення p-value за табличними або наближеними нормальними оцінками.

5. Порівняння отриманого p-value з рівнями значущості $\alpha = 0,05$ та $\alpha = 0,01$. Якщо $p < \alpha$, нульова гіпотеза про відсутність систематичної різниці між ES5 та ES6 для відповідного показника відхиляється.

Для оцінювання практичної значущості відмінностей додатково використовується коефіцієнт величини ефекту:

$$r = \frac{|Z|}{\sqrt{n}}, \tag{2.2}$$

де Z – нормалізоване значення статистики критерію Вілкоксона;

n – кількість ненульових пар.

Значення r інтерпретуються за загальноприйнятою шкалою:

— $r \approx 0,1$ – малий ефект;

— $r \approx 0,3$ – середній ефект;

— $r \geq 0,5$ – великий ефект.

Таким чином, для кожного показника підтримуваності процедура дає не лише відповідь на питання, чи відрізняються реалізації ES5 та ES6 статистично значущим чином, а й дозволяє кількісно оцінити силу цього ефекту на вибірці зі 100 парних прикладів.

2.6 Обмеження методики та оцінка похибок

Запропонована методика дослідження має низку обмежень, які необхідно враховувати під час інтерпретації результатів та формулювання висновків.

Обмеженість вибірки задач.

Вибірка складається зі 100 парних прикладів коду, спеціально сконструйованих для моделювання типових задач веб-розробки (обробка масивів, фільтрація та трансформація даних, форматування рядків, валідація, прості алгоритми). Такий підхід дозволяє контролювати алгоритмічну сутність завдання та фіксувати різницю саме в стилі кодування (ES5 проти ES6), проте не охоплює великі монолітні системи й багатомодульні проєкти, фреймворк-специфічні патерни (React, Vue, Angular тощо), складну асинхронну взаємодію, конкурентність та інтеграцію з зовнішніми сервісами. Тому результати безпосередньо характеризують насамперед типові фрагменти прикладного коду, а не повну картину промислових кодових баз.

Синтетичний характер прикладів та суб'єктивність рефакторингу.

Хоча кожна пара ES5/ES6 реалізує однакову функціональність, перетворення коду в стилі ES6 неминуче містить елемент суб'єктивності: вибір конкретних конструкцій ES6 (стрілкові функції, деструктуризація, колекції) залежить від розробника, одна й та сама задача може мати кілька однаково коректних, але структурно різних варіантів запису. У дослідженні приймається припущення, що створені приклади є репрезентативними для «якісного» використання ES6, але це не виключає впливу індивідуального стилю програмування.

Залежність результатів від обраних показників.

Висновки ґрунтуються на конкретному наборі показників: Maintainability Index, когнітивна та цикломатична складність, Halstead Volume та LOC. Формула MI відповідає реалізації, прийнятій у середовищі Microsoft Visual Studio, і включає лише частину можливих характеристик коду. Когнітивна складність розраховується за версією правил SonarSource v1.7 і є емпіричною моделлю ментального навантаження, а не фізично вимірюваною величиною. За інших варіантів формул або додаткових

показників (наприклад, з урахуванням глибини модульної структури чи взаємозв’язків між файлами) числові результати могли б дещо відрізнятися.

Залежність від інструментального засобу статичного аналізу.

Обчислення показників здійснюється за допомогою спеціально розробленого інструментального засобу статичного аналізу, який працює на основі абстрактного синтаксичного дерева JavaScript-коду. Попри те, що алгоритми базуються на опублікованих визначеннях показників, на практиці можливі відмінності у трактуванні окремих конструкцій мови (наприклад, умовних виразів, логічних операторів, скорочених форм запису); обробка специфічних синтаксичних форм (optional chaining, логічне об’єднання умов) може частково відрізнятися від реалізацій у промислових інструментах. Для мінімізації інструментальної похибки в роботі для обох стилів кодування використовується одна й та сама реалізація аналізу, що дозволяє вважати порівняння внутрішньо узгодженим.

Обмеженість статистичних висновків.

Використання критерію Вілкоксона для 100 парних спостережень дає змогу надійно оцінити, чи має місце систематична перевага ES6 над ES5 за розглянутими показниками. Однак навіть за дуже малих значень p-value результати формально стосуються саме тієї вибірки прикладів, що була проаналізована, і не гарантують, що в кожному конкретному реальному проєкті перехід на ES6 автоматично покращить підтримуваність коду.

З урахуванням наведених обмежень запропонована методика може розглядатися як інструмент для експериментального порівняння стилів ES5 та ES6 на рівні окремих фрагментів коду й виявлення загальної тенденції впливу сучасних конструкцій ES6 на показники підтримуваності. Водночас для повної екстраполяції результатів на великі промислові системи потрібні додаткові дослідження на розширених вибірках та, за можливості, з залученням інших незалежних інструментів аналізу.

Висновки до розділу 2

У другому розділі обґрунтовано вибір напряму дослідження та сформульовано дослідницьку гіпотезу щодо впливу використання конструкцій стандарту ES6 на підтримуваність JavaScript-коду. Показано, що порівняння стилів ES5 та ES6 доцільно виконувати на парних прикладах з однаковою алгоритмічною сутністю, оцінюючи відмінності за формалізованими показниками підтримуваності.

Обґрунтовано вибір узгодженого набору показників: індексу підтримуваності (Maintainability Index), когнітивної та цикломатичної складності, обсягу програми за Гелстедом і кількості логічних рядків коду. Визначено критерії «кращості» для кожного показника та показано, що аналіз має виконуватися на рівні векторів показників для парних реалізацій ES5/ES6.

Описано методику формування експериментальної вибірки зі 100 парних прикладів коду, яка моделює типові задачі веб-розробки та дозволяє контролювати різницю саме в стилі кодування. Для статистичної перевірки гіпотези про перевагу ES6 над ES5 обґрунтовано використання непараметричного критерію Вілкоксона для парних вибірок та коефіцієнта величини ефекту r, що дає змогу оцінити як статистичну, так і практичну значущість виявлених відмінностей.

Проаналізовано основні обмеження запропонованої методики, пов’язані з характером вибірки, суб’єктивністю рефакторингу, вибором показників і використанням конкретного інструменту статичного аналізу. Показано, що за цих обмежень методика є адекватним інструментом для експериментального порівняння стилів ES5 та ES6 на рівні окремих фрагментів коду та виявлення загальної тенденції впливу сучасних конструкцій ES6 на показники підтримуваності. Отримані результати другого розділу створюють підґрунтя для розроблення інструментального забезпечення дослідження, що розглядається у третьому розділі.

РОЗДІЛ 3 ПРОЄКТУВАННЯ Й РОЗРОБКА ІНСТРУМЕНТАЛЬНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ДОСЛІДЖЕННЯ ВПЛИВУ СТАНДАРТУ ES6 НА ПІДТРИМУВАНІСТЬ JAVASCRIPT-КОДУ

3.1 Формалізація задачі

3.1.1 Опис дослідницької задачі

У контексті стрімкого розвитку веб-технологій та еволюції стандарту ECMAScript питання оцінки якості програмного коду набуває критичного значення. Як зазначається у навчальних посібниках з інженерії програмного забезпечення, основною метою розробки є не лише створення функціонального продукту, але й забезпечення його рентабельності та можливості супроводу протягом тривалого життєвого циклу. Стандарт ECMAScript 2015 (ES6) вніс фундаментальні зміни у синтаксис мови JavaScript, запровадивши нові конструкції (стрілкові функції, класи, деструктуризацію, шаблонні рядки тощо), які потенційно можуть як спростити, так і ускладнити сприйняття коду розробниками.

Існуючі інструменти статичного аналізу, такі як SonarQube, ESLint або Code Climate, орієнтовані переважно на безперервну інтеграцію (CI/CD) та довготривалий моніторинг великих кодових баз. Вони часто вимагають складної серверної інфраструктури, попереднього налаштування оточення та не надають зручного інтерфейсу для порівняльного експериментального аналізу ізольованих фрагментів коду «тут і зараз». Для проведення наукового дослідження, яке передбачає кількісне порівняння парних реалізацій алгоритмів (у стилях ES5 та ES6) за фіксованим набором показників, необхідне спеціалізоване інструментальне забезпечення.

Основною задачею даного етапу роботи є проєктування та програмна реалізація веб-застосунку «Code Complexity Lab». Ця система має вирішувати проблему автоматизованого обчислення показників підтримуваності безпосередньо у браузері користувача, забезпечуючи при цьому конфіденційність коду, високу швидкість обробки та візуалізацію результатів у зрозумілому форматі.

Система повинна забезпечувати розрахунок наступних ключових показників:

— Maintainability Index (MI) — інтегральний показник, що агрегує декілька показників в єдине число (0–100), дозволяючи швидко оцінити загальний стан модуля.

— Cognitive Complexity (CC) — сучасний показник від SonarSource, який, на відміну від цикломатичної складності, оцінює не лише структурну розгалуженість, а й ментальне навантаження на програміста при читанні коду.

— Cyclomatic Complexity (G) — класичний показник по МакКейбу, що визначає кількість лінійно незалежних маршрутів виконання програми.

— Halstead Volume (V) — показник, що базується на інформаційній теорії та оцінює обсяг і лексичну насиченість коду.

— Lines of Code (LOC) — кількість логічних рядків коду, очищених від коментарів та форматування, що є базовим показником розміру.

3.1.2 Діаграма варіантів використання (Use Case)

Для чіткого визначення функціональних меж системи та ролей користувачів було розроблено діаграму варіантів використання (Use Case Diagram), яка представлена на рисунку 3.1.

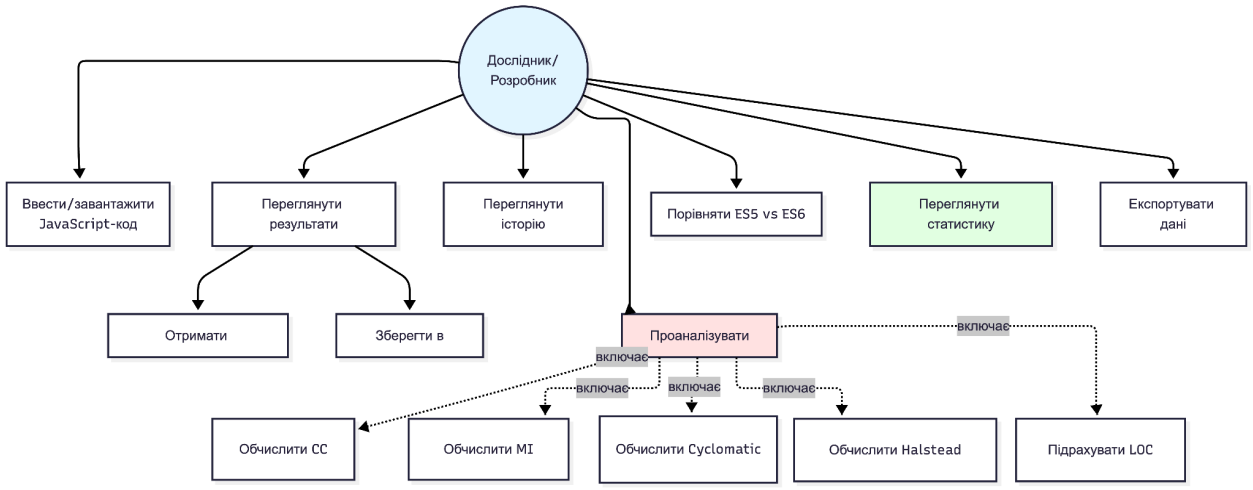


Рисунок 3.1 – Діаграма варіантів використання (Use Case)

назва - по центру

Опис діаграми та взаємодії акторів:

На діаграмі 3.1 представлено основного актора — Дослідника (Researcher). Це користувач, який проводить експериментальний аналіз коду. Система не передбачає реєстрації або розмежування прав доступу, оскільки працює локально в браузері.

Ключові варіанти використання (Use Cases) включають:

— Введення/Завантаження коду: Користувач може ввести код вручну у текстовий редактор або завантажити файл з розширенням .js. Система повинна валідувати вхідні дані.

— Запуск аналізу (Analyze Code): Ініціалізація процесу парсингу та розрахунку метрик. Цей процес включає внутрішні кроки: побудову AST, обхід дерева, агрегацію результатів.

— Перегляд результатів (View Metrics): Відображення обчислених значень MI, CC, G, V, LOC з візуальною індикацією (кольорові маркери, прогрес-бари).

— Керування історією (Manage History): Перегляд, відновлення або очищення результатів попередніх аналізів, що зберігаються локально.

— Експорт даних (Export CSV): Можливість вивантаження накопиченої статистики для подальшої обробки у зовнішніх статистичних пакетах (Excel).

3.1.3 Функціональні вимоги до системи

Базуючись на формалізації задачі та аналізі предметної області, було сформовано перелік функціональних вимог, які є обов'язковими для реалізації.

Вимоги до підсистеми аналізу:

— Система повинна виконувати повний синтаксичний аналіз JavaScript-коду, включаючи найновіші стандарти (ES2020+), такі як Optional Chaining (?.) та Nullish Coalescing (??).

— Обчислення метрик повинно відбуватися виключно на стороні клієнта (в браузері) для забезпечення приватності даних.

— Система повинна коректно обробляти синтаксичні помилки, надаючи користувачу зрозумілі повідомлення, та застосовувати fallback-механізми (наприклад, регулярні вирази) для наближеної оцінки, якщо побудова AST неможлива.

Вимоги до інтерфейсу користувача:

— Система повинна забезпечувати підсвітку синтаксису (syntax highlighting) для введеного коду.

— Візуалізація метрик повинна включати кольорову градацію відповідно до загальноприйнятих стандартів якості (наприклад, MI < 65 — червоний, MI > 85 — зелений).

— Інтерфейс має бути адаптивним для роботи на пристроях з різною роздільною здатністю екрана.

Вимоги до зберігання та експорту:

— Система повинна автоматично зберігати останні 10 сесій аналізу у localStorage браузера.

— Система повинна надавати можливість пакетного завантаження прикладів для формування сторінки статистики.

3.2 Базова архітектура системи

3.2.1 Вибір архітектурного підходу (SPA)

Для реалізації системи обрано архітектурний стиль Single Page Application (SPA) з клієнтським рендерингом (Client-Side Rendering). Такий підхід передбачає, що при першому зверненні до сервера браузер завантажує єдиний HTML-файл та необхідні скрипти й стилі, після чого вся взаємодія з користувачем відбувається без перезавантаження сторінки.

Обґрунтування вибору:

1. Відтворюваність експерименту: Оскільки вся логіка аналізу запакована у статичні JavaScript-файли, результати обчислень будуть ідентичними на будь-якому комп'ютері, незалежно від наявності інтернет-з'єднання чи навантаження на сервер.

2. Конфіденційність: Досліджуваний код може бути комерційною таємницею. Клієнтська архітектура гарантує, що вихідний код ніколи не покидає межі оперативної пам'яті браузера користувача.
3. Незалежність від серверної інфраструктури: Систему можна розмістити на будь-якому статичному хостингу (GitHub Pages, Netlify) або запускати локально без необхідності налаштування складних бекенд-сервісів чи баз даних.

3.2.2 Компонентна модель системи

Архітектура системи побудована за модульним принципом, що забезпечує низьку зв'язність (low coupling) та високу згуртованість (high cohesion) компонентів.

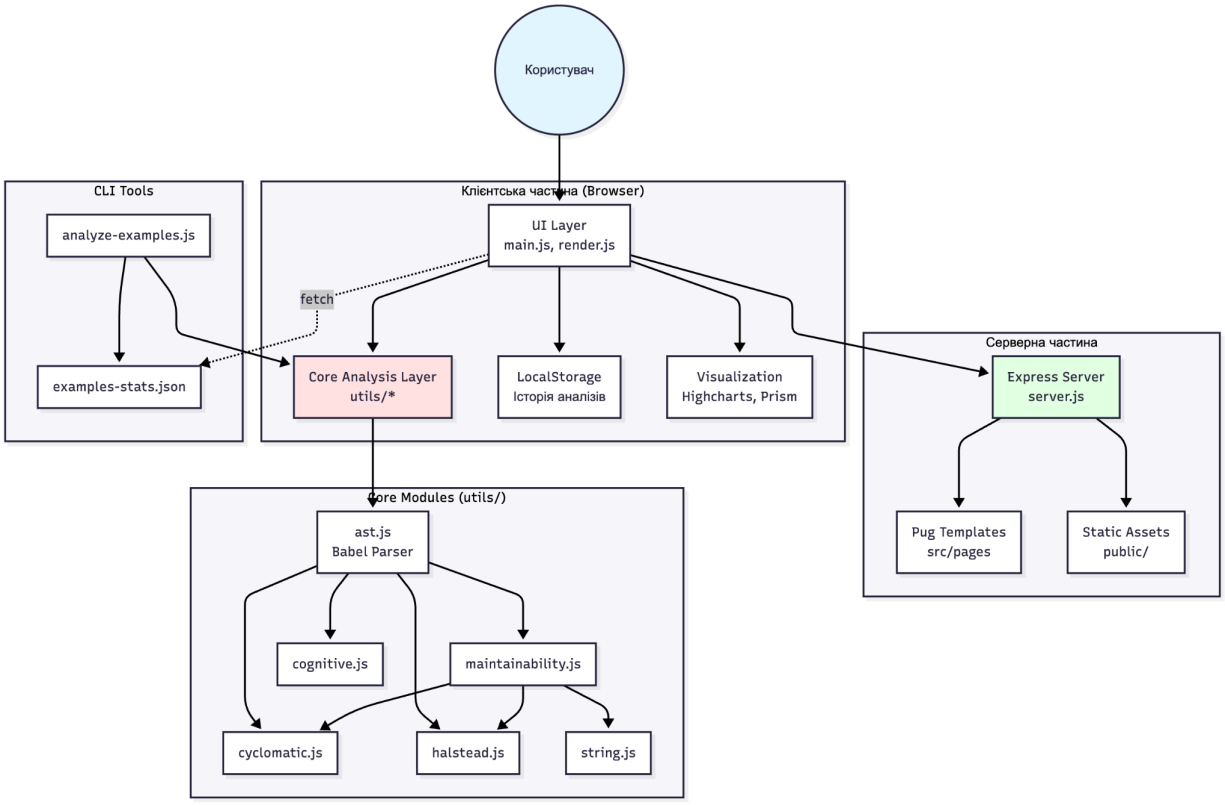


Рисунок 3.2 – Компонентна архітектура системи

Опис діаграми 3.2:

На діаграмі зображено три основні рівні системи:

1. Рівень інтерфейсу (UI Layer): Включає HTML-розмітку (згенеровану з Pug), стилі Bootstrap та скрипти керування DOM (main.js, render.js). Цей рівень відповідає за отримання вводу від користувача та відображення результатів.

2. Рівень бізнес-логіки (Core Analysis Layer): Набір "чистих" модулів у папці utils/ (ast.js, cognitive.js, maintainability.js тощо). Цей рівень не залежить від UI і виконує безпосередні математичні та логічні операції над кодом.

3. Рівень даних (Data Layer): Взаємодія з localStorage для збереження історії та отримання статичних JSON-файлів з прикладами коду через fetch API.

Взаємодія між UI та Core відбувається через чітко визначені інтерфейси функцій. Наприклад, main.js передає рядок коду у maintainabilityIndex(), отримує числовий результат і передає його у render.js для оновлення DOM.

3.2.3 Розміщення компонентів

Фізичне розміщення компонентів системи орієнтоване на мінімізацію затримок та спрощення розгортання.

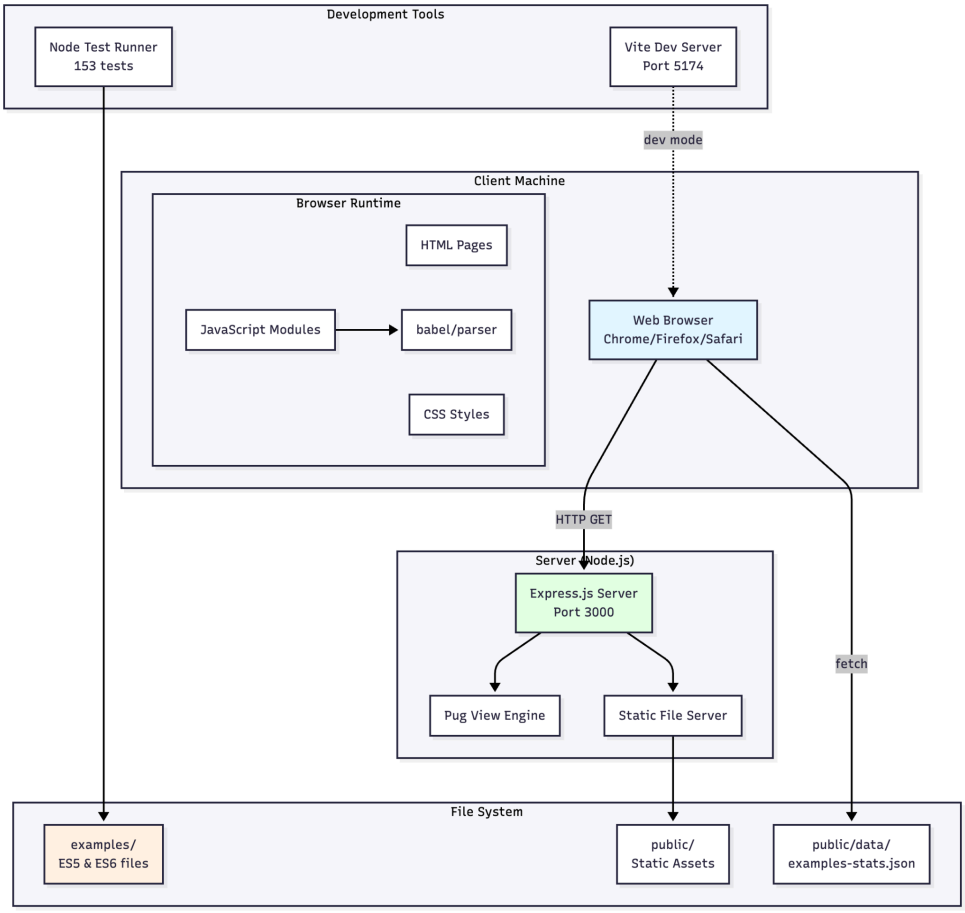


Рисунок 3.3 – Діаграма розміщення компонентів

Опис діаграми 3.3:

Діаграма демонструє, що на стороні сервера (Node.js + Express) знаходяться лише статичні ресурси (HTML, CSS, JS bundles). Весь обчислювальний процес відбувається на клієнтському пристрої (Workstation) у середовищі веб-браузера (Chrome/Firefox/Edge). CLI-інструменти для пакетного аналізу виконуються локально в середовищі Node.js, генеруючи JSON-звіти, які потім споживаються веб-застосунком.

3.3 Внутрішнє проєктування

3.3.1 Вибір мови програмування та обґрунтування

В якості основної мови програмування обрано JavaScript (стандарт ES6+).

Обґрунтування вибору:

1. Нативна підтримка: JavaScript є "рідною" мовою браузерів, що дозволяє уникнути транспіляції з інших мов (окрім використання Babel для сумісності) та спрощує налагодження.

2. Інтроспекція: Для аналізу JavaScript-коду найкраще підходять інструменти, написані на самому JavaScript (наприклад, Babel Parser), оскільки вони використовують ті ж самі структури даних (об'єкти, масиви) для представлення AST.

3. Єдина екосистема: Використання Node.js для інструментів збірки (Vite) та тестування дозволяє використовувати одну мову на всіх етапах розробки.

3.3.2 Технологічна платформа

Технологічний стек підібрано з урахуванням сучасних тенденцій у веб-розробці та вимог до продуктивності.

Таблиця 3.1 – Технологічна платформа системи

Компонент	Технологія	Версія	Призначення
Build tool	Vite	5.4.0	Забезпечує надшвидку збірку проекту, гарячу заміну модулів (HMR) під час розробки та оптимізацію ресурсів для продакшну.

Templates	Pug	CLI 1.0.0-alpha6	Шаблонізатор, що дозволяє писати лаконічний HTML-код, використовувати змінні та цикли під час генерації статики.
UI Framework	Bootstrap	5.3.3	Надає готову сітку (grid system) та компоненти (картки, прогрес-бари), що пришвидшує розробку адаптивного інтерфейсу.
Syntax Highlighting	Prism	1.29.0	Легковагова бібліотека для підсвічування синтаксису коду в блоках прев'ю.
Charts	Highcharts	-	Потужна бібліотека для побудови інтерактивних графіків на сторінці статистики.
Server	Express	4.21.2	Мінімалістичний веб-сервер для роздачі статичних файлів у продакшн-середовищі.
AST Parser	@babel/parser	7.25.0	Індустріальний стандарт для парсингу JS. Використовується з плагінами jsx, typescript та ін.
Testing	Node Test Runner	Built-in	Вбудований інструмент для запуску тестів, що не потребує додаткових важких залежностей.

Конфігурація аналізатора коду (Babel):

Для забезпечення максимальної сумісності з сучасним кодом, парсер налаштовано наступним чином:

```
// ast.js — базовий парсинг та обхід AST (@babel/parser)
import * as babelParser from '@babel/parser';

export const SOURCE_TYPE = 'unambiguous';
export const ALLOW_RETURN_OUTSIDE_FN = true;
export const EMIT_TOKENS = true;

export const BABEL_PLUGINS = [
 'jsx',
```

```
  'typescript',
  'classProperties',
  'classPrivateProperties',
  'classPrivateMethods',
  'decorators-legacy',
  'dynamicImport',
  'exportDefaultFrom',
  'exportNamespaceFrom',
  'objectRestSpread',
  'optionalCatchBinding',
  'optionalChaining',
  'nullishCoalescingOperator',
  'topLevelAwait',
  'doExpressions',
  'logicalAssignment',
];

export const PARSER_OPTIONS = {
  sourceType: SOURCE_TYPE,
  allowReturnOutsideFunction: ALLOW_RETURN_OUTSIDE_FN,
  plugins: BABEL_PLUGINS,
  tokens: EMIT_TOKENS,
};

const toStringSafe = (v) => String(v ?? '');
const isAstNode = (v) => !!v && typeof v.type === 'string';
const ownKeys = (o) => Object.keys(o);
const hasOwn = (o, k) => Object.prototype.hasOwnProperty.call(o, k);

const pushIfNode = (stack, v) => {
  if (isAstNode(v)) stack.push(v);
};

const pushArrayNodesReversed = (stack, arr) => {
  for (let i = arr.length - 1; i >= 0; i--) {
    pushIfNode(stack, arr[i]);
  }
};

export function parse(src) {
  return babelParser.parse(toStringSafe(src), PARSER_OPTIONS);
}

export function walk(node, enter) {
```

```
  if (!isAstNode(node) || typeof enter !== 'function') return;

  const stack = [node];

  while (stack.length) {
    const n = stack.pop();
    if (!isAstNode(n)) continue;

    enter(n);

    for (const key of ownKeys(n)) {
      if (!hasOwn(n, key)) continue;
      const val = n[key];
      if (!val) continue;

      if (Array.isArray(val)) {
        pushArrayNodesReversed(stack, val);
      } else {
        pushIfNode(stack, val);
      }
    }
  }
}
```

3.3.3 Ієрархія та взаємодія модулів системи

Система має чітку модульну структуру, де кожен файл відповідає за конкретну підзадачу.

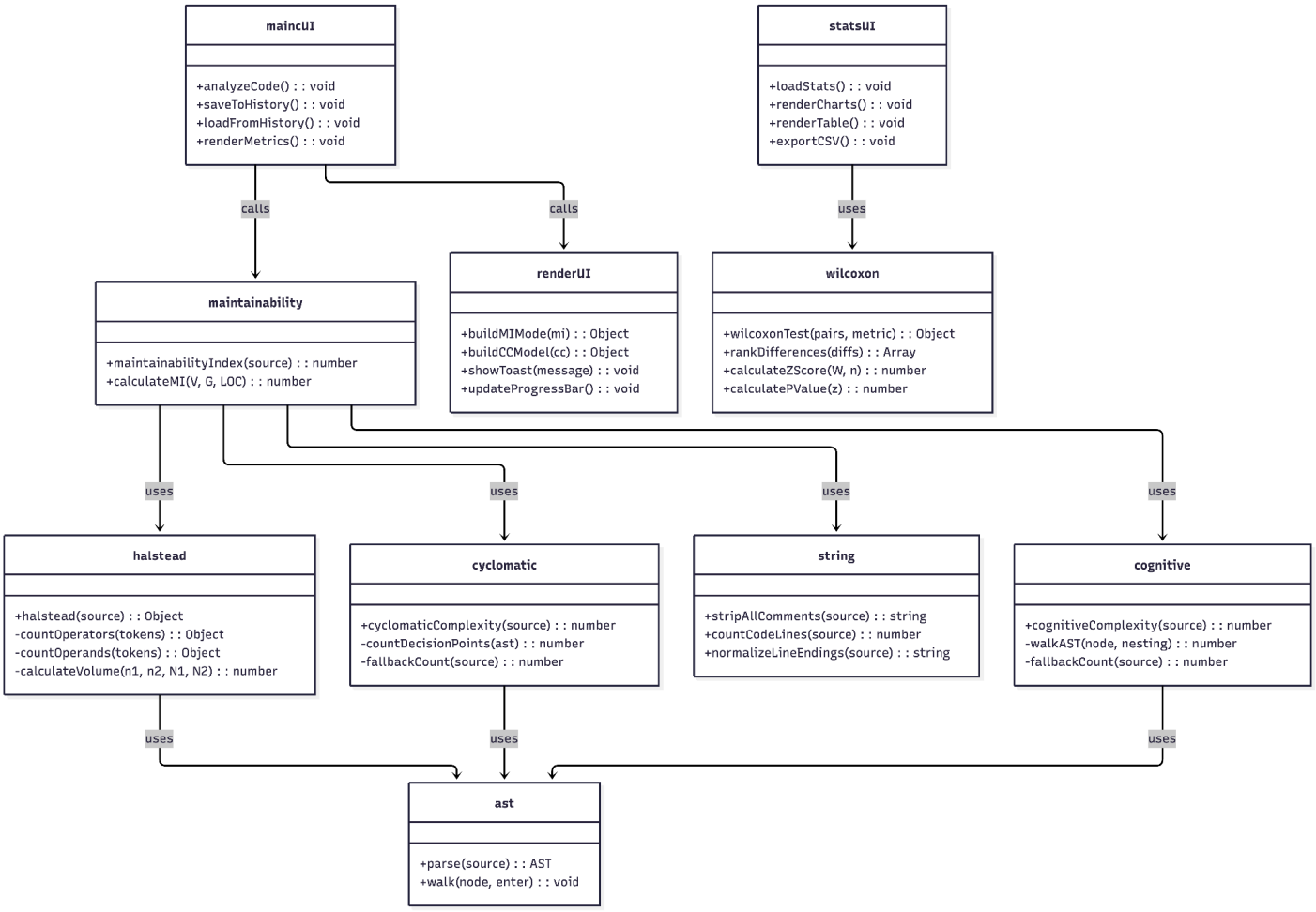


Рисунок 3.4 – Діаграма класів/модулів (структура utils/)

Опис модулів (public/js/utils/):

1. ast.js: Відповідає за взаємодію з @babel/parser. Реалізує функцію walk(node, enter), яка дозволяє виконувати обхід абстрактного синтаксичного дерева в глибину (Depth-First Search), надаючи колбек-функцію enter для кожного відвіданого вузла.

2. maintainability.js: Головний модуль розрахунку якості. Він агрегує дані з інших модулів (обсяг, складність, LOC) та обчислює індекс за формулою Індексу Підримуваності.

3. cognitive.js: Реалізує алгоритм розрахунку когнітивної складності. Модуль відстежує рівень вкладеності (nesting) та нараховує штрафні бали за структурні елементи, що порушують лінійний потік читання.

4. cyclomatic.js: Обчислює цикломатичну складність шляхом підрахунку предикатних вузлів (вузлів розгалуження) у графі потоку керування.

5. halstead.js: Аналізує лексичний склад програми (токени), розділяючи їх на оператори та операнди, і обчислює метрики Гелстеда (обсяг, довжину, словник).

6. string.js: Допоміжний модуль для роботи з рядками. Містить регулярні вирази для видалення коментарів (stripAllComments) та підрахунку значущих рядків коду.

3.3.4 Використані принципи проєктування

При розробці системи було застосовано наступні інженерні принципи:

— Розділення відповідальностей (Separation of Concerns): Модулі в папці utils/ є "чистими функціями" (pure functions) — вони приймають вихідний код і повертають числові значення, не маючи жодних побічних ефектів (side effects) на кшталт зміни DOM або звернення до мережі. Логіка відображення повністю ізольована в render.js.

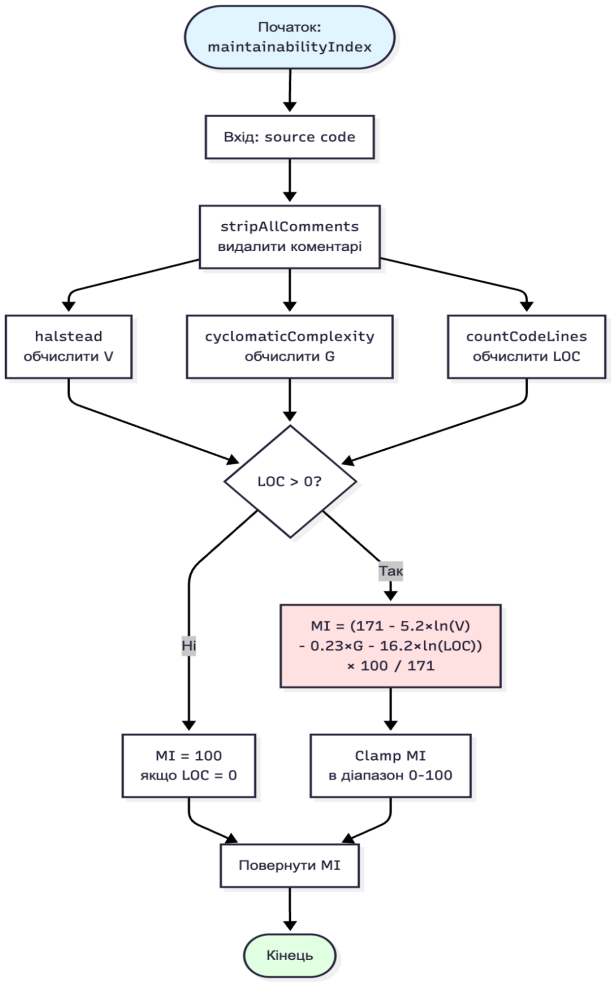
— Модульність: Кожен алгоритм розрахунку показника винесено в окремий файл, що спрощує тестування та модифікацію.

— Fallback-механізми: Враховуючи динамічну природу JavaScript, передбачено механізм відмовостійкості. Якщо Babel не може розпарсити код (наприклад, через синтаксичну помилку), система перехоплює виключення і запускає спрощений аналіз на основі регулярних виразів (Regex counting), про що повідомляє користувача.

— Конфігурованість: Порогові значення для метрик (наприклад, межі "зеленої", "жовтої" та "червоної" зон) винесені в конфігураційні константи, що дозволяє легко адаптувати інструмент під різні стандарти кодування.

3.3.5 Алгоритми обчислення показників

Блок-схему алгоритму обчислення показника Maintainability Index (MI) наведено на рисунку 3.5.



по центру: і малюнок і назву

Рисунок 3.5 – Блок-схема алгоритму обчислення Maintainability Index

Опис алгоритму (рис. 3.5):

1. Отримати вихідний код.
2. Очистити код від коментарів (функція stripAllComments).
3. Паралельно обчислити складові:
   - V (Halstead Volume) — на основі аналізу токенів.
   - G (Cyclomatic Complexity) — на основі обходу AST.
   - LOC (Lines of Code) — підрахунок непорожніх рядків.
4. Якщо LOC = 0, встановити MI = 100 (порожній файл ідеально підтримується).
5. Інакше, застосувати формулу 1.2.
6. Виконати операцію clamp (обмеження), щоб результат був у межах від 0 до 100.

Блок-схему алгоритму обчислення показника Cognitive Complexity (CC) наведено на рисунку 3.6.

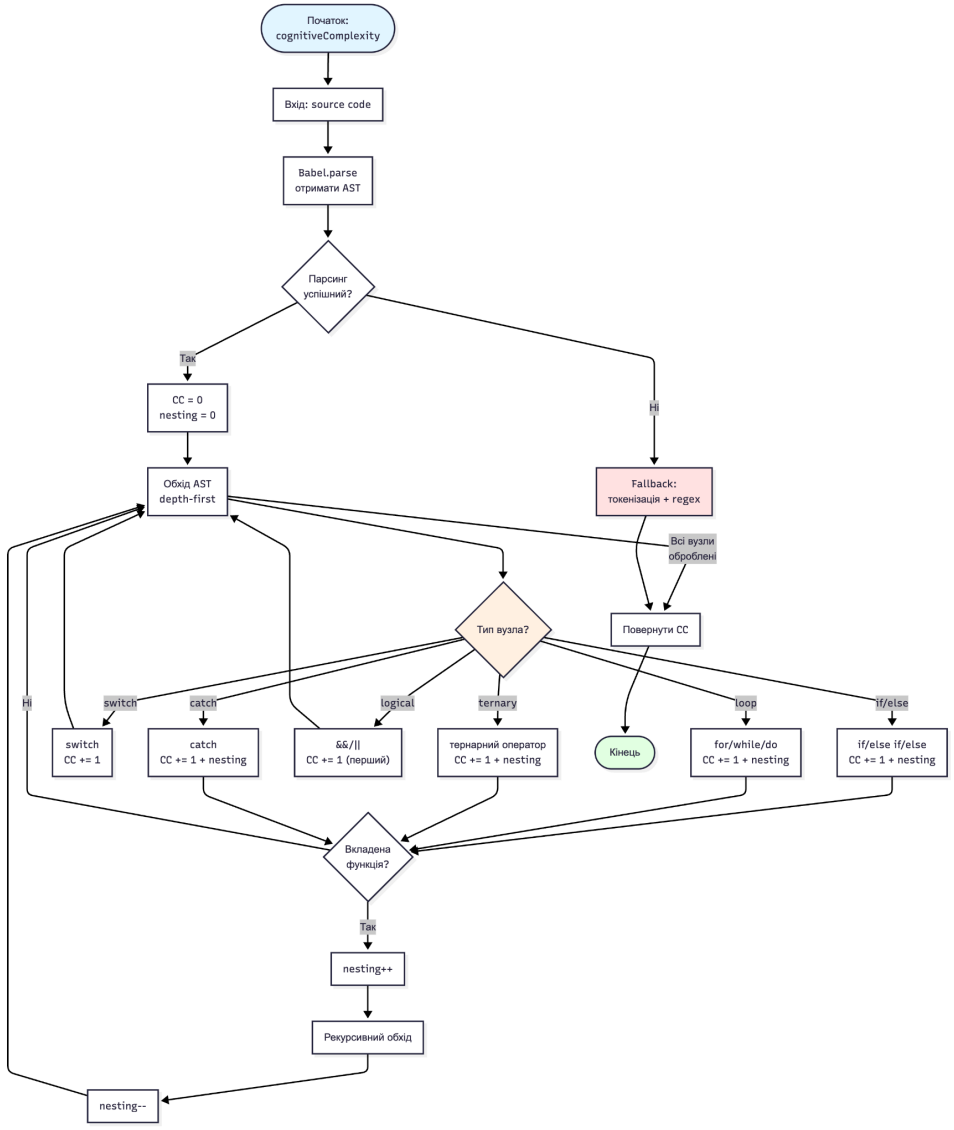


Рисунок 3.6 – Блок-схема алгоритму обчислення Cognitive Complexity

Опис алгоритму (рис. 3.6):

1. Спроба побудувати AST.
2. Якщо успішно:
   - Ініціалізувати score = 0.
   - Запустити рекурсивний обхід дерева.
   - Для кожного вузла перевірити тип:

- if, else if, else, switch: +1 до score.
- Цикли (for, while, do): +1 до score.
- catch: +1 до score.
- Тернарний оператор: +1 до score.
- Логічні оператори (&&, ||): +1 за послідовність.

○ Для кожної конструкції додати поточний рівень вкладеності (nesting) до score.

○ При вході у функцію збільшити nesting.

3. Якщо парсинг не вдався (Fallback):
   ○ Розбити код на токени.
   ○ Підрахувати ключові слова.
   ○ Оцінити вкладеність за балансом фігурних дужок {}.

Псевдокод для підрахунку операторів Halstead:

```
operators_list = ['+', '-', '*', '/', '=', 'if', 'return',...]
operands_list = // змінні, числа, рядки

tokens = getTokens(source_code)

for each token in tokens:
  if token is Operator OR Keyword OR Punctuation:
    add to operators_found
  else if token is Identifier OR Literal:
    add to operands_found

n1 = countUnique(operators_found)
N1 = countTotal(operators_found)
n2 = countUnique(operands_found)
N2 = countTotal(operands_found)
```

Vocabulary = n1 + n2

Length = N1 + N2

Volume = Length * log2(Vocabulary)

3.4 Розробка інтерфейсу користувача

3.4.1 Структура екранів системи

Інтерфейс системи спроєктовано як набір трьох основних сторінок, перемикання між якими відбувається миттєво.

1. Головна сторінка (Code Analyzer) (/):

Це робоче місце дослідника. Центральним елементом є велика область textarea для введення коду з плейсхолдером "Вставте JavaScript-код...".

- Панель інструментів: Містить кнопки "Аналізувати код" (дублюється комбінацією клавіш Ctrl+Enter), "Приклад" (завантажує демо-код), "Завантажити.js" (відкриває діалог вибору файлу) та "Очистити".
- Блок результатів: З'являється після аналізу. Розділений на дві основні картки:
  - Карта MI: Відображає велике значення індексу, кольоровий прогрес-бар, текстовий статус ("Висока підтримуваність") та додаткові показники (V, LOC, G).
  - Карта CC: Відображає значення когнітивної складності та її інтерпретацію.
- Прев'ю коду: Блок з підсвіткою синтаксису (Prism.js), кнопкою копіювання в буфер обміну та перемикачем у повноекранний режим.
- Історія: Стрічка внизу екрана, що показує міні-картки останніх 10 аналізів з можливістю швидкого повернення до них.

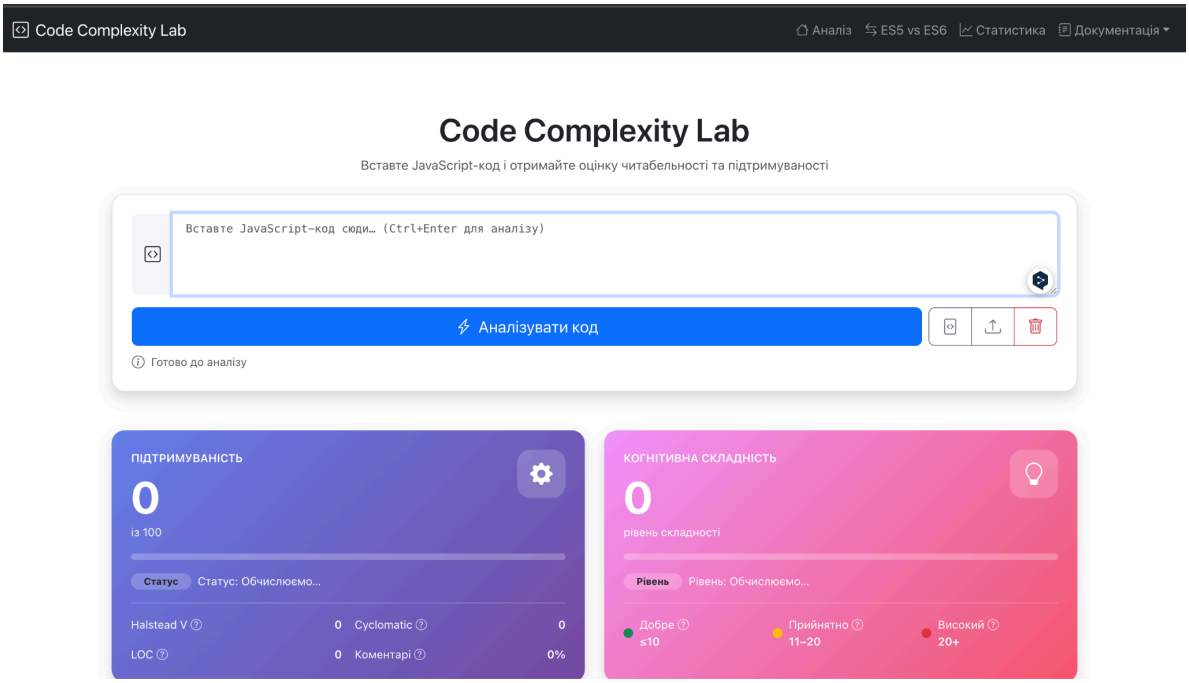


Рисунок 3.7 – Головна сторінка

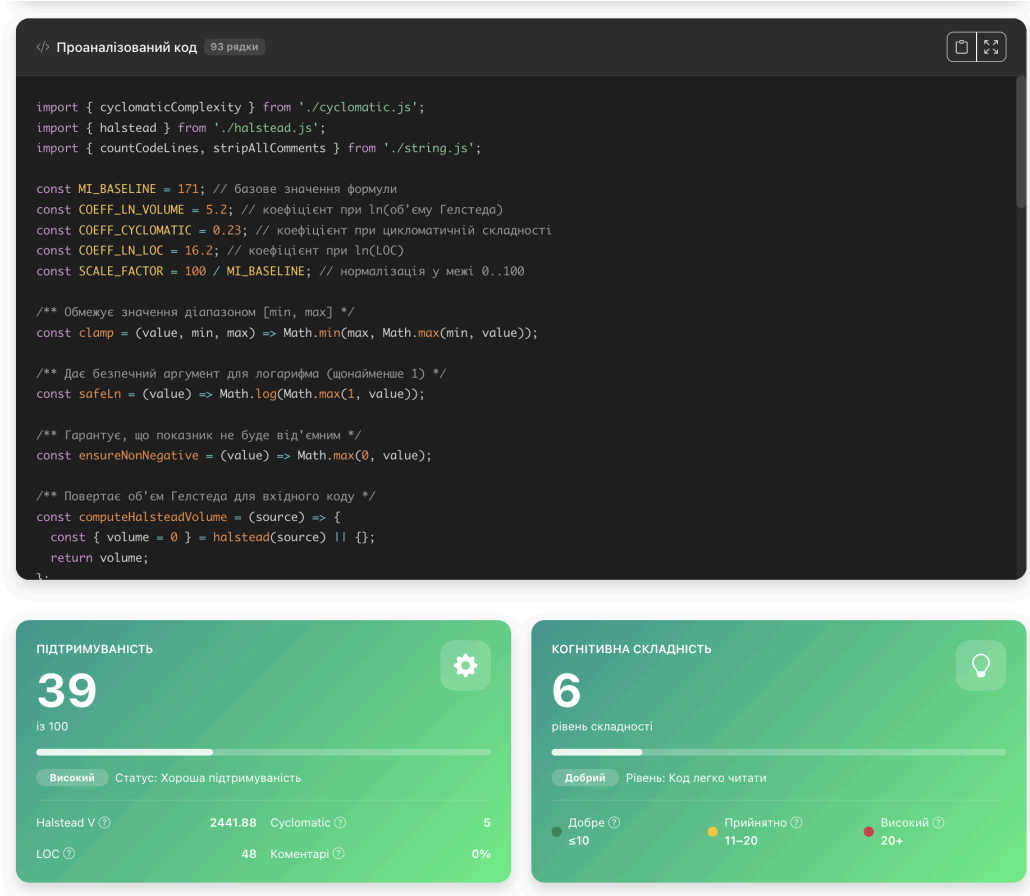


Рисунок 3.8 – Головна сторінка з прев’ю коду та активними картками результатів

2. Сторінка статистики (/stats):

Призначена для аналізу результатів масового експерименту.

— Summary Cards: 4 картки у верхній частині, що показують агреговані середні значення MI, CC, CG, LOC для ES5 та ES6, а також відсоткову зміну (наприклад, "+8.48%").

— Статистична значущість: Блок з результатами тесту Вілкоксона для трьох основних метрик (p-value, Effect Size, кількість випадків покращення/погіршення).

— Графіки: Інтерактивні стовпчикові діаграми Highcharts, що дозволяють візуально порівняти розподіли показників.

— Детальна таблиця: Таблиця з переліком усіх 100 файлів вибірки. Кожен рядок містить назву файлу, значення показників для обох стандартів, розраховану різницю (Δ) та кольорові бейджі. Таблиця підтримує сортування та експорт у CSV.

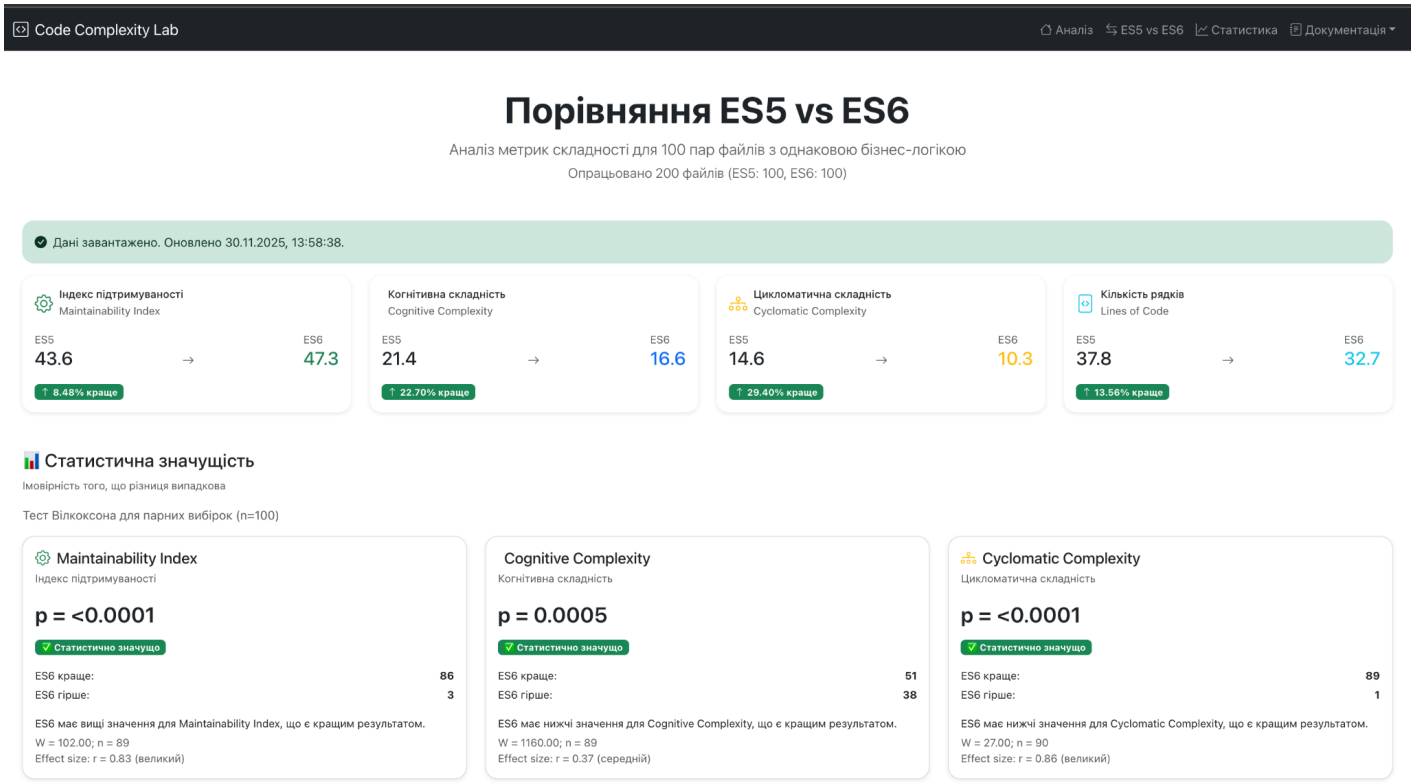


Рисунок 3.9 – Сторінка статистики з порівнянням показників складності та статистичною значущістю результатів

Рисунок 3.10 – Сторінка статистики з графіками порівняння показників складності коду ES5 та ES6

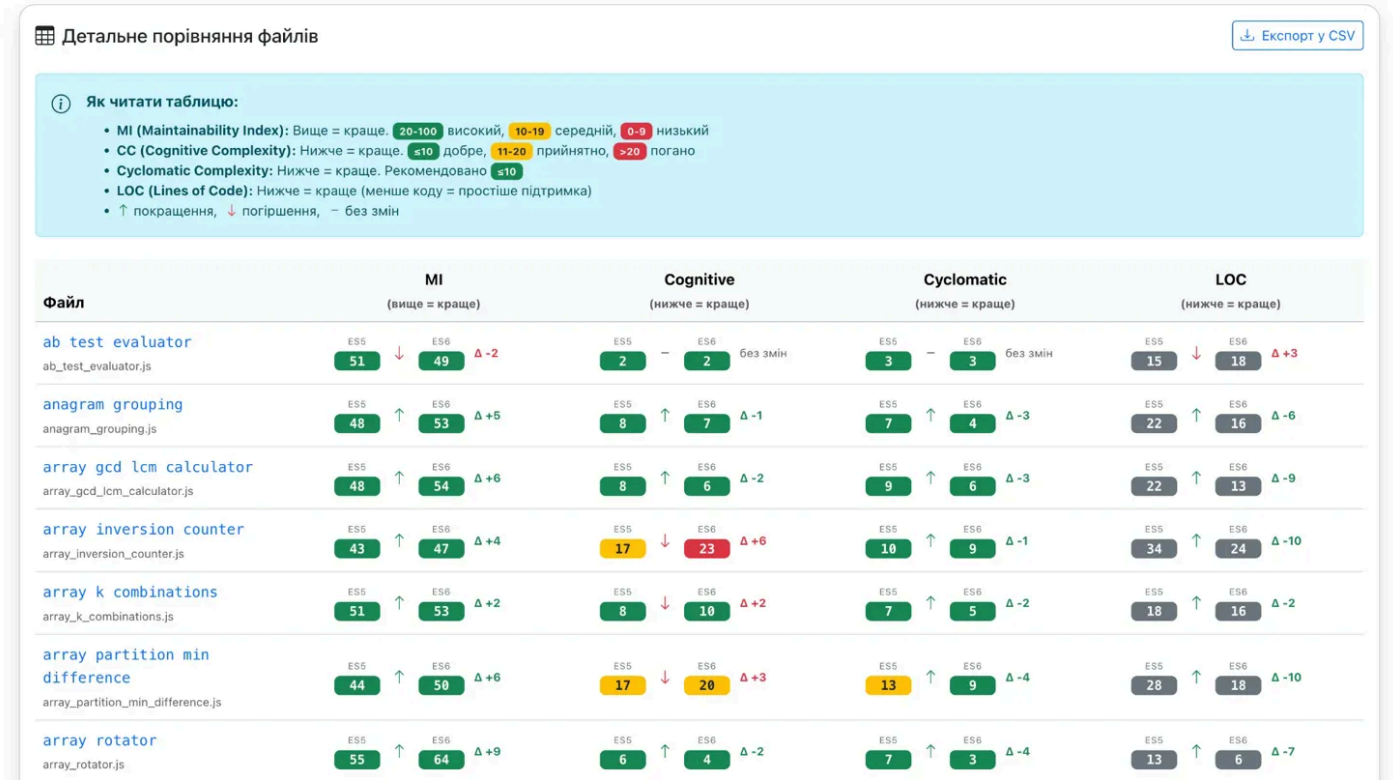


Рисунок 3.11 – Сторінка статистики з таблицею детального порівняння показників складності для кожного прикладу коду

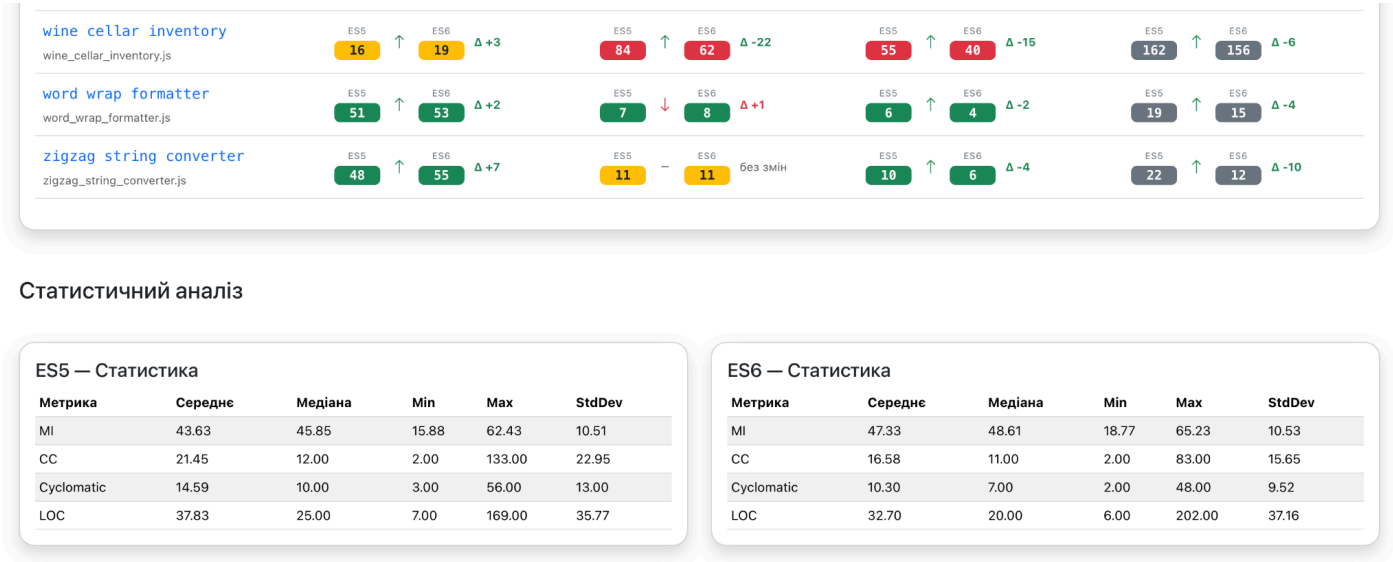


Рисунок 3.12 – Сторінка статистики з таблицею описової статистики показників складності для ES5 та ES6

3.4.2 Реалізація інтерфейсу користувача

Для створення сучасного та зручного UI використано фреймворк Bootstrap 5.3.3.

- UI-компоненти:
  - Progress bars: Використовуються для візуалізації MI та CC. Колір смуги динамічно змінюється через класи bg-success, bg-warning, bg-danger залежно від значення показника.
  - Badges: Використовуються в таблицях та картках для відображення статусів (наприклад, "ES6 Better").
  - Toasts: Спливаючі повідомлення у правому нижньому куті для інформування про помилки парсингу або успішне копіювання коду.
- Інтерактивність:
  - Для оптимізації продуктивності реалізовано патерн Debounce (затримка 350 мс) при введенні коду, щоб не запускати важкий процес парсингу на кожне натискання клавіші.
  - Підсвітка синтаксису реалізована через бібліотеку Prism.js, яка автоматично токенізує код всередині блоків <pre><code>.
- Local Storage:
  - Система автоматично зберігає результати аналізу в локальному сховищі браузера. Структура запису включає: унікальний ID, timestamp, перші 3 рядки коду (для прев'ю), обчислені показники та статуси. Зберігається історія останніх 10 записів, що дозволяє досліднику не втрачати дані при перезавантаженні сторінки.

3.5 Тестування та налагодження програми

3.5.1 Методи тестування

Забезпечення надійності розробленого інструментарію є критично важливим для достовірності наукових результатів. Стратегія тестування базується на двох підходах:

1. Тестування "білої скриньки" (White-box testing): Перевірка внутрішньої логіки алгоритмів, покриття всіх гілок умовних операторів та циклів. Це реалізовано через модульні (unit) тести для кожного файлу з папки utils/.

2. Тестування "чорної скриньки" (Black-box testing): Перевірка правильності роботи системи на основі вхідних та вихідних даних без прив'язки до внутрішньої реалізації. Наприклад, подача на вхід відомого фрагмента коду та звірка отриманого MI з еталонним значенням.

Як інструмент для написання та виконання тестів використано Node.js Test Runner, який є вбудованим у середовище виконання Node.js. Це дозволило уникнути встановлення важких фреймворків (на кшталт Jest) та забезпечити максимальну швидкість виконання тестів.

3.5.2 Приклади тест-кейсів

Всього розроблено 153 юніт-тести. Нижче наведено три детальні приклади, що демонструють перевірку ключових алгоритмів.

Приклад 1: Тест когнітивної складності (обробка if-else if)

Цей тест перевіряє специфічне правило SonarSource, згідно з яким конструкція else if не повинна створювати додаткової вкладеності, на відміну від вкладеного if всередині else.

```
test('Cognitive Complexity - else if та else', async (t) => {
 await t.test('if-else if → 2', () => {
   const code = `
     if (a) {          // +1
       return 1;
     } else if (b) {   // +1 (hybrid, no nesting)
       return 2;
     }
   `;
   const result = cognitiveComplexity(code);
   console.log('if-else if:', result);
   assert.strictEqual(result, 2, 'if +1, else if +1');
 });
```

Приклад 2: Тест цикломатичної складності (Optional Chaining)

Оператор опціонального ланцюжка ?., введений у ES2020, фактично є прихованою умовою. Тест перевіряє, чи враховує парсер ці приховані гілки виконання.

```
test('Optional chaining працює коректно', async (t) => {
 await t.test('a?.b → CC = 2', () => {
   assert.strictEqual(cyclomaticComplexity('a?.b'), 2);
 });

 await t.test('a?.b?.c?.d → CC = 4', () => {
   assert.strictEqual(cyclomaticComplexity('a?.b?.c?.d'), 4);
 });
 await t.test('a?.() → CC = 2', () => {
   assert.strictEqual(cyclomaticComplexity('a?.()'), 2);
 });
 await t.test('змішаний ланцюжок a.b?.c.d?.e → CC = 3', () => {
   assert.strictEqual(cyclomaticComplexity('a.b?.c.d?.e'), 3);
 });
});
```

Приклад 3: Інтеграційний тест Maintainability Index

Перевірка точності розрахунку показника на коді з вкладеними конструкціями. Тест звіряє обчислене значення MI з еталонним для коду з відомою структурою, що містить умовні оператори, цикл та обробку винятків.

```
await t.test('код з вкладеними структурами має очікуване значення MI', () => {
  const code = `
    function complex(a, b, c, d, e) {
      if (a && b || c && d) {
        for (let i = 0; i < 100; i++) {
          if (i % 2 === 0 && i % 3 === 0) {
            try {
              result += process(i);
            } catch (err) {
              if (err.code === 'E1' || err.code === 'E2') {
                handleError(err);
              }
            }
          }
        }
      }
      return result ?? defaultValue;
    }
  `;
  const result = maintainabilityIndex(code);
  const resultMiToInt = parseInt(result.mi, 10)

  assert.strictEqual(resultMiToInt, 58);
});
```

Аналіз покриття коду:

За результатами запуску тестів отримано наступні показники покриття:

— Рядки (Lines): ~96% — переважна більшість логіки покрита тестами.

— Гілки (Branches): ~85% — перевірено всі основні логічні шляхи. Непокритими залишилися лише специфічні try-catch блоки для обробки рідкісних помилок парсера, які важко емулювати у синтетичних тестах.

Висновки до розділу 3

У третьому розділі магістерської роботи виконано проєктування та програмну реалізацію спеціалізованого інструментального забезпечення — веб-застосунку «Code Complexity Lab», призначеного для дослідження впливу стандарту ES6 на підтримуваність JavaScript-коду.

1. Виконані роботи:
   - Проведено формалізацію задачі та розроблено архітектуру системи на основі моделі SPA з клієнтськими обчисленнями. Це рішення забезпечило високу швидкість роботи, конфіденційність досліджуваного коду та незалежність від серверних ресурсів.
   - Реалізовано систему на сучасному технологічному стеку (Vite, Pug, Bootstrap) з використанням потужного парсера @babel/parser, що дозволяє аналізувати найновіші конструкції мови JavaScript.
   - Розроблено модульну структуру ядра аналізу з чітким розділенням бізнес-логіки та інтерфейсу. Реалізовано алгоритми розрахунку п'яти ключових показників: Maintainability Index, Cognitive Complexity, Cyclomatic Complexity, Halstead Volume та LOC.
   - Створено інтуїтивно зрозумілий інтерфейс користувача з можливостями редагування коду, візуалізації результатів, збереження історії та перегляду агрегованої статистики.
2. Ключові рішення:
   - Вибір клієнтської архітектури забезпечив повну відтворюваність експерименту.
   - Використання Babel parser з повним набором плагінів дозволило коректно обробляти специфічні конструкції ES6+, такі як стрілкові функції, деструктуризація та класи.
   - Впровадження fallback-механізмів на основі регулярних виразів підвищило стійкість системи до помилок у вхідному коді.

3. Результати тестування:

   - Розроблено комплект із 153 юніт-тестів, що забезпечили покриття коду на рівні 96%. Тестування підтвердило коректність реалізації складних алгоритмів (зокрема, Cognitive Complexity) та їх відповідність теоретичним моделям.

4. Практична значущість:

   - Створена система повністю готова до використання для проведення основного етапу дослідження.
   - Попередні результати аналізу вибірки зі 100 парних прикладів, отримані за допомогою системи, демонструють статистично значуще покращення підтримуваності (MI +8.48%) та зниження складності (CC -22.7%) при переході на ES6.
   - Інструмент може бути використаний у навчальному процесі для демонстрації показників якості коду та при проведенні code review у реальних проектах.

Розроблене програмне забезпечення створює надійну технологічну базу для проведення експериментального дослідження та подальшого аналізу результатів, що буде викладено у четвертому розділі роботи.

РОЗДІЛ 4 ДОСЛІДЖЕННЯ ВПЛИВУ СТАНДАРТУ ES6 НА ПОКАЗНИКИ СУПРОВОДЖУВАНОСТІ КОДУ

У цьому розділі викладено хід експериментального дослідження, умови та основні етапи проведення аналізу, результати порівняння показників супроводжуваності JavaScript-коду, написаного у стилях ES5 та ES6. Надано оцінку достовірності одержаних результатів із застосуванням методів математичної статистики та обговорено практичне значення виявлених закономірностей.

4.1 Підготовка до експерименту

Підготовка до експерименту передбачала формування репрезентативної вибірки парних прикладів коду, визначення критеріїв відбору та налаштування програмного середовища для автоматизованого аналізу.

4.1.1 Опис вибірки парних прикладів коду

Для проведення експериментального дослідження було сформовано вибірку зі 100 парних прикладів JavaScript-коду. Кожна пара складається з двох функціонально еквівалентних реалізацій: однієї у стилі ES5 та однієї у стилі ES6. Обидві реалізації виконують ідентичну задачу та мають однакову алгоритмічну семантику, проте відрізняються використаними синтаксичними конструкціями.

Приклади охоплюють широкий спектр типових задач програмування: алгоритми обробки масивів, валідатори форм, парсери даних, калькулятори, планувальники, роутери, системи управління станом, обробники подій та інші поширені патерни. Такий підхід забезпечує репрезентативність вибірки щодо реальних сценаріїв використання JavaScript у веб-розробці.

Обсяг прикладів варіюється від простих однофункціональних модулів (7–15 рядків коду) до складних багатофункціональних компонентів (до 200 рядків). Середній обсяг ES5-реалізацій становить 37,83 рядка, ES6-реалізацій — 32,70 рядка. Розподіл за складністю наближено відповідає реальному розподілу задач у практиці розробки.

4.1.2 Критерії формування та відбору прикладів

При формуванні вибірки було дотримано таких критеріїв:

1. функціональна еквівалентність: обидві реалізації пари повинні давати ідентичні результати для будь-яких вхідних даних;

2. алгоритмічна семантика: загальна структура алгоритму зберігається, змінюються лише синтаксичні конструкції;

3. використання характерних конструкцій: ES6-версії активно застосовують стрілкові функції, деструктуризацію, let/const, шаблонні рядки, класи, Map/Set;

4. синтаксична коректність: обидва файли пари повинні успішно парситися без помилок;

5. практична релевантність: задачі відповідають типовим сценаріям веб-розробки.

Такий підхід дозволяє ізолювати вплив саме синтаксичних конструкцій ES6 на показники супроводжуваності, виключаючи вплив алгоритмічних відмінностей.

4.1.3 Опис програмного середовища аналізу

Для проведення експерименту використано розроблений веб-застосунок «Code Complexity Lab», детально описаний у третьому розділі роботи. Застосунок забезпечує автоматизований розрахунок показників супроводжуваності на основі статичного аналізу абстрактного синтаксичного дерева (AST) коду.

Для пакетної обробки вибірки розроблено скрипт analyze-examples.js, який послідовно аналізує всі 100 пар прикладів та формує агреговані результати у форматі JSON. Скрипт виконується в середовищі Node.js v20+ з використанням парсера @babel/parser та зберігає результати у файлі examples-stats.json.

Аналіз виконувався на комп'ютері з процесором Mac M3, 16 ГБ оперативної пам'яті під управлінням операційної системи MacOS Tahoe. Час повного аналізу вибірки зі 100 пар становить менше 2 секунд.

4.2 Методика проведення експерименту

4.2.1 Процедура автоматизованого аналізу парних прикладів

Процедура аналізу складається з таких етапів:

— завантаження вихідного коду ES5 та ES6 версій з відповідних директорій;

— побудова AST для кожного файлу за допомогою @babel/parser;

— обхід AST та обчислення показників: Maintainability Index, когнітивна складність, цикломатична складність, кількість рядків коду;

— збереження результатів у структурованому форматі;

— агрегація даних та обчислення описової статистики.

Для кожної пари зберігаються абсолютні значення всіх показників для обох версій, що дозволяє виконати як попарне порівняння, так і загальний статистичний аналіз.

4.2.2 Обрані показники супроводжуваності

Для кількісної оцінки супроводжуваності коду обрано чотири ключові показники:

Індекс підтримуваності (Maintainability Index, MI) — інтегральний показник, що враховує обсяг коду, цикломатичну складність та показники Гелстеда. Розраховується за формулою Microsoft Visual Studio. Вищі значення MI свідчать про кращу підтримуваність: значення понад 20 вважаються прийнятними, понад 40 — добрими.

Когнітивна складність (Cognitive Complexity, CC) — показник, розроблений компанією SonarSource, що оцінює зусилля, необхідні програмісту для розуміння коду. На відміну від цикломатичної складності, враховує вкладеність структур та когнітивне навантаження. Нижчі значення свідчать про легший для розуміння код.

Цикломатична складність (Cyclomatic Complexity) — класичний показник Маккейба, що визначає кількість незалежних шляхів виконання в програмі. Враховує умовні оператори, цикли та логічні вирази. Нижчі значення свідчать про простішу логіку коду.

Кількість рядків коду (Lines of Code, LOC) — показник обсягу програми, що враховує логічні рядки без коментарів та порожніх рядків. Використовується як допоміжний показник для оцінки компактності коду.

4.2.3 Обґрунтування використання тесту Вілкоксона для парних вибірок

Для статистичної перевірки гіпотези про значущість відмінностей між показниками ES5 та ES6 реалізацій обрано непараметричний критерій Вілкоксона для парних вибірок (Wilcoxon signed-rank test).

Вибір цього критерію обумовлено такими факторами:

— дані є парними: кожна ES5-реалізація має відповідну ES6-реалізацію;

— розподіл різниць може не бути нормальним (показники складності часто мають асиметричний розподіл);

— критерій є стійким до викидів та не вимагає припущень про форму розподілу;

— обсяг вибірки (n = 100) є достатнім для застосування критерію.

Рівень значущості встановлено $\alpha = 0,05$. Нульова гіпотеза H_0 стверджує, що медіана різниць між парними спостереженнями дорівнює нулю. Альтернативна гіпотеза H_1 — медіана різниць відмінна від нуля. Інтерпретація величини ефекту за Коеном:

- $r < 0,3$ — малий ефект;
- $0,3 \leq r < 0,5$ — середній ефект;
- $r \geq 0,5$ — великий ефект.

4.3 Результати експерименту

4.3.1 Описова статистика показників

За результатами аналізу 100 парних прикладів отримано описову статистику показників супроводжуваності для ES5 та ES6 реалізацій. Результати наведено у таблиці 4.1.

Таблиця 4.1 – Описова статистика показників супроводжуваності

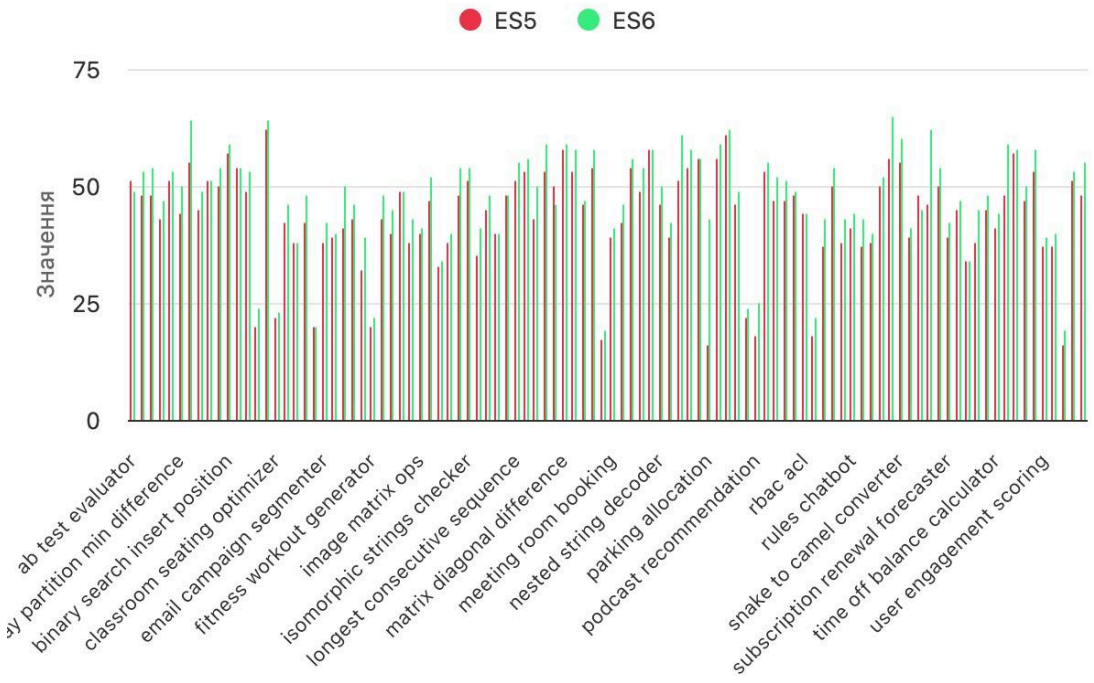
Показник	Середнє	Медіана	Min	Max	StdDev
MI (ES5)	43,63	45,85	15,88	62,43	10,51
MI (ES6)	47,33	48,61	18,77	65,23	10,53
CC (ES5)	21,45	12,00	2,00	133,00	22,95
CC (ES6)	16,58	11,00	2,00	83,00	15,65
Cyclomatic (ES5)	14,59	10,00	3,00	56,00	13,00
Cyclomatic (ES6)	10,30	7,00	2,00	48,00	9,52
LOC (ES5)	37,83	25,00	7,00	169,00	35,77
LOC (ES6)	32,70	20,00	6,00	202,00	37,16

З таблиці видно, що середні значення всіх показників демонструють покращення підтримуваності при переході від ES5 до ES6: індекс підтримуваності зростає (з 43,63 до 47,33), тоді як показники складності та обсягу зменшуються. Стандартні відхилення

залишаються порівнянними, що свідчить про стабільність ефекту на різних типах задач.

4.3.2 Порівняльний аналіз за індексом підтримуваності

Індекс підтримуваності (MI) є ключовим інтегральним показником, що відображає загальну якість коду з точки зору супроводжуваності. На рисунку 4.1 наведено порівняння значень MI для всіх 100 парних прикладів.



по центру назву

Рисунок 4.1 – Порівняння індексу підтримуваності ES5 та ES6 реалізацій

Аналіз результатів показує, що у 86 з 100 прикладів (86%) ES6-реалізації мають вищий індекс підтримуваності порівняно з ES5-версіями. Лише у 3 випадках (3%) спостерігається погіршення, у 11 випадках (11%) значення залишилися незмінними.

Найбільше покращення MI зафіксовано у прикладі parking_allocation.js: ES5-версія має MI = 16, тоді як ES6-версія — MI = 43, що становить приріст у 27 одиниць. Це пояснюється заміною багаторівневих вкладених циклів та умовних конструкцій на методи масивів (map, filter, reduce) та деструктуризацію.

Інші приклади зі значним покращенням: string_kebab_case_converter.js (+16), top_k_frequent_numbers.js (+11), palindrome_permutation_checker.js (+10), array_rotator.js

(+9). У цих випадках ES6-конструкції дозволили суттєво спростити код без втрати функціональності.

Випадки погіршення (markdown_parser.js: -4, state_machine.js: -3, ab_test_evaluator.js: -2) пов'язані з особливостями конкретних реалізацій, де ES6-синтаксис не дав переваг або призвів до незначного ускладнення через надмірне використання деструктуризації.

4.3.3 Порівняльний аналіз за когнітивною складністю

Когнітивна складність відображає зусилля, необхідні для розуміння коду. На рисунку 4.2 наведено порівняння значень когнітивної складності для парних прикладів.

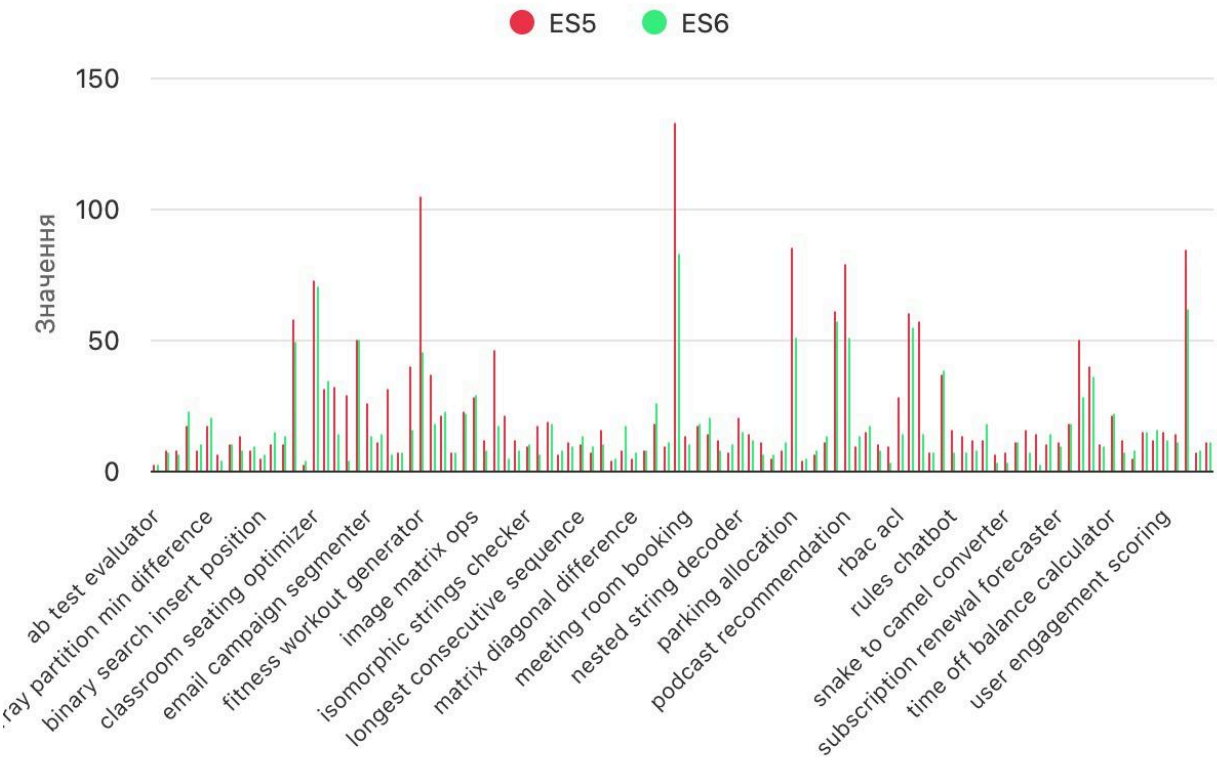


Рисунок 4.2 – Порівняння когнітивної складності ES5 та ES6 реалізацій

За результатами аналізу у 51 прикладі (51%) ES6-версії мають нижчу когнітивну складність, у 38 прикладах (38%) — вищу, у 11 прикладах (11%) значення однакові. Зниження середнього значення когнітивної складності становить 22,70% (з 21,45 до 16,58).

Найбільш вражаюче покращення спостерігається у прикладі fitness_workout_generator.js: когнітивна складність знизилася зі 105 до 45, тобто на 60 одиниць. Подібне значне покращення демонструють medical_test_interpreter.js (-50), recommender.js (-43), parking_allocation.js (-34), inventory_manager.js (-29).

Ці приклади характеризуються складною вкладеною логікою в ES5-версіях, яку вдалося суттєво спростити завдяки стрілковим функціям та методам масивів. Зокрема, заміна вкладених циклів for на ланцюжки методів map/filter/reduce зменшує рівень вкладеності, що безпосередньо впливає на когнітивну складність.

Випадки, де ES6 показує вищу когнітивну складність, здебільшого пов'язані з додаванням більш детальної обробки помилок або розширенням функціональності в процесі рефакторингу, що не є прямим наслідком зміни синтаксису.

4.3.4 Порівняльний аналіз за цикломатичною складністю

Цикломатична складність визначає кількість незалежних шляхів виконання в програмі. На рисунку 4.3 наведено порівняння значень цикломатичної складності.

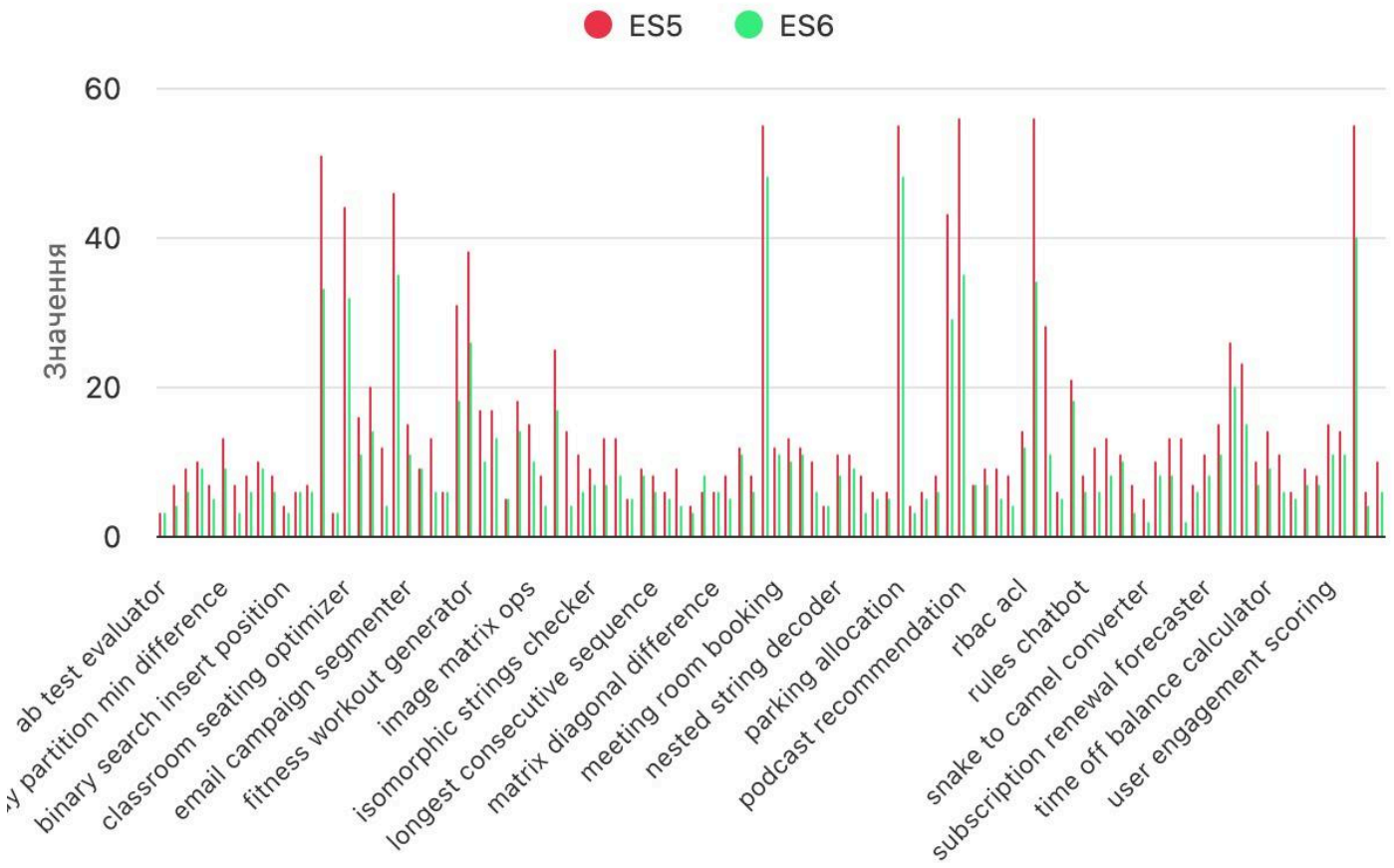


Рисунок 4.3 – Порівняння цикломатичної складності ES5 та ES6 реалізацій

Результати демонструють найбільш виражене покращення серед усіх досліджуваних показників: у 89 прикладах (89%) ES6-версії мають нижчу цикломатичну складність, лише в 1 прикладі (1%) — вищу. Зниження середнього значення становить 29,40% (з 14,59 до 10,30).

Таке значне покращення пояснюється особливостями синтаксису ES6. Стрілкові функції з неявним поверненням замінюють блоки з явним return, методи масивів (forEach, map, filter) замінюють цикли з умовами, оператор розширення (...) спрощує операції з масивами.

Найбільше зниження цикломатичної складності зафіксовано у тих самих прикладах, що демонструють значне покращення когнітивної складності: fitness_workout_generator.js, medical_test_interpreter.js, recommender.js. Це підтверджує кореляцію між показниками та консистентність ефекту ES6.

4.3.5 Порівняльний аналіз за кількістю рядків коду

Кількість рядків коду є допоміжним показником, що характеризує компактність реалізації. На рисунку 4.4 наведено порівняння обсягу коду для парних прикладів.

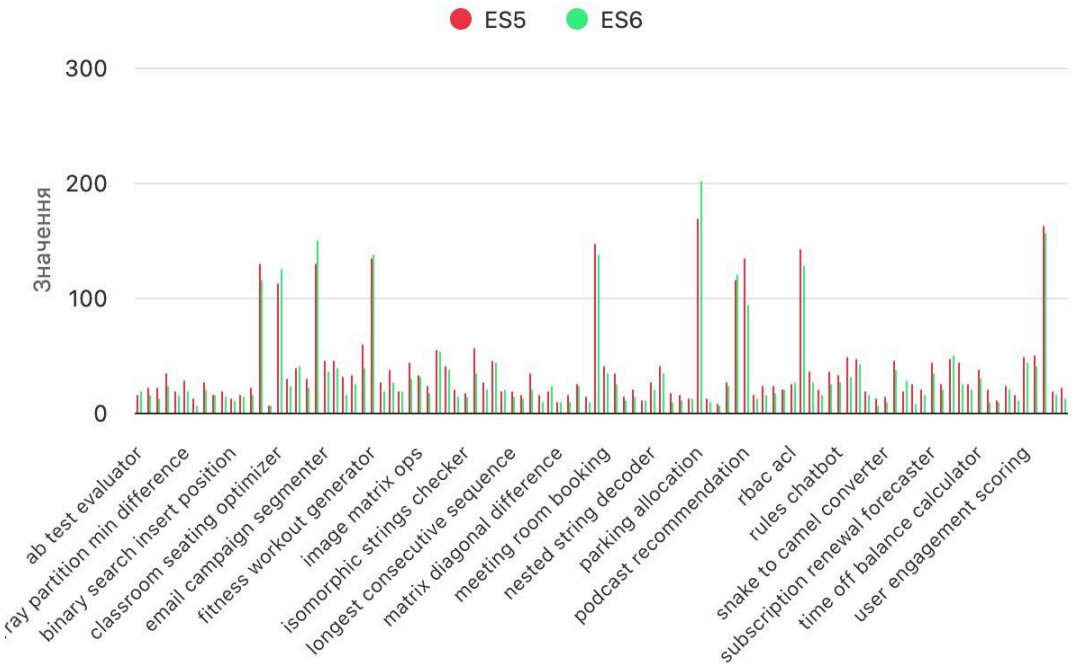


Рисунок 4.4 – Порівняння кількості рядків коду ES5 та ES6 реалізацій

Аналіз показує, що ES6-реалізації в середньому на 13,56% компактніші за ES5-версії (32,70 проти 37,83 рядків). Це досягається завдяки:

— стрілковим функціям з неявним поверненням, що усувають потребу в ключових словах function та return;

— деструктуризації, що спрощує доступ до властивостей об'єктів;

— шаблонним рядкам, що замінюють конкатенацію;

— скороченому синтаксису методів об'єктів та класів.

Варто зауважити, що зменшення кількості рядків не є самоціллю і має супроводжуватися покращенням читабельності. У досліджуваних прикладах скорочення обсягу коду корелює з покращенням індексу підтримуваності, що свідчить про позитивний вплив на загальну якість.

4.3.6 Статистична значущість результатів

Для перевірки статистичної значущості виявлених відмінностей застосовано тест Вілкоксона для парних вибірок. Результати наведено у таблиці 4.2.

Таблиця 4.2 – Результати статистичного аналізу (тест Вілкоксона)

Показник	p-value	W	Effect size r	n
Maintainability Index	< 0,0001	102,00	0,83 (великий)	89
Cognitive Complexity	0,0005	1160,00	0,37 (середній)	89
Cyclomatic Complexity	< 0,0001	27,00	0,86 (великий)	90

Усі отримані p-значення менші за встановлений рівень значущості $\alpha = 0,05$, що дозволяє відхилити нульову гіпотезу про відсутність відмінностей між ES5 та ES6 реалізаціями. Виявлені відмінності є статистично значущими.

Величина ефекту для індексу підтримуваності (r = 0,83) та цикломатичної складності (r = 0,86) інтерпретується як «великий ефект» за класифікацією Коена. Це означає, що спостережувані відмінності мають не лише статистичну, а й практичну значущість.

Для когнітивної складності величина ефекту (r = 0,37) відповідає «середньому ефекту». Менша величина ефекту пояснюється більшою варіабельністю цього показника та тим, що не всі ES6-конструкції однаково впливають на когнітивне навантаження.

4.3.7 Узагальнення результатів

Зведені результати порівняння ES5 та ES6 реалізацій наведено у таблиці 4.3.

Таблиця 4.3 – Узагальнені результати порівняння ES5 та ES6

Показник	ES5	ES6	Зміна	ES6 краще
Maintainability Index	43,6	47,3	+8,48%	86 / 100
Cognitive Complexity	21,4	16,6	-22,70%	51 / 100
Cyclomatic Complexity	14,6	10,3	-29,40%	89 / 100
Lines of Code	37,8	32,7	-13,56%	83 / 100

Результати підтверджують дослідницьку гіпотезу: використання конструкцій ES6 призводить до статистично значущого покращення показників супроводжуваності JavaScript-коду порівняно з еквівалентними ES5-реалізаціями.

4.4 Обговорення результатів

4.4.1 Інтерпретація отриманих даних

Отримані результати дозволяють сформулювати такі висновки щодо впливу ES6 на супроводжуваність коду:

По-перше, найбільший позитивний вплив ES6 має на цикломатичну складність (зниження на 29,40%). Це пов'язано з тим, що методи масивів (map, filter, reduce, forEach) та стрілкові функції дозволяють замінити імперативні цикли з умовними конструкціями на декларативні вирази, які не створюють додаткових точок розгалуження.

По-друге, когнітивна складність демонструє середній ефект покращення (22,70%). Хоча ES6-код загалом легший для розуміння, деякі конструкції (глибока деструктуризація, складні стрілкові функції) можуть підвищувати когнітивне навантаження. Це пояснює більшу варіабельність результатів для цього показника.

По-третє, інтегральний індекс підтримуваності зростає на 8,48%, що відображає комплексний позитивний вплив ES6 на якість коду. Зростання MI є наслідком зменшення обсягу коду, зниження складності та покращення показників Гелстеда.

По-четверте, зменшення кількості рядків коду (13,56%) досягається без втрати читабельності. ES6-синтаксис дозволяє виразити ту саму логіку компактніше, що полегшує огляд коду та зменшує ймовірність помилок.

4.4.2 Приклади з найбільшим та найменшим покращенням

Детальний аналіз прикладів з екстремальними значеннями покращення дозволяє виявити фактори, що впливають на ефективність переходу до ES6.

Приклади з найбільшим покращенням характеризуються:

— складною вкладеною логікою в ES5-версії (багаторівневі цикли, глибокі умовні конструкції);

— інтенсивною роботою з масивами та об'єктами;

— можливістю застосування функціонального стилю програмування;

— наявністю повторюваних патернів, що спрощуються завдяки деструктуризації.

Приклади з мінімальним покращенням або погіршенням характеризуються:

— простою лінійною логікою без вкладених структур;

— специфічними патернами, що не піддаються спрощенню (парсери, кінцеві автомати);

— надмірним або неоптимальним використанням ES6-конструкцій.

Ці спостереження свідчать, що сам по собі синтаксис ES6 не гарантує покращення якості коду. Ефективність переходу залежить від характеру задачі та продуманості рефакторингу.

4.4.3 Обмеження дослідження та перспективи

Проведене дослідження має певні обмеження, які слід враховувати при інтерпретації результатів:

— вибірка обмежена 100 прикладами, хоча статистичний аналіз підтверджує її репрезентативність;

— приклади охоплюють переважно функціональні модулі, а не великі застосунки;

— не досліджено вплив на продуктивність виконання коду;

— не враховано суб'єктивне сприйняття читабельності розробниками.

Перспективними напрямами подальших досліджень є:

— розширення вибірки до реальних проектів з відкритим кодом;

— дослідження впливу новіших стандартів ECMAScript (ES2020+);

— проведення експериментів із залученням розробників для оцінки суб'єктивної читабельності;

— інтеграція розробленого інструменту з системами безперервної інтеграції.

Висновки до розділу 4

У розділі викладено хід та результати експериментального дослідження впливу стандарту ES6 на показники супроводжуваності JavaScript-коду. На основі аналізу 100 парних прикладів функціонально еквівалентних ES5- та ES6-реалізацій встановлено такі закономірності:

Виявлено статистично значуще покращення всіх досліджуваних показників супроводжуваності при переході від ES5 до ES6:

— індекс підтримуваності (MI) зріс у середньому на 8,48% (з 43,6 до 47,3), при цьому у 86% прикладів ES6-версії мають вищий MI;

— когнітивна складність знизилася на 22,70% (з 21,4 до 16,6), покращення спостерігається у 51% прикладів;

— цикломатична складність зменшилася на 29,40% (з 14,6 до 10,3), ES6 краще у 89% прикладів;

— кількість рядків коду скоротилася на 13,56% (з 37,8 до 32,7) без втрати функціональності.

Статистичну значущість результатів підтверджено тестом Вілкоксона для парних вибірок:

— для індексу підтримуваності $p < 0,0001$, величина ефекту $r = 0,83$ (великий);

— для когнітивної складності $p = 0,0005$, величина ефекту $r = 0,37$ (середній);

— для цикломатичної складності $p < 0,0001$, величина ефекту $r = 0,86$ (великий).

Виявлено, що найбільший позитивний ефект ES6 спостерігається у коді зі складною вкладеною логікою та інтенсивною обробкою даних, де можливе застосування методів масивів та функціонального стилю. Водночас для простого лінійного коду або специфічних патернів (парсери, кінцеві автомати) ефект є мінімальним.

Практичне значення результатів полягає в обґрунтуванні рекомендацій щодо модернізації JavaScript-кодової бази. Перехід на ES6 доцільний передусім для модулів із високою цикломатичною та когнітивною складністю, де очікуване покращення показників є найбільшим.

Результати дослідження підтверджують дослідницьку гіпотезу та демонструють, що стандарт ES6 забезпечує не лише синтаксичні зручності, а й вимірюване покращення якості коду з точки зору супроводжуваності.

ВИСНОВКИ

У магістерській кваліфікаційній роботі вирішено актуальне науково-практичне завдання дослідження впливу стандарту ECMAScript 2015 (ES6) на супроводжуваність JavaScript-коду шляхом кількісного порівняння показників складності парних реалізацій у стилях ES5 та ES6. За результатами виконаної роботи сформульовано такі висновки:

1. Проведено аналіз сучасного стану проблеми супроводжуваності програмного забезпечення та існуючих підходів до її кількісної оцінки. Встановлено, що незважаючи на широке впровадження стандарту ES6 у практику веб-розробки, питання кількісного порівняння супроводжуваності коду, написаного у стилях ES5 та ES6, залишаються недостатньо дослідженими. Обґрунтовано доцільність використання показників Maintainability Index, когнітивної та цикломатичної складності для оцінки супроводжуваності JavaScript-коду.

2. Розроблено методику експериментального дослідження впливу ES6 на супроводжуваність коду, що базується на порівнянні парних функціонально еквівалентних реалізацій. Сформульовано дослідницьку гіпотезу та визначено критерії формування вибірки. Обґрунтовано застосування непараметричного критерію Вілкоксона для статистичної перевірки значущості результатів та величини ефекту за класифікацією Коена для оцінки практичної значущості.

3. Спроєктовано та реалізовано веб-застосунок «Code Complexity Lab» для автоматизованого обчислення показників супроводжуваності JavaScript-коду. Застосунок побудовано на архітектурі Single Page Application з клієнтськими обчисленнями на основі парсера @babel/parser. Реалізовано алгоритми розрахунку п'яти ключових показників: Maintainability Index, Cognitive Complexity, Cyclomatic Complexity, Halstead Volume та Lines of Code. Якість реалізації підтверджено комплектом із 153 юніт-тестів з покриттям коду на рівні 96%.

4. Проведено експериментальне дослідження на вибірці зі 100 парних прикладів JavaScript-коду. Встановлено статистично значуще покращення всіх досліджуваних показників супроводжуваності при переході від ES5 до ES6:

— індекс підтримуваності (MI) зріс на 8,48% (з 43,6 до 47,3), $p < 0,0001$, величина ефекту $r = 0,83$ (великий), покращення у 86% прикладів;

— когнітивна складність знизилася на 22,70% (з 21,4 до 16,6), $p = 0,0005$, величина ефекту $r = 0,37$ (середній), покращення у 51% прикладів;

— цикломатична складність зменшилася на 29,40% (з 14,6 до 10,3), $p < 0,0001$, величина ефекту $r = 0,86$ (великий), покращення у 89% прикладів;

— кількість рядків коду скоротилася на 13,56% (з 37,8 до 32,7) без втрати функціональності.

5. Виявлено, що найбільший позитивний ефект ES6 спостерігається у коді зі складною вкладеною логікою та інтенсивною обробкою масивів і об'єктів, де можливе застосування методів масивів (map, filter, reduce) та функціонального стилю програмування. Найбільше покращення MI (+27 одиниць) зафіксовано у прикладі parking_allocation.js, найбільше зниження когнітивної складності (-60 одиниць) — у fitness_workout_generator.js. Водночас для простого лінійного коду або специфічних патернів ефект є мінімальним.

6. Результати дослідження підтверджують дослідницьку гіпотезу: використання синтаксичних конструкцій ES6 (стрілкові функції, let/const, деструктуризація, шаблонні рядки, класи, колекції Map/Set) за умови збереження алгоритмічної семантики призводить до вимірюваного покращення показників супроводжуваності JavaScript-коду.

Практичне значення одержаних результатів полягає в:

— обґрунтуванні рекомендацій щодо модернізації JavaScript-кодової бази: перехід на ES6 доцільний передусім для модулів із високою цикломатичною та когнітивною складністю;

— створенні інструменту «Code Complexity Lab», що може використовуватися в навчальному процесі для демонстрації показників якості коду та при проведенні code review у реальних проєктах;

— формуванні методики порівняльного аналізу супроводжуваності коду, що може бути адаптована для дослідження інших мов програмування та стандартів.

Перспективними напрямами подальших досліджень є: розширення вибірки до реальних проєктів з відкритим кодом; дослідження впливу новіших стандартів ECMAScript (ES2020+); проведення експериментів із залученням розробників для оцінки суб'єктивної читабельності; інтеграція розробленого інструменту з системами безперервної інтеграції.

джерела оформити згідно вимог

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. SonarQube про цикломатичну складність. [Електронний ресурс] / Режим доступу : URL : - https://www.sonarsource.com/resources/library/cyclomatic-complexity/

2. Офіційна документація Microsoft / Maintainability Index / Режим доступу : URL : - https://learn.microsoft.com/en-us/visualstudio/code-quality/code-metrics-maintainability-index-range-and-meaning

3. Офіційна документація SonarQube / Cognitive & cyclomatic complexity. [Електронний ресурс] / Режим доступу : URL : - https://docs.sonarsource.com/sonarqube-server/user-guide/code-metrics/metrics-definition;

4. SonarSource. Cognitive Complexity. White Paper. PDF. [Електронний ресурс] / Режим доступу : URL : - https://www.sonarsource.com/docs/CognitiveComplexity.pdf;

5. Офіційна документація ESLint. [Електронний ресурс] / Режим доступу : URL : - https://eslint.org/docs/latest/rules/

6. Code Climate: платформа статичного аналізу та якості коду. [Електронний ресурс] / Режим доступу : URL : - https://www.inapps.net/code-climate-puts-static-analysis-at-developers-fingertips-inapps-2022/

7. ECMAScript 2015 Language Specification. PDF. [Електронний ресурс] / Режим доступу : URL : - https://ecma-international.org/wp-content/uploads/ECMA-262_6th_edition_june_2015.pdf

8. Wilcoxon signed-rank test. Numiqo Specefication. [Електронний ресурс] / Режим доступу : URL : - https://numiqo.com/tutorial/wilcoxon-test

ДОДАТОК А

Технічне завдання

ЗАТВЕРДЖУЮ
Перший проректор Українського
державного
університету науки і технологій
Анатолій РАДКЕВИЧ

“CODE COMPLEXITY LAB”

Технічне завдання

ЛИСТ ЗАТВЕРДЖЕННЯ
44165850.1553 – 01 – ЛЗ

Завідувач кафедри КІТ
_____Вадим ГОРЯЧКІН
Керівник розробки
_____Віктор ШАРАВАРА
Виконавець
_____Михайло СВИРИДОВ
Нормоконтролер
_____Світлана ВОЛКОВА

2026

ЗАТВЕРДЖЕНО

44165850.1553 – 01

“CODE COMPLEXITY LAB”

Технічне завдання

Листів 15

2026

ЗМІСТ

ВСТУП

Програма: "CODE COMPLEXITY LAB"

Назва програми: Code Complexity Lab.

Область застосування: Статичний аналіз JavaScript-коду, дослідження впливу стандарту ES6 на підтримуваність програмного забезпечення, порівняння показників складності ES5 та ES6 реалізацій, навчальний процес та code review.

Об'єкт, у якому використовують програму: Процеси розробки та супроводу JavaScript-застосунків, академічні дослідження якості програмного коду, навчальний процес у галузі програмної інженерії та веб-розробки.

Причини розробки: Необхідність кількісного дослідження впливу синтаксичних конструкцій стандарту ES6 на підтримуваність JavaScript-коду порівняно з ES5, відсутність інструментів для комплексного браузерного аналізу показників підтримуваності без серверної обробки, потреба в інструменті для академічних досліджень якості коду та навчальних демонстрацій.

Ключові слова і терміни: Показники підтримуваності, Maintainability Index, когнітивна складність, цикломатична складність, показники Гелстеда, статичний аналіз коду, абстрактне синтаксичне дерево (AST), JavaScript ES5/ES6, браузерні обчислення, веб-застосунок, Single Page Application (SPA).

1 ПІДСТАВА ДЛЯ РОЗРОБКИ

Основою для розробки є наказ проректора Українського державного університету науки і технології Радкевич А.В. «Про затвердження тем та призначення керівників дипломних проектів» №1401ст від 02.10.25

Тема проекту: "Дослідження впливу стандарту ES6 у програмуванні мовою JavaScript на можливості супроводу коду".

Керівник – Шаравара В.В.

2 ПРИЗНАЧЕННЯ РОЗРОБКИ

2.1 Функціональне призначення

Code Complexity Lab розробляється з метою забезпечення дослідників, розробників програмного забезпечення та викладачів потужним інструментом для кількісної оцінки підтримуваності JavaScript-коду:

— обчислення комплексу показників підтримуваності: Maintainability Index (MI), Cognitive Complexity (CC), Cyclomatic Complexity, Halstead Volume та Lines of Code (LOC);

— порівняльний аналіз ES5 та ES6 реалізацій з однаковою функціональністю;

— статичний аналіз коду на основі абстрактного синтаксичного дерева (AST) з використанням парсера Babel;

— візуалізація результатів аналізу у вигляді інтерактивних карток, таблиць та діаграм;

— збереження історії останніх 10 аналізів у локальному сховищі браузера;

— пакетний аналіз парних прикладів ES5/ES6 з формуванням статистичних звітів;

— підсвітка синтаксису коду для покращення читабельності.

2.2 Експлуатаційне призначення

Code Complexity Lab спрямований на надання користувачам таких переваг:

— автоматизація процесу оцінювання підтримуваності коду, що усуває необхідність ручних розрахунків складних математичних показників;

— забезпечення повної конфіденційності досліджуваного коду завдяки виконанню всіх обчислень у браузері без передачі даних на сервер;

— підвищення якості навчального процесу через наочну демонстрацію впливу різних синтаксичних конструкцій на показники складності;

— прискорення процесу code review та технічного аудиту JavaScript-проєктів;

— зменшення технічного боргу завдяки обґрунтованому плануванню рефакторингу на основі кількісних показників.

Ці можливості та переваги нададуть користувачам Code Complexity Lab необхідні інструменти для наукового дослідження підтримуваності коду, прийняття обґрунтованих технічних рішень та підвищення загальної якості JavaScript-проєктів.

3 ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

3.1 Вимоги до функціональних характеристик

Програма повинна виконувати наступні функції:

— парсинг JavaScript-коду з побудовою абстрактного синтаксичного дерева (AST) через бібліотеку @babel/parser з підтримкою плагінів JSX, TypeScript, decorators, optional chaining;

— обчислення показника Maintainability Index за формулою Visual Studio з нормалізацією до діапазону 0-100;

— обчислення когнітивної складності згідно зі специфікацією SonarSource v1.7 з урахуванням вкладеності та гібридних конструкцій;

— обчислення цикломатичної складності з підтримкою сучасних конструкцій ES6+ (стрілкові функції, optional chaining, nullish coalescing);

— обчислення показників Гелстеда (Volume) на основі підрахунку унікальних операторів та операндів;

— підрахунок логічних рядків коду (LOC) з виключенням коментарів та порожніх рядків;

— підсвітка синтаксису введеного коду за допомогою бібліотеки Prism.js;

— збереження результатів аналізу в localStorage з обмеженням історії до 10 записів;

— відображення результатів у вигляді кольорових карток з градієнтами залежно від значень показників;

— побудова інтерактивних діаграм та таблиць на сторінці статистики;

— пакетний аналіз прикладів через CLI-скрипт з формуванням JSON-звітів.

Вхідні дані: JavaScript-код вводиться користувачем у текстове поле редактора коду в веб-інтерфейсі або завантажується з локальних файлів у форматі .js.

Вихідні дані представлені у вигляді:

— числових значень показників (MI, CC, Cyclomatic Complexity, Halstead Volume, LOC);

— візуальних карток з кольоровим кодуванням (зелений/жовтий/червоний залежно від діапазону значень);

— таблиць порівняння парних прикладів ES5/ES6;

— діаграм розподілу показників за допомогою бібліотеки Highcharts.

Часові характеристики:

— миттєве (до 100 мс) обчислення показників для фрагментів коду до 500 рядків;

— автоматичний перерахунок з затримкою 350 мс (debounce) під час введення коду для підвищення продуктивності;

— збереження в localStorage відбувається синхронно після кожного успішного аналізу.

Інші вимоги:

— застосунок повинен працювати повністю автономно в браузері без підключення до мережі після початкового завантаження;

— підтримка копіювання результатів у буфер обміну одним кліком.

3.2 Вимоги до надійності

— захист від втрати даних: всі результати аналізу автоматично зберігаються в localStorage браузера з можливістю відновлення після випадкового закриття вкладки;

— резервне копіювання: вихідний код програми та набір тестових прикладів повинні зберігатися в системі контролю версій Git з віддаленим репозиторієм на GitHub;

— валідація результатів: всі алгоритми обчислення показників покриті набором з юніт-тестів, що гарантує коректність обчислень.

3.3 Умови експлуатації

Оптимальна температура навколишнього середовища має становити від 18 до 25°C, відносна вологість – від 40 до 70%, що відповідає стандартним умовам експлуатації комп'ютерної техніки в офісних приміщеннях та навчальних аудиторіях.

Програмне забезпечення:

— необхідний сучасний веб-браузер з підтримкою ECMAScript 2020+: Google Chrome версії 90+, Mozilla Firefox версії 88+, Microsoft Edge версії 90+, Safari версії 14+ або їх еквіваленти;

— підтримка JavaScript повинна бути увімкнена в налаштуваннях браузера;

— для виконання пакетного аналізу необхідне середовище Node.js версії 18.x або новіше.

Мережеві вимоги:

— інтернет-з'єднання потрібне лише для початкового завантаження застосунку;

— після завантаження застосунок працює повністю автономно без підключення до мережі;

— рекомендована швидкість з'єднання для початкового завантаження: мінімум 1 Мбіт/с.

3.4 Вимоги до складу і параметрів технічних засобів

Склад технічних засобів:

— веб-сервер для хостингу програми;

— клієнтський комп'ютер або мобільний пристрій;

— система забезпечення безпеки даних.

Технічні характеристики веб-сервера:

— процесор: 1 ГГц, 1 ядро;

— оперативна пам'ять: 512 МБ;

— дисковий простір: 100 МБ;

— ОС: Linux/Windows з Node.js 14.0+;

— веб-сервер: Nginx 1.18+ або Apache 2.4+.

Технічні характеристики клієнтського обладнання:

— веб-сервер: Nginx 1.18+ або Apache 2.4+.

— процесор: двоядерний 1.5 ГГц;

— оперативна пам'ять: 2 ГБ;

— екран: від 1024×768 (ПК) або 360×640 (мобільні);

— інтернет: від 1 Мбіт/с;

— браузер: Chrome 90+, Firefox 88+, Edge 90+, Safari 14+;

— мобільні ОС: IOS 14+ або Android 8.0+.

3.5 Вимоги до інформаційної і програмної сумісності

Інформаційні структури:

— структуроване збереження показників у форматі JSON з можливістю експорту у CSV;

— інтерактивний веб-інтерфейс з відображенням результатів аналізу в режимі реального часу;

— порівняльна візуалізація показників складності ES5 та ES6 коду.

Технологічний стек:

Фронтенд: Vue.js, Pug, Bootstrap, JavaScript (ES6+)

Бекенд: Node.js, Express.js

Аналіз коду: Babel Parser для побудови AST та обчислення метрик

Інструменти: Vite (збірка), Node.js Test Runner (тестування)

3.6 Вимоги до маркування і упаковки

Приклад маркування програми (назва програми, розробник, контакти, рік розробки) представлений на рис. 4.1

Code Complexity Lab

Розробник: Свиридов Михайло

УДУНТ, кафедра КІТ

2026

Рисунок 4.1 – Приклад маркування програми

Упаковка програмного продукту, включаючи документацію, повинна бути захищена від механічних та кліматичних пошкоджень, наприклад, пластикова коробка

для CD дисків.

3.7 Вимоги до транспортування і зберігання

Транспортування повинно забезпечити збереження програмного продукту, його цілісність та запобігти несанкціонованому доступу. Веб-додаток розповсюджується переважно через хмарне розміщення на веб-сервері з доступом через HTTPS-протокол. Альтернативними формами є архів вихідного коду на USB флеш-накопичувачах з відповідною упаковкою, що захищає від механічних ушкоджень та атмосферного впливу, а також електронний архів у форматі ZIP з контрольною сумою для перевірки цілісності.

4 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

До складу програмної документації мають входити:

— текст програми;

— керівництво користувача.

Вся документація до програмного додатку повинна задовольняти вимогам до програмної документації [1].

5 СТАДІЇ І ЕТАПИ РОЗРОБКИ

В таблиці 6.1 приведені стадії та етапи розробки.

Таблиця 6.1 – Стадії та етапи розробки

Стадія	Зміст робіт	Строки виконання
Технічне завдання	Узгодження і затвердження технічного завдання	03.10.2025-22.10.2025
Робочий проєкт	Програмування та відлагодження програми	23.10.2025-08.12.2025
	Тестування програми	09.12.2025-14.12.2025
	Розробка, узгодження, затвердження програмної документації	15.12.2025-07.01.2025

6 ПОРЯДОК КОНТРОЛЮ І ПРИЙМАННЯ

Контроль за виконанням роботи здійснює керівник дипломного проекту: Шаравара В.В.

7 БІБЛІОГРАФІЧНИЙ СПИСОК

1. Івченко, Ю.М. Основи стандартизації програмних систем [Текст]: методичні вказівки до дипломного проектування та лабораторних робіт / уклад.: Ю. М. Івченко, В. І. Шинкаренко, В. Г. Івченко; Дніпропетр. нац. ун-т залізн. трансп. ім. акад. В. Лазаряна. – Д.: Вид-во Дніпропетр. нац. ун-ту залізн. тра- нсп. ім. акад. В. Лазаряна, 2009. - 38 с.

ДОДАТОК Б

Текст програми

ЗАТВЕРДЖУЮ
Перший проректор Українського
державного
університету науки і технологій
Анатолій РАДКЕВИЧ

“CODE COMPLEXITY LAB”

Текст програми
ЛИСТ ЗАТВЕРДЖЕННЯ
44165850.1553 – 01 12 01

Завідувач кафедри КІТ
_____Вадим ГОРЯЧКІН
Керівник розробки
_____Віктор ШАРАВАРА
Виконавець
_____Михайло СВИРИДОВ
Нормоконтролер
_____Світлана ВОЛКОВА

2026

ЗАТВЕРДЖЕНО

44165850.1553 – 01 12 01

“CODE COMPLEXITY LAB”

Текст програми

Листів 60

2026

АНОТАЦІЯ

Документ 44165850.1553 – 01 12 01 «Code Complexity Lab. Текст програми» входить до складу програмної документації для інструментального засобу, розробленого з метою дослідження впливу стандарту ES6 мови JavaScript на показники підтримуваності коду. У цьому документі представлено веб-застосунок, побудований на основі Vite, Express.js та Babel parser, який реалізує механізми статичного аналізу коду для обчислення показників підтримуваності, когнітивної складності, цикломатичної складності, метрик Холстеда та кількості рядків коду. Програмне забезпечення включає інтерфейс користувача на основі Bootstrap та Pug для введення коду ES5 та ES6, модулі для синтаксичного аналізу та розрахунку показників, а також систему порівняльного аналізу результатів. Розроблений інструмент призначено для проведення експериментального дослідження з метою кількісної оцінки впливу сучасних можливостей JavaScript на підтримуваність програмного коду.

server.js

```
import express from 'express';
import path from 'path';
import { fileURLToPath } from 'url';

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

app.set('views', path.join(__dirname, 'src/pages'));
app.set('view engine', 'pug');

app.use(express.static(path.join(__dirname, 'public')));
app.use('/presentation_example',
express.static(path.join(__dirname, 'presentation_example')));

app.get('/', (_, res) => res.render('index', { title: 'Code Complexity
Lab' }));
app.get('/stats', (_, res) => res.render('stats', { title: 'Статистика
ES5 vs ES6' }));
app.get('/comparison', (_, res) => res.render('comparison', { title:
'ES5 vs ES6: Порівняння' }));

app.get('/docs', (_, res) => res.render('docs/index', { title:
'Документація' }));
app.get('/docs/mi', (_, res) => res.render('docs/mi', { title:
'Maintainability Index' }));
app.get('/docs/cognitive', (_, res) => res.render('docs/cognitive', {
title: 'Cognitive Complexity' }));
app.get('/docs/cyclomatic', (_, res) => res.render('docs/cyclomatic',
{ title: 'Cyclomatic Complexity' }));

const PORT = process.env.PORT ?? 3000;

app.listen(PORT, () => {
      console.log(`🚀 Code Complexity Lab слухає на
http://localhost:${PORT}`);
});
```

vite.config.js

```
// vite.config.js
import { defineConfig } from 'vite';

export default defineConfig({
 root: 'public',
 publicDir: 'public',
 server: { open: true },
 build: {
  outDir: 'dist',
  emptyOutDir: true
 }
});
```

examples.js

```
function makeUserES5(options) {
 options = options || {};
 var name = options.name || 'Anonymous';
 var email = options.email || '';
 var roles = options.roles || [];
 return { name: name, email: email, roles: roles };
}

const makeUserES6 = ({ name = 'Anonymous', email = '', roles = []
} = {}) =>
 ({ name, email, roles });


function getUserES5(id, cb) {
 api.get('/user/' + id, function (err, user) {
  if (err) return cb(err);
  api.get('/posts?u=' + user.id, function (err2, posts) {
   if (err2) return cb(err2);
   api.get('/comments?u=' + user.id, function (err3, comments) {
    if (err3) return cb(err3);
```

```
      cb(null, { user: user, posts: posts, comments: comments });
    });
  });
});
}



const getUserES6 = (id) =>
  api.get(`/user/${id}`)
    .then(user =>
      Promise.all([
        Promise.resolve(user),
        api.get(`/posts?u=${user.id}`),
        api.get(`/comments?u=${user.id}`)
      ])
    )
    .then(([user, posts, comments]) => ({ user, posts, comments }));
```

package.json

on

```
{
  "name": "code-complexity-lab",
  "version": "0.2.0",
  "private": true,
  "type": "module",
  "scripts": {
    "clean": "rimraf public/**/*.html",
    "pug:build": "pug -P src/pages -o public --extension html",
    "pug:watch": "pug -w src/pages -o public --extension html",
    "serve": "vite --strictPort --port 5174",
    "dev": "run-p pug:watch serve",
    "build": "run-s clean pug:build vite:build",
    "vite:build": "vite build",
    "preview": "vite preview",
    "analyze-examples": "node scripts/analyze-examples.js",
    "stats": "npm run analyze-examples && npm start",
    "start": "npm run build && npm run preview",
    "test": "node --test tests/**/*.test.js",
    "test:watch": "node --test --watch tests/**/*.test.js",
    "test:coverage": "node --test --experimental-test-coverage tests/**/*.test.js"
  },
  "devDependencies": {
    "@babel/parser": "^7.25.0",
    "npm-run-all": "^4.1.5",
    "pug-cli": "^1.0.0-alpha6",
    "rimraf": "^6.0.1",
    "vite": "^5.4.0"
  },
  "dependencies": {
    "express": "^4.21.2",
    "pug": "^3.0.3"
  }
}
```

scripts/analyze-examples.js

```
// scripts/analyze-examples.js
import fs from 'fs/promises';
import path from 'path';
import { fileURLToPath } from 'url';

import { maintainabilityIndex } from '../public/js/utils/maintainability.js';
import { cognitiveComplexity } from '../public/js/utils/cognitive.js';
import { cyclomaticComplexity } from '../public/js/utils/cyclomatic.js';
import { countLines, stripAllComments } from '../public/js/utils/string.js';

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

const EXAMPLES_DIR = path.join(__dirname, '../examples');
const OUTPUT_PATH = path.join(__dirname, '../public/data/examples-stats.json');
const CATEGORIES = ['es5', 'es6'];

const countTotalLines = (source) => {
```

```
  if (!source) return 0;
  return String(source).split(/\r?\n/).length;
};

const getJsFiles = async (dir) => {
  const entries = await fs.readdir(dir, { withFileTypes: true });
  return entries
    .filter((entry) => entry.isFile() && entry.name.endsWith('.js'))
    .map((entry) => entry.name)
    .sort((a, b) => a.localeCompare(b));
};

const readFile = (filePath) => fs.readFile(filePath, 'utf8');

const analyzeFile = (source) => {
  const miResult = maintainabilityIndex(source);
  const cc = cognitiveComplexity(source);
  const cyclomatic = cyclomaticComplexity(source);
  const totalLines = countTotalLines(source);
  const codeLines = countLines(stripAllComments(source));
  const commentLines = Math.max(0, totalLines - codeLines);

  return {
    mi: miResult.mi,
    miRounded: Math.round(miResult.mi),
    cc,
    cyclomatic,
    halsteadVolume: miResult.halsteadVolume ?? miResult.V ?? 0,
    loc: miResult.loc,
    totalLines,
    codeLines,
    commentLines,
     commentPercentage: totalLines > 0 ? Number(((commentLines /
totalLines) * 100).toFixed(2)) : 0,
  };
};

const calculateStats = (values) => {
  const sorted = [...values].sort((a, b) => a - b);
  const count = sorted.length;

  if (!count) {
    return { mean: 0, median: 0, min: 0, max: 0, stdDev: 0 };
  }

  const sum = sorted.reduce((acc, value) => acc + value, 0);
  const mean = sum / count;
  const median =
    count % 2 === 0
      ? (sorted[count / 2 - 1] + sorted[count / 2]) / 2
      : sorted[Math.floor(count / 2)];

  const variance = sorted.reduce((acc, value) => acc + (value -
mean) ** 2, 0) / count;
  const stdDev = Math.sqrt(variance);

  return {
    mean: Number(mean.toFixed(2)),
    median: Number(median.toFixed(2)),
    min: sorted[0],
    max: sorted[count - 1],
    stdDev: Number(stdDev.toFixed(2)),
  };
};

const calculateImprovement = (es5Value, es6Value, higherIsBetter)
=> {
  if (es5Value === 0 && es6Value === 0) return 0;
  if (es5Value === 0) return Number((higherIsBetter ? es6Value :
-es6Value).toFixed(2));

  const diff = higherIsBetter
    ? ((es6Value - es5Value) / es5Value) * 100
    : ((es5Value - es6Value) / es5Value) * 100;

  return Number(diff.toFixed(2));
};

const analyzeCategory = async (category) => {
  const categoryDir = path.join(EXAMPLES_DIR, category);
  const files = await getJsFiles(categoryDir);

        console.log(`📁 ${category.toUpperCase()}: знайдено
${files.length} файлів`);
```

```
  const results = [];

  for (const fileName of files) {
    const filePath = path.join(categoryDir, fileName);
    const source = await readFile(filePath);
    const metrics = analyzeFile(source);

    console.log(`  ✓ ${fileName}`);
    results.push({ fileName, category, ...metrics });
  }

  return results;
};

const ensureOutputDir = () => fs.mkdir(path.dirname(OUTPUT_PATH), { recursive: true });

const analyzeAllExamples = async () => {
  console.log('🔍 Аналізуємо приклади...\n');

  const results = {
    timestamp: new Date().toISOString(),
    categories: {},
    statistics: {},
    improvements: {},
    pairwise: [],
  };

  for (const category of CATEGORIES) {
    results.categories[category] = await analyzeCategory(category);
  }

  const es5Results = results.categories.es5 ?? [];
  const es6Results = results.categories.es6 ?? [];

  if (!es5Results.length || !es6Results.length) {
    throw new Error('Потрібні обидві категорії es5 та es6 для порівняння');
  }

  const summary = (list, key) => calculateStats(list.map((item) => item[key]));

  results.statistics = {
    es5: {
      mi: summary(es5Results, 'mi'),
      cc: summary(es5Results, 'cc'),
      cyclomatic: summary(es5Results, 'cyclomatic'),
      loc: summary(es5Results, 'loc'),
      halsteadVolume: summary(es5Results, 'halsteadVolume'),
    },
    es6: {
      mi: summary(es6Results, 'mi'),
      cc: summary(es6Results, 'cc'),
      cyclomatic: summary(es6Results, 'cyclomatic'),
      loc: summary(es6Results, 'loc'),
      halsteadVolume: summary(es6Results, 'halsteadVolume'),
    },
  };

  results.improvements = {
    mi: calculateImprovement(results.statistics.es5.mi.mean, results.statistics.es6.mi.mean, true),
    cc: calculateImprovement(results.statistics.es5.cc.mean, results.statistics.es6.cc.mean, false),
    cyclomatic: calculateImprovement(
      results.statistics.es5.cyclomatic.mean,
      results.statistics.es6.cyclomatic.mean,
      false,
    ),
    loc: calculateImprovement(
      results.statistics.es5.loc.mean,
      results.statistics.es6.loc.mean,
      false,
    ),
  };

  const es6ByFile = new Map(es6Results.map((item) => [item.fileName, item]));

  results.pairwise = es5Results
    .map((es5File) => {
      const es6File = es6ByFile.get(es5File.fileName);
```

```
    if (!es6File) return null;

    return {
      fileName: es5File.fileName,
      es5: {
        mi: Math.round(es5File.mi),
        cc: es5File.cc,
        cyclomatic: es5File.cyclomatic,
        loc: es5File.loc,
      },
      es6: {
        mi: Math.round(es6File.mi),
        cc: es6File.cc,
        cyclomatic: es6File.cyclomatic,
        loc: es6File.loc,
      },
      improvements: {
        mi: calculateImprovement(es5File.mi, es6File.mi, true),
        cc: calculateImprovement(es5File.cc, es6File.cc, false),
        cyclomatic: calculateImprovement(es5File.cyclomatic, es6File.cyclomatic, false),
        loc: calculateImprovement(es5File.loc, es6File.loc, false),
      },
    };
  })
  .filter(Boolean);

 await ensureOutputDir();
 await fs.writeFile(OUTPUT_PATH, JSON.stringify(results, null, 2));

 console.log('\n✅ Аналіз завершено!');
 console.log(`📊 Результати збережено: ${OUTPUT_PATH}`);
 const formatHigherIsBetter = (value) => `${value > 0 ? '+' : ''}${value}%`;
 const formatLowerIsBetter = (value) => (value > 0 ? `-${value}%` : `${value}%`);

 console.log('\n📈 Покращення (ES6 vs ES5):');
 console.log(`  MI: ${formatHigherIsBetter(results.improvements.mi)}`);
 console.log(`  CC: ${formatLowerIsBetter(results.improvements.cc)}`);
 console.log(`  Cyclomatic: ${formatLowerIsBetter(results.improvements.cyclomatic)}`);
 console.log(`  LOC: ${formatLowerIsBetter(results.improvements.loc)}`);

 return results;
};

analyzeAllExamples().catch((error) => {
 console.error('❌ Помилка аналізу:', error);
 process.exitCode = 1;
});
```

public/js/main.js

```
// public/js/main.js
import { maintainabilityIndex } from './utils/maintainability.js';
import { cognitiveComplexity } from './utils/cognitive.js';
import { countLines, stripAllComments } from './utils/string.js';
import { renderMI, renderCC, setSample, resetMetricsView } from './render.js';

// ========================================
// КОНСТАНТИ
// ========================================

const HUNDRED_PERCENT = 100;
const EMPTY_STRING = '';
const KEY_ENTER = 'Enter';
const EVENT_CLICK = 'click';
const EVENT_KEYDOWN = 'keydown';
const EVENT_INPUT = 'input';
const EVENT_CHANGE = 'change';
const EVENT_DOM_READY = 'DOMContentLoaded';
const SELECTOR_TOOLTIP_TRIGGER = '[data-bs-toggle="tooltip"]';
const DATASET_TOOLTIP_BOUND = 'tooltipBound';
const TOOLTIP_BOUND_VALUE = 'true';
const CLASS_HIDDEN = 'd-none';
```

```
const AUTO_CLEAR_AFTER_ANALYSIS = true;
const DEBOUNCE_DELAY_MS = 350;
const LOADING_DELAY_MS = 350;
const COPY_RESET_DELAY_MS = 2000;
const HISTORY_STORAGE_KEY = 'code-complexity-history';
const HISTORY_MAX_ITEMS = 10;
const HISTORY_PREVIEW_LINES = 3;
const SESSION_PREFILL_KEY = 'ccl:prefill-code';

const STATUS_MESSAGES = {
  ready: 'Готово до аналізу',
  analyzing: 'Аналіз…',
    complete: '✓ Аналіз завершено. Вставте новий код для повторної перевірки.',
  empty: 'Будь ласка, спершу вставте JavaScript-код.',
  cleared: 'Поле очищено.',
  sample: 'Завантажено приклад.',
  fileLoaded: (name) => `Завантажено ${name}`,
  fileError: 'Не вдалося прочитати файл. Спробуйте інший.',
};

const STATUS_ICON_CLASSES = {
  info: 'bi bi-info-circle text-muted',
  success: 'bi bi-check-circle-fill text-success',
  warning: 'bi bi-exclamation-triangle-fill text-warning',
  error: 'bi bi-x-circle-fill text-danger',
};

const SELECTORS = {
  codeInput: '#code',
  analyzeButton: '#analyze',
  analyzeSpinner: '#analyze-spinner',
  analyzeText: '#analyze-text',
  sampleButton: '#sample',
  clearButton: '#clear',
  uploadButton: '#upload',
  fileInput: '#file-input',
  inputStatus: '#input-status',
  inputStatusIcon: '#input-status i',
  inputStatusText: '#input-status span',
  codePreview: '#code-preview',
  codeDisplay: '#code-display',
  copyCodeButton: '#copy-code',
  fullscreenButton: '#fullscreen-code',
  collapseButton: '#collapse-preview',
  codeLinesBadge: '#code-lines',
  miProgress: '#mi-progress',
  ccProgress: '#cc-progress',
  historySection: '#history-section',
  historyList: '#history-list',
  historyCount: '#history-count',
  clearHistoryButton: '#clear-history',
};

const SAMPLE_SNIPPET = `
// ES5-стиль:
function makeUserES5(options) {
  options = options || {};
  var name = options.name || 'Anonymous';
  var email = options.email || '';
  var roles = options.roles || [];
  return { name: name, email: email, roles: roles };
}

// ES6-стиль:
const makeUserES6 = ({ name = 'Anonymous', email = '', roles = [] } = {}) =>
  ({ name, email, roles });
`;

// ========================================
// СТАН
// ========================================

let analyzeDebounced = null;
let copyFeedbackTimer = null;
let previewVisible = false;
let skipAutoClearOnce = false;

// ========================================
// UTILITY ФУНКЦІЇ (маленькі чисті функції)
// ========================================

const debounce = (func, delay) => {
```

```
  let timeoutId;
  return (...args) => {
    window.clearTimeout(timeoutId);
    timeoutId = window.setTimeout(() => func(...args), delay);
  };
};

const delay = (ms) => new Promise((resolve) => window.setTimeout(resolve, ms));

const isAnalyzeShortcut = (event) =>
      event.key === KEY_ENTER && (!event.shiftKey || event.ctrlKey);

const computeCommentsPercentage = (source) => {
  const totalLines = countLines(source);
                    const uncommentedLines = countLines(stripAllComments(source));
  if (!totalLines) return 0;
       const commentLines = Math.max(0, totalLines - uncommentedLines);
  return (commentLines / totalLines) * HUNDRED_PERCENT;
};

const buildMaintainabilityPayload = (source) => {
  const maintainabilityMetrics = maintainabilityIndex(source);
  const commentsPct = computeCommentsPercentage(source);
  return { ...maintainabilityMetrics, commentsPct };
};

const buildMetricsSnapshot = (source) => ({
  maintainability: buildMaintainabilityPayload(source),
  cognitive: cognitiveComplexity(source),
});

// ========================================
// DOM СЕЛЕКТОРИ (централізовані)
// ========================================

const getElement = (key) => document.querySelector(SELECTORS[key]);

const getInputValue = (key) => {
  const element = getElement(key);
    return element && 'value' in element ? element.value : EMPTY_STRING;
};

const setInputValue = (key, value) => {
  const element = getElement(key);
  if (element && 'value' in element) element.value = value;
};

const bindEvent = (key, eventName, handler) => {
  const element = getElement(key);
  if (element) element.addEventListener(eventName, handler);
};

const queryAllElements = (selector) => Array.from(document.querySelectorAll(selector));

const toggleHidden = (element, hidden) => {
  if (element) element.classList.toggle(CLASS_HIDDEN, hidden);
};

const setDisabled = (keys, disabled) => {
  keys.forEach((key) => {
    const element = getElement(key);
    if (element) element.disabled = disabled;
  });
};

const escapeHtml = (value) =>
  String(value ?? '')
    .replace(/&/g, '&amp;')
    .replace(/</g, '&lt;')
    .replace(/>/g, '&gt;')
    .replace(/"/g, '&quot;')
    .replace(/'/g, '&#39;');

// ========================================
// ІСТОРІЯ КОДУ (LOCALSTORAGE + UI)
// ========================================
```

```
const generateId = () =>
        'xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx'.replace(/[xy]/g,
(char) => {
   const random = (Math.random() * 16) | 0;
   const value = char === 'x' ? random : (random & 0x3) | 0x8;
   return value.toString(16);
  });

const formatTimestamp = (timestamp) => {
  const date = new Date(timestamp);
  const now = new Date();
  const diffMs = now - date;
  const diffMinutes = Math.floor(diffMs / 60000);
  const diffHours = Math.floor(diffMs / 3600000);
  const diffDays = Math.floor(diffMs / 86400000);

  if (diffMinutes < 1) return 'щойно';
  if (diffMinutes < 60) return `${diffMinutes} хв тому`;
  if (diffHours < 24) return `${diffHours} год тому`;
  if (diffDays === 1) return 'вчора';
  if (diffDays < 7) return `${diffDays} дн. тому`;

  return date.toLocaleDateString('uk-UA', {
   day: 'numeric',
   month: 'short',
      year: date.getFullYear() !== now.getFullYear() ? 'numeric' :
undefined,
  });
};

const createCodePreview = (code, lines =
HISTORY_PREVIEW_LINES) => {
  const codeLines = String(code ?? '').split('\n');
  const preview = codeLines.slice(0, lines).join('\n');
  return codeLines.length > lines ? `${preview}\n...` : preview;
};

const getMiStatus = (mi) => {
  if (mi >= 10) return 'high';
  if (mi >= 20) return 'medium';
  return 'low';
};

const getCcStatus = (cc) => {
  if (cc <= 10) return 'good';
  if (cc <= 20) return 'ok';
  return 'bad';
};

const getMiLabel = (status) => ({
  high: 'Високий',
  medium: 'Середній',
  low: 'Низький',
}[status] || status);

const getCcLabel = (status) => ({
  good: 'Добре',
  ok: 'Прийнятно',
  bad: 'Погано',
}[status] || status);

const getHistory = () => {
  try {
   const raw = localStorage.getItem(HISTORY_STORAGE_KEY);
   return raw ? JSON.parse(raw) : [];
  } catch (error) {
   console.error('Помилка читання історії:', error);
   return [];
  }
};

const saveHistory = (history) => {
  try {
                localStorage.setItem(HISTORY_STORAGE_KEY,
JSON.stringify(history));
   return true;
  } catch (error) {
   console.error('Помилка збереження історії:', error);
   return false;
  }
};

const readSessionPrefill = () => {
  try {
```

```
    const payload = sessionStorage.getItem(SESSION_PREFILL_KEY);
    if (!payload) return '';
    sessionStorage.removeItem(SESSION_PREFILL_KEY);
    return payload;
  } catch (error) {
    console.warn('Не вдалося прочитати префіл із сесії:', error);
    return '';
  }
};

const applySessionPrefill = () => {
  const buffered = readSessionPrefill();
  if (!buffered) return;
  setCodeInput(buffered);
  skipAutoClearOnce = true;
  updateInputStatus('Завантажено приклад ES5/ES6. Аналізуємо…', 'info');
  analyzeSource();
};

const addToHistory = (code, mi, cc) => {
  const normalizedCode = String(code ?? '');
  if (!normalizedCode.trim()) return [];

  const existing = getHistory().filter((entry) => entry.code !== normalizedCode);

  const entry = {
    id: generateId(),
    timestamp: Date.now(),
    code: normalizedCode,
    preview: createCodePreview(normalizedCode),
    metrics: {
      mi: Math.round(Number(mi) || 0),
      cc: Math.round(Number(cc) || 0),
      miStatus: getMiStatus(Number(mi) || 0),
      ccStatus: getCcStatus(Number(cc) || 0),
    },
  };

  existing.unshift(entry);
  const trimmed = existing.slice(0, HISTORY_MAX_ITEMS);
  saveHistory(trimmed);
  return trimmed;
};

const clearHistoryStorage = () => {
  try {
    localStorage.removeItem(HISTORY_STORAGE_KEY);
    return true;
  } catch (error) {
    console.error('Помилка очищення історії:', error);
    return false;
  }
};

const getHistoryEmptyTemplate = () => `
  <div class="text-center text-muted py-4" id="history-empty">
    <i class="bi bi-inbox fs-2 d-block mb-2"></i>
      <p class="mb-0">Історія порожня. Проаналізуйте код щоб почати.</p>
  </div>
`;

const formatHistoryPreview = (preview) =>
  escapeHtml(preview).replace(/\n/g, '<br>');

const createHistoryItemHTML = (entry) => {
  const { id, timestamp, preview, metrics } = entry;
  return `
    <article class="history-item" data-history-id="${id}">
      <div class="history-item-header">
        <div class="history-item-timestamp">
          <i class="bi bi-clock"></i>
          <span>${formatTimestamp(timestamp)}</span>
        </div>
      </div>
                                                                    <div class="history-item-preview">${formatHistoryPreview(preview)}</div>
      <div class="history-item-metrics">
        <div class="history-metric-badge mi-${metrics.miStatus}">
          <span class="history-metric-label">MI</span>
```

```
        <span class="history-metric-value">${metrics.mi}</span>
                                                <span       class="small
opacity-75">${getMiLabel(metrics.miStatus)}</span>
       </div>
       <div class="history-metric-badge cc-${metrics.ccStatus}">
        <span class="history-metric-label">CC</span>
        <span class="history-metric-value">${metrics.cc}</span>
                                                <span       class="small
opacity-75">${getCcLabel(metrics.ccStatus)}</span>
       </div>
      </div>
    </article>
  `;
};

const renderHistory = () => {
  const history = getHistory();
  const historySection = getElement('historySection');
  const historyList = getElement('historyList');
  const historyCount = getElement('historyCount');

  if (!historySection || !historyList) return;

  historySection.classList.remove('d-none');
  if (historyCount) historyCount.textContent = history.length;

  if (!history.length) {
   historyList.innerHTML = getHistoryEmptyTemplate();
   return;
  }

                                historyList.innerHTML                    =
history.map(createHistoryItemHTML).join('');
  historyList.querySelectorAll('.history-item').forEach((item) => {
                               item.addEventListener(EVENT_CLICK,
handleHistoryItemClick);
  });
};

const handleHistoryItemClick = (event) => {
  const card = event.currentTarget;
  const historyId = card?.dataset?.historyId;
  if (!historyId) return;

  const history = getHistory();
  const entry = history.find((item) => item.id === historyId);
  if (!entry) return;

  skipAutoClearOnce = true;
  setCodeInput(entry.code);
  updateInputStatus('Завантажено з історії. Аналізуємо…', 'info');
  focusCodeInput();
  analyzeSource();
};

const handleClearHistory = () => {
  if (!window.confirm('Видалити всю історію аналізів?')) return;
  clearHistoryStorage();
  renderHistory();
};


// ========================================
// РЕНДЕРИНГ (побічні ефекти ізольовані)
// ========================================

const updateInputStatus = (message, type = 'info') => {
  const wrapper = getElement('inputStatus');
                                const                icon                =
document.querySelector(SELECTORS.inputStatusIcon);
                                const                text                =
document.querySelector(SELECTORS.inputStatusText);
  if (wrapper) wrapper.dataset.statusType = type;
   if (icon) icon.className = STATUS_ICON_CLASSES[type] ??
STATUS_ICON_CLASSES.info;
  if (text) text.textContent = message;
};

const focusCodeInput = () => {
  const input = getElement('codeInput');
  if (input) input.focus();
};

const setCodeInput = (value) => setInputValue('codeInput', value);
```

```
const clearCodeInput = () => setCodeInput(EMPTY_STRING);

const highlightCode = (element) => {
  const highlighter = window.Prism?.highlightElement;
  if (typeof highlighter === 'function') highlighter(element);
};

const formatLinesCount = (count) => {
  const mod10 = count % 10;
  const mod100 = count % 100;
  if (mod10 === 1 && mod100 !== 11) return `${count} рядок`;
   if (mod10 >= 2 && mod10 <= 4 && (mod100 < 10 || mod100 >=
20)) return `${count} рядки`;
  return `${count} рядків`;
};

const updateCodeLinesBadge = (source) => {
  const badge = getElement('codeLinesBadge');
  if (!badge) return;
  const lines = source ? source.split('\n').length : 0;
  badge.textContent = formatLinesCount(lines);
};

const updateTooltip = (element, text) => {
  if (!element) return;
  element.setAttribute('data-bs-original-title', text);
  element.setAttribute('title', text);
  const TooltipClass = window.bootstrap?.Tooltip;
  if (TooltipClass) {
    const instance = TooltipClass.getInstance(element);
    if (instance) {
      instance.setContent({ '.tooltip-inner': text });
    }
  }
};

const resetCopyButtonAppearance = () => {
  const button = getElement('copyCodeButton');
  const icon = button?.querySelector('i');
  window.clearTimeout(copyFeedbackTimer);
  if (!button || !icon) return;
  icon.className = 'bi bi-clipboard';
  button.classList.add('btn-outline-light');
  button.classList.remove('btn-success');
  updateTooltip(button, 'Копіювати код');
};

const exitFullscreenPreview = () => {
  const preview = getElement('codePreview');
  const button = getElement('fullscreenButton');
  const icon = button?.querySelector('i');
  if (!preview) return;
  preview.classList.remove('fullscreen');
  document.body.style.overflow = '';
  if (icon) icon.className = 'bi bi-arrows-fullscreen';
  updateTooltip(button, 'Повноекранний режим');
};

const displayCodePreview = (source) => {
  const preview = getElement('codePreview');
  const display = getElement('codeDisplay');
  if (!preview || !display) return;
  const hasContent = Boolean(source && source.trim());
  if (!hasContent) {
    display.textContent = EMPTY_STRING;
    exitFullscreenPreview();
    preview.classList.add('d-none');
    updateCodeLinesBadge('');
    resetCopyButtonAppearance();
    const body = preview.querySelector('.code-preview-body');
    if (body) body.scrollTop = 0;
    previewVisible = false;
    return;
  }
  display.textContent = source;
  preview.classList.remove('d-none');
  updateCodeLinesBadge(source);
  resetCopyButtonAppearance();
  highlightCode(display);
  const body = preview.querySelector('.code-preview-body');
  if (body) body.scrollTop = 0;
  previewVisible = true;
  if (!preview.classList.contains('fullscreen')) {
```

```
    setTimeout(() => {
      preview.scrollIntoView({ behavior: 'smooth', block: 'start' });
    }, 100);
  }
};

const toggleFullscreen = (event) => {
  if (event) event.preventDefault();
  const preview = getElement('codePreview');
  const button = getElement('fullscreenButton');
  const icon = button?.querySelector('i');
  if (!preview || !button || !icon || preview.classList.contains('d-none')) return;
  const entering = !preview.classList.contains('fullscreen');
  if (entering) {
    preview.classList.add('fullscreen');
    document.body.style.overflow = 'hidden';
    icon.className = 'bi bi-fullscreen-exit';
    updateTooltip(button, 'Вийти з повного екрану');
  } else {
    preview.classList.remove('fullscreen');
    document.body.style.overflow = '';
    icon.className = 'bi bi-arrows-fullscreen';
    updateTooltip(button, 'Повноекранний режим');
    setTimeout(() => {
      preview.scrollIntoView({ behavior: 'smooth', block: 'start' });
    }, 100);
  }
};

const collapsePreview = () => {
  const preview = getElement('codePreview');
  if (!preview) return;
  exitFullscreenPreview();
  preview.classList.add('d-none');
  previewVisible = false;
  resetCopyButtonAppearance();
  const body = preview.querySelector('.code-preview-body');
  if (body) body.scrollTop = 0;
  const input = getElement('codeInput');
  if (input) input.scrollIntoView({ behavior: 'smooth', block: 'center' });
};

const handleCopyCodeClick = async () => {
  const button = getElement('copyCodeButton');
  const display = getElement('codeDisplay');
  const icon = button?.querySelector('i');
  const code = display?.textContent || '';
  if (!button || !icon || !code.trim()) return;
  try {
    if (!navigator.clipboard?.writeText) throw new Error('Clipboard API недоступний');
    await navigator.clipboard.writeText(code);
    icon.className = 'bi bi-check-lg';
    button.classList.remove('btn-outline-light');
    button.classList.add('btn-success');
    updateTooltip(button, 'Скопійовано!');
    window.clearTimeout(copyFeedbackTimer);
    copyFeedbackTimer = window.setTimeout(() => resetCopyButtonAppearance(), COPY_RESET_DELAY_MS);
  } catch (error) {
    console.error('Помилка копіювання:', error);
  }
};

const handleKeyPress = (event) => {
  if (event.key !== 'Escape') return;
  const preview = getElement('codePreview');
  if (preview?.classList.contains('fullscreen')) toggleFullscreen();
};

const setAnalyzingState = (isAnalyzing) => {
  const spinner = getElement('analyzeSpinner');
  const label = getElement('analyzeText');
  toggleHidden(spinner, !isAnalyzing);
  if (label) label.textContent = isAnalyzing ? 'Аналіз…' : 'Аналізувати код';
  setDisabled(['analyzeButton', 'sampleButton', 'clearButton', 'uploadButton'], isAnalyzing);
};

const updateMiProgress = (value) => {
  const progress = getElement('miProgress');
```

```
  const safeValue = Math.max(0, Math.min(100, Number(value) || 0));
  if (progress) {
    progress.style.width = `${safeValue}%`;
    progress.setAttribute('aria-valuenow', String(safeValue));
  }
};

const updateCcProgress = (value) => {
  const progress = getElement('ccProgress');
  const numeric = Math.max(0, Number(value) || 0);
  const percent = Math.min(100, (numeric / 30) * 100);
  if (progress) {
    progress.style.width = `${percent}%`;
                                progress.setAttribute('aria-valuenow', String(Math.round(percent)));
  }
};

// ========================================
// ОБЧИСЛЕННЯ МЕТРИК
// ========================================

const readSourceCode = () => getInputValue('codeInput') || EMPTY_STRING;

const analyzeSource = async () => {
  const source = readSourceCode();
  if (!source.trim()) {
    updateInputStatus(STATUS_MESSAGES.empty, 'warning');
    focusCodeInput();
    return;
  }

  updateInputStatus(STATUS_MESSAGES.analyzing, 'info');
  setAnalyzingState(true);
  await delay(LOADING_DELAY_MS);

  const metrics = buildMetricsSnapshot(source);
  renderCC(metrics.cognitive);
  renderMI(metrics.maintainability);
  updateMiProgress(metrics.maintainability.mi ?? 0);
  updateCcProgress(metrics.cognitive);
  displayCodePreview(source);
        addToHistory(source, metrics.maintainability.mi ?? 0, metrics.cognitive ?? 0);
  renderHistory();

   const shouldClearInput = AUTO_CLEAR_AFTER_ANALYSIS && !skipAutoClearOnce;
  if (shouldClearInput) {
    clearCodeInput();
  }
  skipAutoClearOnce = false;

  updateInputStatus(STATUS_MESSAGES.complete, 'success');
  setAnalyzingState(false);
  focusCodeInput();
};

// ========================================
// ОБРОБНИКИ ПОДІЙ
// ========================================

const handleAnalyzeClick = () => analyzeSource();

const handleCodeKeydown = (event) => {
  if (!isAnalyzeShortcut(event)) return;
  event.preventDefault();
  analyzeSource();
};

const handleCodeInput = () => {
  const value = readSourceCode();
  if (!value.trim()) {
    updateInputStatus(STATUS_MESSAGES.ready, 'info');
    return;
  }
  updateInputStatus(STATUS_MESSAGES.ready, 'info');
  if (analyzeDebounced) analyzeDebounced();
};

const handleClearClick = () => {
  clearCodeInput();
```

```
  resetMetricsView();
  updateMiProgress(0);
  updateCcProgress(0);
  displayCodePreview('');
  updateInputStatus(STATUS_MESSAGES.cleared, 'info');
  focusCodeInput();
};

const handleSampleClick = () => {
  setSample(SAMPLE_SNIPPET);
  updateInputStatus(STATUS_MESSAGES.sample, 'success');
  analyzeSource();
};

const handleUploadClick = () => {
  const input = getElement('fileInput');
  if (input) input.click();
};

const handleFileSelection = async (event) => {
  const file = event.target.files?.[0];
  if (!file) return;
  try {
    const text = await file.text();
    setCodeInput(text);
    updateInputStatus(STATUS_MESSAGES.fileLoaded(file.name),
'success');
    analyzeSource();
  } catch (error) {
    console.error('Помилка читання файлу:', error);
    updateInputStatus(STATUS_MESSAGES.fileError, 'error');
  } finally {
    event.target.value = '';
  }
};

// ========================================
// ІНІЦІАЛІЗАЦІЯ UI
// ========================================

const initTooltips = () => {
  const Tooltip = window.bootstrap?.Tooltip;
  if (!Tooltip) return;

queryAllElements(SELECTOR_TOOLTIP_TRIGGER).forEach((el
ement) => {
    if (!element.dataset[DATASET_TOOLTIP_BOUND]) {
      new Tooltip(element);
                element.dataset[DATASET_TOOLTIP_BOUND]   =
TOOLTIP_BOUND_VALUE;
    }
  });
};

const initPopovers = () => {
  const PopoverClass = window.bootstrap?.Popover;
  if (!PopoverClass) return;

document.querySelectorAll('[data-bs-toggle="popover"]').forEach((
element) => {
    if (!element.dataset.popoverBound) {
      new PopoverClass(element);
      element.dataset.popoverBound = 'true';
    }
  });
};

const bindControls = () => {
                bindEvent('analyzeButton',          EVENT_CLICK,
handleAnalyzeClick);
                bindEvent('codeInput',          EVENT_KEYDOWN,
handleCodeKeydown);
  bindEvent('codeInput', EVENT_INPUT, handleCodeInput);
  bindEvent('clearButton', EVENT_CLICK, handleClearClick);
  bindEvent('sampleButton', EVENT_CLICK, handleSampleClick);
                bindEvent('copyCodeButton',          EVENT_CLICK,
handleCopyCodeClick);
  bindEvent('fullscreenButton', EVENT_CLICK, toggleFullscreen);
  bindEvent('collapseButton', EVENT_CLICK, collapsePreview);
  bindEvent('uploadButton', EVENT_CLICK, handleUploadClick);
  bindEvent('fileInput', EVENT_CHANGE, handleFileSelection);
                bindEvent('clearHistoryButton',          EVENT_CLICK,
handleClearHistory);
};
```

```
const initializeApp = () => {
        analyzeDebounced = debounce(analyzeSource, DEBOUNCE_DELAY_MS);
  bindControls();
  renderHistory();
  updateInputStatus(STATUS_MESSAGES.ready, 'info');
  updateCodeLinesBadge('');
  resetCopyButtonAppearance();
  focusCodeInput();
  applySessionPrefill();
  initTooltips();
  initPopovers();
  document.addEventListener('keydown', handleKeyPress);
};

document.addEventListener(EVENT_DOM_READY, initializeApp);
```

public/js/render.js

```
// public/js/render.js

// ========================================
// КОНСТАНТИ
// ========================================

const PERCENT_MIN = 0;
const PERCENT_MAX = 100;

const MI_THRESHOLD_LOW = 9;
const MI_THRESHOLD_MEDIUM = 19;
const MI_RECOMMENDATION_THRESHOLD = 20;
const MI_CRITICAL_THRESHOLD = 10;

const CC_THRESHOLD_GOOD = 10;
const CC_THRESHOLD_OK = 20;
const CC_WARNING_THRESHOLD = 20;
const CC_SEVERE_THRESHOLD = 30;

const MI_BAND_LOW = 'low';
const MI_BAND_MEDIUM = 'med';
const MI_BAND_HIGH = 'high';

const CC_TIER_GOOD = 'good';
const CC_TIER_OK = 'ok';
const CC_TIER_BAD = 'bad';

const CSS_MI_LOW = 'mi--low';
const CSS_MI_MEDIUM = 'mi--med';
const CSS_MI_HIGH = 'mi--high';
const CSS_CC_GOOD = 'cc--good';
const CSS_CC_OK = 'cc--ok';
const CSS_CC_BAD = 'cc--bad';

const MESSAGE_MI_CRITICAL =
    'MI < 10: розбийте великі функції, спростіть логіку та приберіть дублювання.';
const MESSAGE_MI_WARNING =
   'MI < 20: винесіть допоміжні блоки у утиліти та додайте короткі пояснювальні коментарі.';
const MESSAGE_CC_WARNING =
    'CC > 20: скоротіть довгі умови та винесіть ланцюжки перевірок у окремі предикати.';
const MESSAGE_CC_SEVERE =
     'CC ≥ 30: зменште рівні вкладеності, використайте guard-услови й розбийте логіку на функції.';

const NEWLINE = '\n';
const CHIP_ANIMATION_CLASS = 'updating';
const CHIP_ANIMATION_DELAY_MS = 150;
const TOAST_DELAY_MS = 5000;
const TOAST_MI_ID = 'toast-mi';
const TOAST_CC_ID = 'toast-cc';
const CLASS_HIDDEN = 'd-none';

const MI_STATUS_META = {
  [MI_BAND_LOW]: {
    badgeText: 'Низький',
    badgeClass: 'bg-danger text-white',
    description: 'Потребує уваги',
```

```
  },
  [MI_BAND_MEDIUM]: {
   badgeText: 'Середній',
   badgeClass: 'bg-warning text-dark',
   description: 'Є куди покращувати',
  },
  [MI_BAND_HIGH]: {
   badgeText: 'Високий',
   badgeClass: 'bg-success text-white',
   description: 'Хороша підтримуваність',
  },
};

const CC_STATUS_META = {
  [CC_TIER_GOOD]: {
   badgeText: 'Добрий',
   badgeClass: 'bg-success text-white',
   description: 'Код легко читати',
  },
  [CC_TIER_OK]: {
   badgeText: 'Попередження',
   badgeClass: 'bg-warning text-dark',
   description: 'Стежте за складністю',
  },
  [CC_TIER_BAD]: {
   badgeText: 'Високий',
   badgeClass: 'bg-danger text-white',
   description: 'Рекомендовано рефакторинг',
  },
};

const TIP_MESSAGES = {
  miCritical: MESSAGE_MI_CRITICAL,
  miWarning: MESSAGE_MI_WARNING,
  ccWarning: MESSAGE_CC_WARNING,
  ccSevere: MESSAGE_CC_SEVERE,
};

const SELECTORS = {
  miCard: '.metric-card.mi-card',
  miChip: '#mi-chip',
  miStatusLabel: '#mi-human-label',
  miStatusBadge: '#mi-status-icon',
  miVolume: '#mi-vol',
  miCyclomatic: '#mi-cyc',
  miLoc: '#mi-loc',
  miComments: '#mi-comments',
  ccCard: '.metric-card.cc-card',
  ccChip: '#cc-chip',
  ccStatusLabel: '#cc-human-label',
  ccStatusBadge: '#cc-status-icon',
  tipsSection: '#tips-section',
  tipsList: '#tips-list',
  codeInput: '#code',
};

const TOAST_TEXT_IDS = {
  [TOAST_MI_ID]: 'toast-mi-text',
  [TOAST_CC_ID]: 'toast-cc-text',
};

const MI_SKIN_CLASS_MAP = {
  [MI_BAND_LOW]: CSS_MI_LOW,
  [MI_BAND_MEDIUM]: CSS_MI_MEDIUM,
  [MI_BAND_HIGH]: CSS_MI_HIGH,
};

const CC_SKIN_CLASS_MAP = {
  [CC_TIER_GOOD]: CSS_CC_GOOD,
  [CC_TIER_OK]: CSS_CC_OK,
  [CC_TIER_BAD]: CSS_CC_BAD,
};

const MI_SKIN_CLASSES = Object.values(MI_SKIN_CLASS_MAP);
const CC_SKIN_CLASSES = Object.values(CC_SKIN_CLASS_MAP);

const BADGE_DEFAULT_CLASS = 'badge rounded-pill px-3 bg-light text-dark';
const MI_DEFAULT_STATUS_TEXT = 'Статус: —';
const MI_DEFAULT_BADGE_TEXT = 'Статус';
const CC_DEFAULT_STATUS_TEXT = 'Рівень: —';
const CC_DEFAULT_BADGE_TEXT = 'Рівень';
```

```
const DEFAULT_METRIC_VALUE = '0';
const DEFAULT_COMMENTS_VALUE = '0%';

// ========================================
// СТАН
// ========================================

let toastState = { mi: '', cc: '' };
let metricState = { mi: null, cc: null };

// ========================================
// UTILITY ФУНКЦІЇ (маленькі чисті функції)
// ========================================

const ensureNumber = (value, fallback = 0) => {
  const numeric = Number(value);
  return Number.isFinite(numeric) ? numeric : fallback;
};

const clampValue = (value, min = PERCENT_MIN, max = PERCENT_MAX) =>
  Math.max(min, Math.min(max, value));

const roundToInt = (value) => Math.round(ensureNumber(value));

const toFixedString = (value, digits = 2) => ensureNumber(value).toFixed(digits);

const toPercentString = (value) => `${clampValue(roundToInt(value))}%`;

const preferNonZero = (primary, fallback) => {
  const primaryNumber = ensureNumber(primary, NaN);
  if (Number.isFinite(primaryNumber) && primaryNumber !== 0) {
    return primaryNumber;
  }
  return ensureNumber(fallback);
};

const pickCyclomaticValue = (value, fallback) => {
  const primary = ensureNumber(value, NaN);
  return Number.isFinite(primary) ? primary : ensureNumber(fallback);
};

const toSafeString = (value) => {
  const numeric = ensureNumber(value, NaN);
  return Number.isFinite(numeric) ? String(numeric) : '0';
};

const formatMiChipValue = (value) => String(value);

const getMiBandCode = (miValue) => {
  if (miValue <= MI_THRESHOLD_LOW) return MI_BAND_LOW;
  if (miValue <= MI_THRESHOLD_MEDIUM) return MI_BAND_MEDIUM;
  return MI_BAND_HIGH;
};

const getCcTierCode = (ccValue) => {
  if (ccValue <= CC_THRESHOLD_GOOD) return CC_TIER_GOOD;
  if (ccValue <= CC_THRESHOLD_OK) return CC_TIER_OK;
  return CC_TIER_BAD;
};

// ========================================
// DOM СЕЛЕКТОРИ (централізовані)
// ========================================

const getElement = (key) => document.querySelector(SELECTORS[key]);

const getElementById = (id) => document.getElementById(id);

const setTextContent = (key, value) => {
  const element = getElement(key);
  if (element) element.textContent = value;
};

const setInputValue = (key, value) => {
  const element = getElement(key);
```

```
  if (element) element.value = value;
};

const focusElement = (key) => {
  const element = getElement(key);
  if (element) element.focus();
};

const updateClassSet = (element, classesToRemove, classToAdd)
=> {
  if (!element) return;
                          classesToRemove.forEach((cls)          =>
element.classList.remove(cls));
  if (classToAdd) element.classList.add(classToAdd);
};

const toggleHidden = (element, shouldHide) => {
      if (element) element.classList.toggle(CLASS_HIDDEN,
shouldHide);
};

// ========================================
// ОБЧИСЛЕННЯ МЕТРИК
// ========================================

const extractMiMetrics = (result) => {
  const source = result && typeof result === 'object' ? result : {};
  const gValue = ensureNumber(source.G);
  return {
    mi: ensureNumber(source.mi),
    loc: ensureNumber(source.loc),
    volume: preferNonZero(source.halsteadVolume, source.V),
    cyclomatic: pickCyclomaticValue(source.cyclomatic, gValue),
    commentsPct: ensureNumber(source.commentsPct),
  };
};

const buildMiDisplayModel = (result) => {
  const metrics = extractMiMetrics(result);
  const miValue = clampValue(roundToInt(metrics.mi));
  const bandCode = getMiBandCode(miValue);
  const statusMeta = MI_STATUS_META[bandCode];
  return {
    miValue,
    bandCode,
    chipText: formatMiChipValue(miValue),
    statusBadgeText: statusMeta.badgeText,
    statusBadgeClass: statusMeta.badgeClass,
    statusDescription: statusMeta.description,
    volumeText: toFixedString(metrics.volume),
    cyclomaticText: toSafeString(metrics.cyclomatic),
    locText: toSafeString(metrics.loc),
    commentsText: toPercentString(metrics.commentsPct),
  };
};

const buildCcDisplayModel = (value) => {
  const ccValue = Math.max(0, roundToInt(value));
  const tierCode = getCcTierCode(ccValue);
  const statusMeta = CC_STATUS_META[tierCode];
  return {
    ccValue,
    tierCode,
    chipText: String(ccValue),
    statusBadgeText: statusMeta.badgeText,
    statusBadgeClass: statusMeta.badgeClass,
    statusDescription: statusMeta.description,
  };
};

const determineMiToastMessage = (miValue) => {
  if (!Number.isFinite(miValue)) return '';
      if (miValue <= MI_CRITICAL_THRESHOLD) return
MESSAGE_MI_CRITICAL;
   if (miValue < MI_RECOMMENDATION_THRESHOLD) return
MESSAGE_MI_WARNING;
  return '';
};

const determineCcToastMessage = (ccValue) => {
  if (!Number.isFinite(ccValue)) return '';
       if (ccValue >= CC_SEVERE_THRESHOLD) return
MESSAGE_CC_SEVERE;
      if (ccValue > CC_WARNING_THRESHOLD) return
```

```
MESSAGE_CC_WARNING;
  return '';
};

const collectTips = ({ mi, cc }) => {
  const tips = [];
  if (Number.isFinite(mi)) {
                  if (mi <= MI_CRITICAL_THRESHOLD)
tips.push(TIP_MESSAGES.miCritical);
        else if (mi < MI_RECOMMENDATION_THRESHOLD)
tips.push(TIP_MESSAGES.miWarning);
  }
  if (Number.isFinite(cc)) {
                   if (cc >= CC_SEVERE_THRESHOLD)
tips.push(TIP_MESSAGES.ccSevere);
               else if (cc > CC_WARNING_THRESHOLD)
tips.push(TIP_MESSAGES.ccWarning);
  }
  return [...new Set(tips)];
};

// ========================================
// РЕНДЕРИНГ (побічні ефекти ізольовані)
// ========================================

const animateChipText = (key, text) => {
  const element = getElement(key);
  if (!element) return;
  element.classList.add(CHIP_ANIMATION_CLASS);
  window.setTimeout(() => {
    element.textContent = text;
    element.classList.remove(CHIP_ANIMATION_CLASS);
  }, CHIP_ANIMATION_DELAY_MS);
};

const updateMiChip = (model) => animateChipText('miChip',
model.chipText);

const updateMiStatus = (model) => {
  const badge = getElement('miStatusBadge');
  if (badge) {
    badge.textContent = model.statusBadgeText;
                  badge.className = `badge rounded-pill px-3
${model.statusBadgeClass}`;
  }
  const label = getElement('miStatusLabel');
              if (label) label.textContent = `Статус:
${model.statusDescription}`;
};

const updateMiMetrics = (model) => {
  setTextContent('miVolume', model.volumeText);
  setTextContent('miCyclomatic', model.cyclomaticText);
  setTextContent('miLoc', model.locText);
  setTextContent('miComments', model.commentsText);
};

const updateMiSkin = (model) => {
  const card = getElement('miCard');
                  updateClassSet(card, MI_SKIN_CLASSES,
MI_SKIN_CLASS_MAP[model.bandCode]);
};

const updateCcChip = (model) => animateChipText('ccChip',
model.chipText);

const updateCcStatus = (model) => {
  const badge = getElement('ccStatusBadge');
  if (badge) {
    badge.textContent = model.statusBadgeText;
                  badge.className = `badge rounded-pill px-3
${model.statusBadgeClass}`;
  }
  const label = getElement('ccStatusLabel');
              if (label) label.textContent = `Рівень:
${model.statusDescription}`;
};

const updateCcSkin = (model) => {
  const card = getElement('ccCard');
                  updateClassSet(card, CC_SKIN_CLASSES,
CC_SKIN_CLASS_MAP[model.tierCode]);
};
```

```
const renderTips = () => {
  const section = getElement('tipsSection');
  const list = getElement('tipsList');
  if (!section || !list) return;
  list.innerHTML = '';
  const tips = collectTips(metricState);
  if (!tips.length) {
    toggleHidden(section, true);
    return;
  }
  tips.forEach((tip) => {
    const item = document.createElement('li');
    item.textContent = tip;
    list.appendChild(item);
  });
  toggleHidden(section, false);
};

const setToastMessage = (id, message) => {
  const textElement = getElementById(TOAST_TEXT_IDS[id]);
  if (textElement) textElement.textContent = message;
};

const showToastNotification = (id, message) => {
  if (!message) return;
  const toastElement = getElementById(id);
  const Toast = window.bootstrap?.Toast;
  if (!toastElement || !Toast) return;
  setToastMessage(id, message);
      new Toast(toastElement, { delay: TOAST_DELAY_MS
}).show();
};

const notifyMiMetric = (miValue) => {
  const message = determineMiToastMessage(miValue);
  if (!message) {
    toastState = { ...toastState, mi: '' };
    return;
  }
  if (message === toastState.mi) return;
  toastState = { ...toastState, mi: message };
  showToastNotification(TOAST_MI_ID, message);
};

const notifyCcMetric = (ccValue) => {
  const message = determineCcToastMessage(ccValue);
  if (!message) {
    toastState = { ...toastState, cc: '' };
    return;
  }
  if (message === toastState.cc) return;
  toastState = { ...toastState, cc: message };
  showToastNotification(TOAST_CC_ID, message);
};

const resetMiView = () => {
  setTextContent('miChip', DEFAULT_METRIC_VALUE);
  setTextContent('miVolume', DEFAULT_METRIC_VALUE);
  setTextContent('miCyclomatic', DEFAULT_METRIC_VALUE);
  setTextContent('miLoc', DEFAULT_METRIC_VALUE);
                                    setTextContent('miComments',
DEFAULT_COMMENTS_VALUE);
  const statusLabel = getElement('miStatusLabel');
            if (statusLabel) statusLabel.textContent =
MI_DEFAULT_STATUS_TEXT;
  const badge = getElement('miStatusBadge');
  if (badge) {
    badge.textContent = MI_DEFAULT_BADGE_TEXT;
    badge.className = BADGE_DEFAULT_CLASS;
  }
  const card = getElement('miCard');
  updateClassSet(card, MI_SKIN_CLASSES, null);
};

const resetCcView = () => {
  setTextContent('ccChip', DEFAULT_METRIC_VALUE);
  const statusLabel = getElement('ccStatusLabel');
            if (statusLabel) statusLabel.textContent =
CC_DEFAULT_STATUS_TEXT;
  const badge = getElement('ccStatusBadge');
  if (badge) {
    badge.textContent = CC_DEFAULT_BADGE_TEXT;
    badge.className = BADGE_DEFAULT_CLASS;
  }
```

```
  const card = getElement('ccCard');
  updateClassSet(card, CC_SKIN_CLASSES, null);
};

// ========================================
// ПУБЛІЧНИЙ API
// ========================================

/**
 * Рендерить показники Maintainability Index.
 * @param {object} result - Дані MI від аналізатора.
 */
export function renderMI(result) {
  if (!result || typeof result !== 'object') return;
  const model = buildMiDisplayModel(result);
  updateMiChip(model);
  updateMiStatus(model);
  updateMiMetrics(model);
  updateMiSkin(model);
  metricState = { ...metricState, mi: model.miValue };
  renderTips();
  notifyMiMetric(model.miValue);
}

/**
 * Рендерить показники когнітивної складності.
 * @param {number} ccValue - Значення когнітивної складності.
 */
export function renderCC(ccValue) {
  const model = buildCcDisplayModel(ccValue);
  updateCcChip(model);
  updateCcStatus(model);
  updateCcSkin(model);
  metricState = { ...metricState, cc: model.ccValue };
  renderTips();
  notifyCcMetric(model.ccValue);
}

/**
 * Заповнює поле з кодом демонстраційним прикладом.
 * @param {string} code - Код, що буде вставлено у поле вводу.
 */
export function setSample(code) {
  const sanitized = `${(code || '').trim()}${NEWLINE}`;
  setInputValue('codeInput', sanitized);
  focusElement('codeInput');
}

/**
 * Скидає відображення метрик до початкового стану.
 */
export function resetMetricsView() {
  resetMiView();
  resetCcView();
  metricState = { mi: null, cc: null };
  toastState = { mi: '', cc: '' };
  renderTips();
}
```

public/js/stats.js

```
import { wilcoxonTest } from './wilcoxon.js';

const METRIC_CONFIG = {
  mi: {
    key: 'mi',
    title: 'Індекс підтримуваності',
    subtitle: 'Maintainability Index',
    shortLabel: 'MI',
    higherBetter: true,
    icon: 'gear',
    color: 'success',
    chartId: 'chart-mi',
    badgeClass: (value) => {
      if (value >= 20) return 'bg-success';
      if (value >= 10) return 'bg-warning text-dark';
      return 'bg-danger';
    },
  },
  cc: {
    key: 'cc',
```

```
    title: 'Когнітивна складність',
    subtitle: 'Cognitive Complexity',
    shortLabel: 'CC',
    higherBetter: false,
    icon: 'brain',
    color: 'primary',
    chartId: 'chart-cc',
    badgeClass: (value) => {
      if (value <= 10) return 'bg-success';
      if (value <= 20) return 'bg-warning text-dark';
      return 'bg-danger';
    },
  },
  cyclomatic: {
    key: 'cyclomatic',
    title: 'Цикломатична складність',
    subtitle: 'Cyclomatic Complexity',
    shortLabel: 'Cyclomatic',
    higherBetter: false,
    icon: 'diagram-3',
    color: 'warning',
    chartId: 'chart-cyclomatic',
    badgeClass: (value) => {
      if (value <= 10) return 'bg-success';
      if (value <= 20) return 'bg-warning text-dark';
      return 'bg-danger';
    },
  },
  loc: {
    key: 'loc',
    title: 'Кількість рядків',
    subtitle: 'Lines of Code',
    shortLabel: 'LOC',
    higherBetter: false,
    icon: 'file-code',
    color: 'info',
    chartId: 'chart-loc',
  },
};

const ORDERED_METRICS = ['mi', 'cc', 'cyclomatic', 'loc'];
const statusEl = document.getElementById('stats-status');

const setStatus = (variant, text, icon = 'info-circle-fill') => {
  if (!statusEl) return;
        statusEl.className = `alert alert-${variant} d-flex
align-items-center gap-2`;
                statusEl.innerHTML = `<i class="bi
bi-${icon}"></i><span>${text}</span>`;
};

const setSuccessStatus = (timestamp) => {
  if (!statusEl) return;
            const formatted = timestamp ? new
Date(timestamp).toLocaleString('uk-UA') : '';
  const suffix = formatted ? ` Оновлено ${formatted}.` : '';
          setStatus('success', `Дані завантажено.${suffix}`,
'check-circle-fill');
};

const fetchStats = async () => {
   const response = await fetch('/data/examples-stats.json', { cache:
'no-cache' });
   if (!response.ok) throw new Error(`Не вдалося завантажити дані
(${response.status})`);
  return response.json();
};

const formatNumber = (value, digits = 1) => {
  if (!Number.isFinite(value)) return '—';
  const numeric = Number(value);
  if (digits <= 0) return Math.round(numeric).toString();
  return numeric.toFixed(digits);
};

const formatMetricValue = (value) => {
  if (!Number.isFinite(value)) return '—';
            return Number.isInteger(value) ? value :
Number(value.toFixed(1));
};

const formatDelta = (value) => {
  if (!Number.isFinite(value) || value === 0) return '0';
  const abs = Math.abs(value);
```

```
  const formatted = Number.isInteger(abs) ? abs : abs.toFixed(1);
  return value > 0 ? `+${formatted}` : `-${formatted}`;
};

const formatPValue = (value) => {
  if (!Number.isFinite(value)) return '—';
  if (value < 0.0001) return '<0.0001';
    if (value < 0.001) return value.toFixed(4); // 0.0005 замість 5.49e-4
  if (value < 0.01) return value.toFixed(4); // 4 знаки після коми
  if (value < 0.1) return value.toFixed(3); // 3 знаки після коми
  return value.toFixed(2); // 2 знаки після коми
};

const buildPairedMetrics = (data, metrics = ['mi', 'cc', 'cyclomatic']) => {
      const hasMetrics = (item) => metrics.every((key) => Number.isFinite(item?.[key]));

  const fromPairwise =
    Array.isArray(data?.pairwise) && data.pairwise.length
      ? data.pairwise
                        .filter((item) => hasMetrics(item?.es5) && hasMetrics(item?.es6))
          .map((item) => ({ fileName: item.fileName, es5: item.es5, es6: item.es6 }))
      : [];

  if (fromPairwise.length) return fromPairwise;

  const es5Map = new Map(
        (data?.categories?.es5 || []).filter(hasMetrics).map((item) => [item.fileName, item]),
  );

  return (data?.categories?.es6 || [])
                    .filter((item) => hasMetrics(item) && es5Map.has(item.fileName))
            .map((item) => ({ fileName: item.fileName, es5: es5Map.get(item.fileName), es6: item }));
};

const updateHeadlineCounts = (data) => {
        const pairCount = Array.isArray(data?.pairwise) ? data.pairwise.length : 0;
      const es5Count = Array.isArray(data?.categories?.es5) ? data.categories.es5.length : 0;
      const es6Count = Array.isArray(data?.categories?.es6) ? data.categories.es6.length : 0;
  const totalCount = es5Count + es6Count || pairCount * 2 || 0;

  const pairEl = document.getElementById('file-pair-count');
  if (pairEl) pairEl.textContent = pairCount || 0;

  const totalEl = document.getElementById('file-total-count');
  if (totalEl) {
    const parts = [];
    if (es5Count) parts.push(`ES5: ${es5Count}`);
    if (es6Count) parts.push(`ES6: ${es6Count}`);
    const suffix = parts.length ? ` (${parts.join(', ')})` : '';
            totalEl.textContent = `Опрацьовано ${totalCount} файлів${suffix}`;
  }
};

const escapeCsvValue = (value) => `"${String(value ?? '').replace(/"/g, '""')}"`;

const buildComparisonCsv = (pairwise) => {
  const header = [
    '№ прикладу',
    'Назва файлу',
    'MI (ES5)',
    'MI (ES6)',
    'CC (ES5)',
    'CC (ES6)',
  ];

  const rows = pairwise.map((item, index) => [
    index + 1,
    item.fileName || '—',
    formatMetricValue(item.es5?.mi),
    formatMetricValue(item.es6?.mi),
    formatMetricValue(item.es5?.cc),
```

```
    formatMetricValue(item.es6?.cc),
  ]);

  return [header, ...rows]
    .map((row) => row.map(escapeCsvValue).join(';'))
    .join('\n');
};

const downloadCsv = (content, filename) => {
      const blob = new Blob(['\ufeff', content], { type:
'text/csv;charset=utf-8;' });
  const link = document.createElement('a');
  link.href = URL.createObjectURL(blob);
  link.download = filename;
  link.style.display = 'none';
  document.body.appendChild(link);
  link.click();
  document.body.removeChild(link);
  URL.revokeObjectURL(link.href);
};

const getImprovementBadge = (improvement) => {
  if (!Number.isFinite(improvement) || improvement === 0) {
    return '<span class="badge bg-secondary">Без змін</span>';
  }

  const value = Math.abs(improvement).toFixed(2);
  if (improvement > 0) {
          return `<span class="badge bg-success"><i class="bi
bi-arrow-up me-1"></i>${value}% краще</span>`;
  }

       return `<span class="badge bg-danger"><i class="bi
bi-arrow-down me-1"></i>${value}% гірше</span>`;
};

const renderSummaryCards = (data) => {
  const container = document.getElementById('summary-cards');
  if (!container) return;

  const html = ORDERED_METRICS.map((key) => {
    const metric = METRIC_CONFIG[key];
    const es5Mean = data.statistics.es5[key]?.mean ?? 0;
    const es6Mean = data.statistics.es6[key]?.mean ?? 0;
    const improvement = data.improvements[key] ?? 0;

    return `
      <div class="col-12 col-md-6 col-lg-3">
        <div class="card rounded-4 border-0 shadow-sm h-100">
          <div class="card-body">
            <div class="d-flex align-items-center gap-2 mb-3">
                    <i class="bi bi-${metric.icon} text-${metric.color}
fs-4"></i>
              <div>
                <h5 class="mb-0 small">${metric.title}</h5>
                <small class="text-muted">${metric.subtitle}</small>
              </div>
            </div>
            <div class="d-flex justify-content-between align-items-end
mb-3">
              <div>
                <div class="small text-muted">ES5</div>
                      <div class="h4 mb-0">${formatNumber(es5Mean,
1)}</div>
              </div>
              <i class="bi bi-arrow-right text-muted"></i>
              <div>
                <div class="small text-muted">ES6</div>
                                                <div class="h4 mb-0
text-${metric.color}">${formatNumber(es6Mean, 1)}</div>
              </div>
            </div>
            ${getImprovementBadge(improvement)}
          </div>
        </div>
      </div>
    `;
  }).join('');

  container.innerHTML = html;
};

const createComparisonChart = (containerId, metricKey, data) => {
  const categories = data.pairwise.map((item) =>
```

```
    item.fileName.replace('.js', '').replace(/_/g, ' '),
  );
        const es5Values = data.pairwise.map((item) => item.es5[metricKey]);
        const es6Values = data.pairwise.map((item) => item.es6[metricKey]);

  Highcharts.chart(containerId, {
    chart: { type: 'column', height: 400 },
    title: { text: null },
    xAxis: {
      categories,
      crosshair: true,
      labels: { rotation: -45, style: { fontSize: '11px' } },
    },
    yAxis: {
      min: 0,
      title: { text: 'Значення' },
    },
    tooltip: {
      shared: true,
      useHTML: true,
      formatter() {
        const es5Val = this.points?.[0]?.y ?? 0;
        const es6Val = this.points?.[1]?.y ?? 0;
        const delta = es6Val - es5Val;
        const deltaText = delta > 0 ? `+${delta}` : delta;

        return `
          <strong>${this.x}</strong><br/>
          ES5: <b>${es5Val}</b><br/>
          ES6: <b>${es6Val}</b><br/>
          Δ: <b>${deltaText}</b>
        `;
      },
    },
    plotOptions: {
      column: {
        pointPadding: 0.2,
        borderWidth: 0,
      },
    },
    series: [
      { name: 'ES5', data: es5Values, color: '#eb3349' },
      { name: 'ES6', data: es6Values, color: '#38ef7d' },
    ],
    credits: { enabled: false },
    legend: { align: 'center', verticalAlign: 'top' },
  });
};

const createComparisonCell = (es5Value, es6Value, higherIsBetter, badgeClassFn) => {
  const valueEs5 = Number(es5Value);
  const valueEs6 = Number(es6Value);
        const hasValues = Number.isFinite(valueEs5) && Number.isFinite(valueEs6);
  const delta = hasValues ? valueEs6 - valueEs5 : 0;
  const improved = hasValues ? (higherIsBetter ? delta > 0 : delta < 0) : false;
  const worsened = hasValues ? (higherIsBetter ? delta < 0 : delta > 0) : false;

  let arrowIcon = '<i class="bi bi-dash text-muted"></i>';
  let deltaText = 'без змін';
  let deltaClass = 'text-muted';

  if (hasValues && delta !== 0) {
    const arrow = improved ? 'arrow-up' : worsened ? 'arrow-down' : 'dash';
    const arrowClass = improved ? 'text-success' : 'text-danger';
    arrowIcon = `<i class="bi bi-${arrow} ${arrowClass}"></i>`;
    deltaClass = `${arrowClass} fw-bold`;
    deltaText = `Δ ${formatDelta(delta)}`;
  }

  const badge = (value, cls) => {
        if (!Number.isFinite(value)) return '<span class="text-muted">—</span>';
        return `<span class="badge ${cls} font-monospace metric-badge">${formatMetricValue(value)}</span>`;
  };

  const es5BadgeClass = badgeClassFn ? badgeClassFn(valueEs5) :
```

```
'bg-secondary';
  const es6BadgeClass = badgeClassFn ? badgeClassFn(valueEs6) :
'bg-secondary';

  return `
    <div class="metric-comparison-cell">
      <div class="metric-value">
        <span class="metric-label">ES5</span>
        ${badge(valueEs5, es5BadgeClass)}
      </div>
      <div class="metric-arrow">
        ${arrowIcon}
      </div>
      <div class="metric-value">
        <span class="metric-label">ES6</span>
        ${badge(valueEs6, es6BadgeClass)}
      </div>
      <div class="metric-delta">
        <small class="${deltaClass}">${deltaText}</small>
      </div>
    </div>
  `;
};

const renderComparisonTable = (data) => {
  const table = document.getElementById('comparison-table');
  if (!table) return;
  const tbody = table.querySelector('tbody');
  if (!tbody) return;

  if (!Array.isArray(data.pairwise) || !data.pairwise.length) {
    tbody.innerHTML =
                '<tr><td colspan="5" class="text-center text-muted
small">Немає даних для відображення</td></tr>';
    return;
  }

  const rows = data.pairwise
    .map((row) => {
      const cells = ORDERED_METRICS.map((key) => {
        const metric = METRIC_CONFIG[key];
        return `
          <td class="text-center">
                ${createComparisonCell(row.es5[key], row.es6[key],
metric.higherBetter, metric.badgeClass)}
          </td>
        `;
      }).join('');

      const shortName = row.fileName.replace('.js', '').replace(/_/g, '
');

      return `
        <tr>
          <td>
            <code class="text-primary">${shortName}</code><br>
            <small class="text-muted">${row.fileName}</small>
          </td>
          ${cells}
        </tr>
      `;
    })
    .join('');

  tbody.innerHTML = rows;
};

const setupExportButton = (pairwise) => {
  const button = document.getElementById('export-csv-btn');
  if (!button) return;

  const hasData = Array.isArray(pairwise) && pairwise.length > 0;
  if (!hasData) {
    button.setAttribute('disabled', 'disabled');
    button.title = 'Немає даних для експорту';
    return;
  }

  button.removeAttribute('disabled');
  button.title = 'Завантажити результати у CSV';
  button.onclick = () => {
    const csvContent = buildComparisonCsv(pairwise);
    downloadCsv(csvContent, 'es5-vs-es6-comparison.csv');
  };
```

```
};

const buildStatsTable = (stats) => {
  const header = `
    <thead>
      <tr>
        <th>Метрика</th>
        <th>Середнє</th>
        <th>Медіана</th>
        <th>Min</th>
        <th>Max</th>
        <th>StdDev</th>
      </tr>
    </thead>
  `;

  const body = ORDERED_METRICS.map((key) => {
    const metric = METRIC_CONFIG[key];
    const metricStats = stats[key] ?? {};
    return `
      <tr>
        <td>${metric.shortLabel}</td>
        <td>${formatNumber(metricStats.mean, 2)}</td>
        <td>${formatNumber(metricStats.median, 2)}</td>
        <td>${formatNumber(metricStats.min, 2)}</td>
        <td>${formatNumber(metricStats.max, 2)}</td>
        <td>${formatNumber(metricStats.stdDev, 2)}</td>
      </tr>
    `;
  }).join('');

  return `<div class="table-responsive">
    <table class="table table-sm table-striped mb-0 stats-table">
      ${header}
      <tbody>${body}</tbody>
    </table>
  </div>`;
};

const renderStatsTables = (statistics) => {
  const es5Container = document.getElementById('stats-es5-table');
  const es6Container = document.getElementById('stats-es6-table');

  if (es5Container && statistics?.es5) {
    es5Container.innerHTML = buildStatsTable(statistics.es5);
  }

  if (es6Container && statistics?.es6) {
    es6Container.innerHTML = buildStatsTable(statistics.es6);
  }
};

const getWilcoxonTrendText = (metricKey, direction) => {
  const metric = METRIC_CONFIG[metricKey];
    if (!direction) return 'Різниця між ES5 та ES6 практично відсутня.';

  const winner = direction > 0 ? 'ES6' : 'ES5';
   const action = metric.higherBetter ? 'має вищі значення' : 'має нижчі значення';

  return `${winner} ${action} для ${metric.subtitle}, що є кращим результатом.`;
};

const getEffectSizeLabel = (r) => {
  const abs = Math.abs(r);
  if (abs < 0.1) return 'дуже малий';
  if (abs < 0.3) return 'малий';
  if (abs < 0.5) return 'середній';
  return 'великий';
};

const renderWilcoxonAnalysis = (data) => {
  const container = document.getElementById('wilcoxon-cards');
  if (!container) return;

  const pairs = buildPairedMetrics(data);
                                  const infoEl =
document.getElementById('wilcoxon-sample-info');
  if (infoEl && pairs.length) {
      infoEl.textContent = `Тест Вілкоксона для парних вибірок (n=${pairs.length})`;
  }
```

```
  if (!pairs.length) {
    container.innerHTML =
            '<div class="col-12"><div class="alert alert-warning mb-0">Недостатньо даних для статистичного тесту</div></div>';
    return;
  }

  const metrics = ['mi', 'cc', 'cyclomatic'];
  const cards = metrics
    .map((key) => {
      const metric = METRIC_CONFIG[key];
      const result = wilcoxonTest(pairs, key);
          const badgeClass = result.isSignificant ? 'bg-success' : 'bg-secondary';
      const badgeText = result.isSignificant
        ? '✅ Статистично значущо'
        : 'ℹ️ Немає доказів різниці';

      return `
        <div class="col-12 col-md-4">
          <div class="card rounded-4 shadow-sm h-100">
            <div class="card-body">
              <h5 class="d-flex align-items-center gap-2 mb-1">
                <i class="bi bi-${metric.icon} text-${metric.color}"></i>
                <span>${metric.subtitle}</span>
              </h5>
              <p class="text-muted small mb-3">${metric.title}</p>
                          <p class="fs-3 fw-semibold mb-2">p = ${formatPValue(result.pValue)}</p>
                                                <span class="badge ${badgeClass}">${badgeText}</span>
              <div class="mt-3 small">
                <div class="d-flex justify-content-between mb-1">
                  <span>ES6 краще:</span>
                  <strong>${result.betterCount || 0}</strong>
                </div>
                <div class="d-flex justify-content-between">
                  <span>ES6 гірше:</span>
                  <strong>${result.worseCount || 0}</strong>
                </div>
              </div>
                                                <p class="small mt-3 mb-1">${getWilcoxonTrendText(key, result.direction)}</p>
              <p class="small text-muted mb-0">
                          W = ${formatNumber(result.W, 2)}; n = ${result.sampleSize}
                <br>
                    Effect size: r = ${formatNumber(result.effectSize, 2)} (${getEffectSizeLabel(result.effectSize)})
              </p>
            </div>
          </div>
        </div>
      `;
    })
    .join('');

  container.innerHTML = cards;
};

const initDashboard = async () => {
  try {
    setStatus('info', 'Завантажуємо результати аналізу…');
    const data = await fetchStats();

    updateHeadlineCounts(data);
    setSuccessStatus(data.timestamp);
    renderSummaryCards(data);

    if (data.pairwise?.length) {
                        ORDERED_METRICS.forEach((key) => createComparisonChart(METRIC_CONFIG[key].chartId, key, data));
      setupExportButton(data.pairwise);
    } else {
        setStatus('warning', 'Не знайдено парних результатів для побудови графіків');
    }

    renderComparisonTable(data);
    renderStatsTables(data.statistics);
    renderWilcoxonAnalysis(data);
  } catch (error) {
```

```
    console.error(error);
    setStatus(
      'danger',
        'Не вдалося завантажити статистику. Переконайтеся, що виконано аналіз.',
      'exclamation-triangle-fill',
    );
  }
};

initDashboard();
```

public/js/utils/ast.js

```
// ast.js — базовий парсинг та обхід AST (@babel/parser)
import * as babelParser from '@babel/parser';

/* ================================
   Константи
   ================================ */
export const SOURCE_TYPE = 'unambiguous';
export const ALLOW_RETURN_OUTSIDE_FN = true;
export const EMIT_TOKENS = true;

export const BABEL_PLUGINS = [
  'jsx',
  'typescript',
  'classProperties',
  'classPrivateProperties',
  'classPrivateMethods',
  'decorators-legacy',
  'dynamicImport',
  'exportDefaultFrom',
  'exportNamespaceFrom',
  'objectRestSpread',
  'optionalCatchBinding',
  'optionalChaining',
  'nullishCoalescingOperator',
  'topLevelAwait',
  'doExpressions',
  'logicalAssignment',
];

export const PARSER_OPTIONS = {
  sourceType: SOURCE_TYPE,
                                        allowReturnOutsideFunction: ALLOW_RETURN_OUTSIDE_FN,
  plugins: BABEL_PLUGINS,
  tokens: EMIT_TOKENS,
};

const toStringSafe = (v) => String(v ?? '');
const isAstNode = (v) => !!v && typeof v.type === 'string';
const ownKeys = (o) => Object.keys(o);
const hasOwn = (o, k) => Object.prototype.hasOwnProperty.call(o, k);

const pushIfNode = (stack, v) => {
  if (isAstNode(v)) stack.push(v);
};

const pushArrayNodesReversed = (stack, arr) => {
  for (let i = arr.length - 1; i >= 0; i--) {
    pushIfNode(stack, arr[i]);
  }
};

export function parse(src) {
  return babelParser.parse(toStringSafe(src), PARSER_OPTIONS);
}

export function walk(node, enter) {
  if (!isAstNode(node) || typeof enter !== 'function') return;

  const stack = [node];

  while (stack.length) {
    const n = stack.pop();
    if (!isAstNode(n)) continue;

    enter(n);
```

```
    for (const key of ownKeys(n)) {
      if (!hasOwn(n, key)) continue;
      const val = n[key];
      if (!val) continue;

      if (Array.isArray(val)) {
        pushArrayNodesReversed(stack, val);
      } else {
        pushIfNode(stack, val);
      }
    }
  }
}
```

public/js/utils/cognitive.js

```
import { parse } from './ast.js';

// ========================================
// Константи
// ========================================
const LOGICAL_OPERATORS = new Set(['&&', '||']);
const LOOP_TYPES = new Set(['ForStatement', 'ForInStatement', 'ForOfStatement', 'WhileStatement', 'DoWhileStatement']);
const FUNCTION_TYPES = new Set(['FunctionDeclaration', 'FunctionExpression', 'ArrowFunctionExpression']);
const CONTROL_TOKENS = new Set(['if', 'else', 'else if', 'for', 'while', 'do', 'catch', 'switch']);
const STRUCTURED_TOKENS = new Set(['if', 'for', 'while', 'do', 'catch']);
const COMMENT_BLOCK_REGEX = /\/\*[\s\S]*?\*\//g;
const COMMENT_LINE_REGEX = /(^|[^:])\/\/.*$/gm;
const LABEL_BREAK_REGEX = /\bbreak\s+[A-Za-z_$][\w$]*/g;
const LABEL_CONTINUE_REGEX = /\bcontinue\s+[A-Za-z_$][\w$]*/g;
const FALLBACK_TOKEN_REGEX = /\belse if\b|\bif\b|\belse\b|\bfor\b|\bwhile\b|\bdo\b|\bcatch\b|\bswitch\b|\?|&&|\|\||\{|\}|break|continue/g;

// ========================================
// Утиліти
// ========================================
const toStringSafe = (value) => String(value ?? '');
const isAstNode = (value) => Boolean(value && typeof value.type === 'string');
const isLogicalExpression = (node) => node?.type === 'LogicalExpression';
const isSwitchStatement = (node) => node?.type === 'SwitchStatement';
const isIfStatement = (node) => node?.type === 'IfStatement';
const isLabeledJump = (node) => (node?.type === 'BreakStatement' || node?.type === 'ContinueStatement') && Boolean(node.label);
const isTopLevelFunction = (node, parent) => parent?.type === 'Program' && node?.type === 'FunctionDeclaration';
const getChildNodes = (node) => Object.values(node || {}).flatMap((value) => toNodeArray(value));
const toNodeArray = (value) => {
  if (!value) return [];
  if (Array.isArray(value)) return value.filter(isAstNode);
  return isAstNode(value) ? [value] : [];
};
const stripCommentsFallback = (source) => toStringSafe(source)
  .replace(COMMENT_BLOCK_REGEX, '')
  .replace(COMMENT_LINE_REGEX, (match, prefix) => prefix);
const tokenizeFallback = (source) => stripCommentsFallback(source).match(FALLBACK_TOKEN_REGEX) || [];

// ========================================
// Класифікатори
// ========================================
const isElseIfBranch = (node, parent) => isIfStatement(node) && parent?.type === 'IfStatement' && parent.alternate === node;
const isElseBlock = (node, parent) => parent?.type === 'IfStatement' && parent.alternate === node && !isIfStatement(node);
const isFirstLogicalInSequence = (node, parent) => {
  if (!isLogicalExpression(node) ||
```

```
!LOGICAL_OPERATORS.has(node.operator)) return false;
 if (!isLogicalExpression(parent)) return true;
 return parent.operator !== node.operator;
};

// ========================================
// Підрахунок по AST
// ========================================
const cognitiveFromAst = (root) => computeNode(root, 0, null);

function computeNode(node, nesting, parent) {
 if (!isAstNode(node)) return 0;
    if (isIfStatement(node)) return complexityIf(node, nesting, parent);
    if (isSwitchStatement(node)) return complexitySwitch(node, nesting);
  if (LOOP_TYPES.has(node.type)) return complexityLoop(node, nesting);
  if (node.type === 'CatchClause') return complexityCatch(node, nesting);
       if (node.type === 'ConditionalExpression') return complexityConditional(node, nesting);
                  if (isLogicalExpression(node) && LOGICAL_OPERATORS.has(node.operator)) return complexityLogical(node, nesting, parent);
 if (isLabeledJump(node)) return 1;
            if (FUNCTION_TYPES.has(node.type)) return complexityFunction(node, nesting, parent);
    return getChildNodes(node).reduce((sum, child) => sum + computeNode(child, nesting, node), 0);
}

function complexityIf(node, nesting, parent) {
 const head = 1 + nesting;
 const test = computeNode(node.test, nesting, node);
  const consequent = computeNode(node.consequent, nesting + 1, node);
  const alternate = complexityIfAlternate(node.alternate, nesting, node);
 return head + test + consequent + alternate;
}

function complexityIfAlternate(alternate, nesting, parent) {
 if (!alternate) return 0;
  if (isIfStatement(alternate)) return complexityIf(alternate, nesting, parent);
 const body = computeNode(alternate, nesting + 1, parent);
 return 1 + body;
}

function complexitySwitch(node, nesting) {
  const discriminant = computeNode(node.discriminant, nesting, node);
 const cases = (node.cases || []).reduce((sum, switchCase) => {
  const test = computeNode(switchCase.test, nesting, switchCase);
   const body = (switchCase.consequent || []).reduce((acc, stmt) => acc + computeNode(stmt, nesting, switchCase), 0);
  return sum + test + body;
 }, 0);
 return 1 + discriminant + cases;
}

function complexityLoop(node, nesting) {
 const loopBody = loopComplexity(node, nesting);
 return 1 + nesting + loopBody;
}

function loopComplexity(node, nesting) {
 switch (node.type) {
  case 'ForStatement':
   return computeNode(node.init, nesting, node)
    + computeNode(node.test, nesting, node)
    + computeNode(node.update, nesting, node)
    + computeNode(node.body, nesting + 1, node);
  case 'ForInStatement':
  case 'ForOfStatement':
   return computeNode(node.left, nesting, node)
    + computeNode(node.right, nesting, node)
    + computeNode(node.body, nesting + 1, node);
  case 'WhileStatement':
   return computeNode(node.test, nesting, node)
    + computeNode(node.body, nesting + 1, node);
  case 'DoWhileStatement':
   return computeNode(node.body, nesting + 1, node)
```

```
      + computeNode(node.test, nesting, node);
    default:
      return 0;
  }
}

function complexityCatch(node, nesting) {
  const param = computeNode(node.param, nesting, node);
  const body = computeNode(node.body, nesting + 1, node);
  return 1 + nesting + param + body;
}

function complexityConditional(node, nesting) {
  const test = computeNode(node.test, nesting, node);
   const whenTrue = computeNode(node.consequent, nesting + 1,
node);
   const whenFalse = computeNode(node.alternate, nesting + 1,
node);
  return 1 + nesting + test + whenTrue + whenFalse;
}

function complexityLogical(node, nesting, parent) {
  const head = isFirstLogicalInSequence(node, parent) ? 1 : 0;
  const left = computeNode(node.left, nesting, node);
  const right = computeNode(node.right, nesting, node);
  return head + left + right;
}

function complexityFunction(node, nesting, parent) {
  const params = (node.params || []).reduce((sum, param) => sum +
computeNode(param, nesting, node), 0);
  const bodyNesting = isTopLevelFunction(node, parent) ? nesting :
nesting + 1;
  const body = computeNode(node.body, bodyNesting, node);
  return params + body;
}

// ========================================
// Фолбек без AST
// ========================================
const cognitiveFallback = (source) => {
  const tokens = tokenizeFallback(source);
      const finalState = tokens.reduce(applyFallbackToken,
createFallbackState());
  return finalState.complexity + countLabeledTransitions(source);
};

const createFallbackState = () => ({
  complexity: 0,
  nesting: 0,
  blockStack: [],
  previousToken: null,
  lastLogicalOperator: null,
});

const applyFallbackToken = (state, token) => {
  if (token === '{') return openBlock(state);
  if (token === '}') return closeBlock(state);
    if (token === 'else if') return addComplexity(state, 1 +
state.nesting, token);
  if (token === 'else') return addComplexity(state, 1, token);
  if (token === 'switch') return addComplexity(state, 1, token);
          if (STRUCTURED_TOKENS.has(token)) return
addComplexity(state, 1 + state.nesting, token);
  if (token === '?' ) return addComplexity(state, 1, token);
           if (LOGICAL_OPERATORS.has(token)) return
handleLogicalOperator(state, token);
  return setPreviousToken(resetLogical(state), token);
};

const openBlock = (state) => {
                    const increases =
CONTROL_TOKENS.has(state.previousToken);
  const nextStack = [...state.blockStack, increases];
  const nextNesting = increases ? state.nesting + 1 : state.nesting;
  return {
    ...state,
    blockStack: nextStack,
    nesting: nextNesting,
    previousToken: '{',
    lastLogicalOperator: null,
  };
};
```

```
const closeBlock = (state) => {
  if (!state.blockStack.length) {
    return { ...state, previousToken: '}', lastLogicalOperator: null };
  }
  const nextStack = state.blockStack.slice(0, -1);
  const increases = state.blockStack[state.blockStack.length - 1];
    const decreased = increases ? Math.max(0, state.nesting - 1) :
state.nesting;
  return {
    ...state,
    blockStack: nextStack,
    nesting: decreased,
    previousToken: '}',
    lastLogicalOperator: null,
  };
};

const addComplexity = (state, amount, token) => ({
  ...resetLogical(state),
  complexity: state.complexity + amount,
  previousToken: token,
});

const handleLogicalOperator = (state, operator) => {
  if (state.lastLogicalOperator === operator) {
    return { ...state, previousToken: operator };
  }
  return {
    ...state,
    complexity: state.complexity + 1,
    previousToken: operator,
    lastLogicalOperator: operator,
  };
};

const resetLogical = (state) => ({
  ...state,
  lastLogicalOperator: null,
});

const setPreviousToken = (state, token) => ({
  ...state,
  previousToken: token,
});

const countLabeledTransitions = (source) => {
                                    const                breaks                =
(toStringSafe(source).match(LABEL_BREAK_REGEX)                ||
[]).length;
                                  const                continues                =
(toStringSafe(source).match(LABEL_CONTINUE_REGEX)        ||
[]).length;
  return breaks + continues;
};

// ========================================
// Головна функція
// ========================================
export function cognitiveComplexity(source) {
  const code = toStringSafe(source);
  try {
    const ast = parse(code);
    return cognitiveFromAst(ast.program || ast);
  } catch {
    return cognitiveFallback(code);
  }
}
```

public/js/utils/cyclomatic.js

```
// public/js/utils/cyclomatic.js
import { parse, walk } from './ast.js';

const DECISION_NODE_TYPES = new Set([
  'IfStatement',
  'ForStatement',
  'ForInStatement',
  'ForOfStatement',
  'WhileStatement',
  'DoWhileStatement',
  'CatchClause',
```

```
  'ConditionalExpression',
  'LogicalExpression',
]);

const LOGICAL_OPERATORS = new Set(['&&', '||', '??']);

const FALLBACK_PATTERN = /\b(if|for|while|do|case|catch)\b|\?\?|&&|\|\||\s\?\s|\?\.(?:\(|\[)?/g;

/** Перетворює джерело у AST або повертає вже збудований AST */
const ensureAst = (source) => (typeof source === 'string' ? parse(source) : source);

/** Перевіряє, чи додає вузол нову гілку за рахунок арифметики МакКейба */
const isDecisionNode = (node) => {
  if (!DECISION_NODE_TYPES.has(node.type)) return false;
  if (node.type === 'LogicalExpression') return LOGICAL_OPERATORS.has(node.operator);
  return true;
};

/** Враховує лише іменовані гілки switch (без default) */
const isCountableSwitchCase = (node) => node.type === 'SwitchCase' && Boolean(node.test);

/** Підраховує кількість операторів optional chaining у сирому коді (?. або ?.()) */
const countOptionalChaining = (source) => {
  if (typeof source !== 'string') return 0;
  const matches = source.match(/\?\.(?!\?)/g);
  return matches ? matches.length : 0;
};

/** Акумулює кількість гілок у всьому AST */
const countDecisionNodes = (astRoot, source) => {
  let decisions = 0;

  walk(astRoot.program || astRoot, (node) => {
    decisions += isDecisionNode(node) ? 1 : 0;
    decisions += isCountableSwitchCase(node) ? 1 : 0;
  });

  return decisions + countOptionalChaining(source);
};

/** Резервний підрахунок, коли парсер не зміг розібрати код */
const countDecisionsFallback = (source) => {
  if (typeof source !== 'string') return 0;
  const matches = source.match(FALLBACK_PATTERN);
  return matches ? matches.length : 0;
};

/**
 * Юніт-приклади:
 * if (a) {}              // CC = 2
 * a && b                 // CC = 2
 * a && b || c            // CC = 3
 * a ? b : c              // CC = 2
 * try {} catch {}        // CC = 2
 * if (a && b) {}         // CC = 3
 */
export function cyclomaticComplexity(source) {
  try {
    const ast = ensureAst(source);
    return 1 + countDecisionNodes(ast, source);
  } catch {
    return 1 + countDecisionsFallback(source);
  }
}
```

public/js/utils/halstead.js

```
import { parse } from './ast.js';

// ============================================
// КОНСТАНТИ
// ============================================

const EOF_TOKEN = 'eof';
```

```
const OPERATOR_SYMBOLS = new Set([
  '==', '===', '!=', '!==', '<', '>', '<=', '>=',
  '+', '-', '*', '/', '%', '**',
  '++', '--', '+=', '-=', '*=', '/=', '%=', '**=',
  '<<', '>>', '>>>', '<<=', '>>=', '>>>=',
  '&', '|', '^', '~', '&=', '|=', '^=',
  '&&', '||', '??', '?', ':', '::', ',', '.', '=', '=>',
   '!', 'in', 'instanceof', 'new', 'delete', 'typeof', 'void', 'yield', 'await', '?.',
]);

const OPERAND_TOKEN_LABELS = new Set(['name', 'num', 'string', 'regexp', 'template']);

const LITERAL_KEYWORDS = new Set(['true', 'false', 'null', 'undefined']);

const TOKENS_TO_SKIP = new Set([EOF_TOKEN]);

// ========================================
// ВТОРОРЯДНІ ДОПОМІЖНІ ФУНКЦІЇ
// ========================================

const incrementCount = (map, key) => {
  map.set(key, (map.get(key) || 0) + 1);
};

const sumMapValues = (map) => [...map.values()].reduce((sum, count) => sum + count, 0);

const getTokenLabel = (token) => token?.type?.label;

const getTokenValue = (token) => token?.value;

const log2 = (n) => Math.log2(n);

// ========================================
// КЛАСИФІКАТОРИ ТОКЕНІВ
// ========================================

const shouldSkipToken = (token) => TOKENS_TO_SKIP.has(getTokenLabel(token));

const isLiteralKeyword = (value) => LITERAL_KEYWORDS.has(value);

const isOperandByLabel = (label) => OPERAND_TOKEN_LABELS.has(label);

const isOperatorSymbol = (value) => typeof value === 'string' && OPERATOR_SYMBOLS.has(value);

const isOperatorToken = (token) => {
  const label = getTokenLabel(token);
  const value = getTokenValue(token);
  if (isLiteralKeyword(value)) return false;
  return isOperatorSymbol(value) || !isOperandByLabel(label);
};

const getOperatorKey = (token) => {
  const value = getTokenValue(token);
  const label = getTokenLabel(token);
  return isOperatorSymbol(value) ? value : label || String(value);
};

const getOperandKey = (token) => {
  const value = getTokenValue(token);
  const label = getTokenLabel(token);
  if (isLiteralKeyword(value)) return String(value);
  return label === 'name' ? String(value) : String(value ?? label);
};

// ========================================
// ПАРСИНГ
// ========================================

const parseToTokens = (sourceCode) => {
  try {
    const ast = parse(sourceCode);
    return ast.tokens || [];
  } catch {
    return [];
  }
```

```
};

// ========================================
// ОБРОБКА ТОКЕНІВ
// ========================================

/**
 * Оновлює лічильники операторів та операндів для одного токена.
 */
const processToken = (token, operatorCounts, operandCounts) => {
  if (!token || shouldSkipToken(token)) return;
  if (isOperatorToken(token)) {
    incrementCount(operatorCounts, getOperatorKey(token));
  } else {
    incrementCount(operandCounts, getOperandKey(token));
  }
};

/**
 * Повертає мапи підрахунків для всього набору токенів.
 */
const processTokens = (tokens) => tokens.reduce(
  (acc, token) => {
    processToken(token, acc.operatorCounts, acc.operandCounts);
    return acc;
  },
  { operatorCounts: new Map(), operandCounts: new Map() },
);

// ========================================
// ОБЧИСЛЕННЯ МЕТРИК
// ========================================

const calculateBasicMetrics = (operatorCounts, operandCounts) => ({
  n1: operatorCounts.size,
  n2: operandCounts.size,
  N1: sumMapValues(operatorCounts),
  N2: sumMapValues(operandCounts),
});

const calculateDerivedMetrics = (n1, n2, N1, N2) => {
  const vocabulary = n1 + n2;
  const length = N1 + N2;
  const volume = vocabulary > 0 ? length * log2(vocabulary) : 0;
  return { vocabulary, length, volume };
};

// ========================================
// ГОЛОВНА ФУНКЦІЯ
// ========================================

/**
 * Обчислює метрики Хелстеда для переданого коду.
 * @param {string} sourceCode рядок із JS кодом
 * @returns {{n1:number,n2:number,N1:number,N2:number,vocabulary:number,length:number,volume:number}}
 */
export function halstead(sourceCode) {
  const tokens = parseToTokens(sourceCode);
  const { operatorCounts, operandCounts } = processTokens(tokens);
  const metrics = calculateBasicMetrics(operatorCounts, operandCounts);
  const derived = calculateDerivedMetrics(metrics.n1, metrics.n2, metrics.N1, metrics.N2);
  return { ...metrics, ...derived };
}
```

public/js/utils/maintainability.js

```
import { cyclomaticComplexity } from './cyclomatic.js';
import { halstead } from './halstead.js';
import { countCodeLines, stripAllComments } from './string.js';

const MI_BASELINE = 171; // базове значення формули
const COEFF_LN_VOLUME = 5.2; // коефіцієнт при ln(об'єму Гелстеда)
const COEFF_CYCLOMATIC = 0.23; // коефіцієнт при
```

```
цикломатичній складності
const COEFF_LN_LOC = 16.2; // коефіцієнт при ln(LOC)
const SCALE_FACTOR = 100 / MI_BASELINE; // нормалізація
у межі 0..100

/** Обмежує значення діапазоном [min, max] */
const clamp = (value, min, max) => Math.min(max,
Math.max(min, value));

/** Дає безпечний аргумент для логарифма (щонайменше 1) */
const safeLn = (value) => Math.log(Math.max(1, value));

/** Гарантує, що показник не буде від'ємним */
const ensureNonNegative = (value) => Math.max(0, value);

/** Повертає об'єм Гелстеда для вхідного коду */
const computeHalsteadVolume = (source) => {
  const { volume = 0 } = halstead(source) || {};
  return volume;
};

/** Повертає цикломатичну складність, обрізаючи негативні
значення */
const computeCyclomaticComplexity = (source) =>
  ensureNonNegative(cyclomaticComplexity(source));

/** Рахує кількість логічних рядків коду (без коментарів) */
const computeLogicalLoc = (source) => {
  const withoutComments = stripAllComments(source);
  return countCodeLines(withoutComments);
};

/** Застосовує офіційну формулу Maintainability Index */
const calculateMaintainabilityIndex = (volume, cyclomatic, loc) =>
{
  const rawScore =
    MI_BASELINE
    - COEFF_LN_VOLUME * safeLn(volume)
    - COEFF_CYCLOMATIC * cyclomatic
    - COEFF_LN_LOC * safeLn(loc);

  const scaled = rawScore * SCALE_FACTOR;
  return clamp(scaled, 0, 100);
};

/**
 * Обчислює Maintainability Index за формулою:
 * MI = max(0, (171 - 5.2*ln(V) - 0.23*G - 16.2*ln(LOC)) *
100/171)
 *
 * @param {string} source - Вихідний код JavaScript
 * @returns {Object} Об'єкт з метриками:
 *   - mi: Maintainability Index (0-100, більше = краще)
 *   - loc: Lines of Code (логічні рядки без коментарів)
 *   - cyclomatic: Цикломатична складність
 *   - halsteadVolume: Об'єм Гелстеда
 *   - V: Alias для halsteadVolume
 *   - G: Alias для cyclomatic
 *
 * @example
 * const result = maintainabilityIndex('function sum(a, b) { return a
+ b; }');
 * console.log(result.mi); // ~70-80
 */
export function maintainabilityIndex(source) {
  const code = typeof source === 'string' ? source : '';
  const loc = computeLogicalLoc(code);

  if (!source || String(source).trim().length === 0) {
    return {
      mi: 100,
      loc: 0,
      cyclomatic: 0,
      halsteadVolume: 0,
      V: 0,
      G: 0,
    };
  }

  const halsteadVolume = computeHalsteadVolume(code);
  const cyclomatic = computeCyclomaticComplexity(code);
      const mi = calculateMaintainabilityIndex(halsteadVolume,
cyclomatic, loc);
```

```
  return {
    mi,
    loc,
    cyclomatic,
    halsteadVolume,
    V: Math.max(1, halsteadVolume),
    G: cyclomatic,
  };
}
```

public/js/utils/string.js

```
import { parse } from './ast.js';

// ========================================
// КОНСТАНТИ
// ========================================

const NEWLINE_REGEX = /\r\n?/g;

const COMMENT_NODE_TYPES = new Set(['CommentLine', 'CommentBlock']);
const NON_CODE_TOKEN_LABELS = new Set(['{', '}', ';']);
const STRUCTURAL_ONLY_LABELS = new Set(['(', ')', '[', ']', ',']);
const FUNCTION_KEYWORD = 'function';
const EOF_LABEL = 'eof';

// ========================================
// ДОПОМІЖНІ ФУНКЦІЇ
// ========================================

const normalizeNewlines = (source) => String(source ?? '').replace(NEWLINE_REGEX, '\n');

const parseSource = (source) => {
  try {
    const ast = parse(source);
    return {
      tokens: ast.tokens ?? [],
      comments: ast.comments ?? [],
    };
  } catch {
    return { tokens: [], comments: [] };
  }
};

const isCommentNode = (node) => COMMENT_NODE_TYPES.has(node?.type);

const getCommentRanges = (comments) =>
  comments
    .filter(isCommentNode)
    .map(({ start, end }) => ({ start, end }))
    .sort((a, b) => a.start - b.start);

const preserveNewlines = (text) => {
  const newlineCount = (text.match(/\n/g) || []).length;
  return newlineCount > 0 ? '\n'.repeat(newlineCount) : '';
};

const removeRangesFromSource = (source, ranges) => {
  if (!ranges.length) return source;

  let result = '';
  let lastIndex = 0;

  for (const { start, end } of ranges) {
    if (start < lastIndex) continue; // перекриваючі діапазони пропускаємо
    result += source.slice(lastIndex, start);
    result += preserveNewlines(source.slice(start, end));
    lastIndex = end;
  }

  return result + source.slice(lastIndex);
};

const collectTokensByLine = (tokens) => {
  const lines = new Map();
```

```
  for (const token of tokens) {
    const label = token?.type?.label;
    const line = token?.loc?.start?.line;

    if (!token || !line || label === EOF_LABEL) continue;
    const bucket = lines.get(line) ?? [];
    bucket.push(token);
    lines.set(line, bucket);
  }

  return lines;
};

const isStructuralToken = (token, hasFunctionKeyword) => {
  const label = token?.type?.label;
  const keyword = token?.type?.keyword;

  if (!label) return true;
            if       (NON_CODE_TOKEN_LABELS.has(label)       ||
STRUCTURAL_ONLY_LABELS.has(label)) return true;
  if (keyword === FUNCTION_KEYWORD) return true;
  if (hasFunctionKeyword && label === 'name') return true;

  return false;
};

const hasMeaningfulToken = (tokens) => {
        const    hasFunctionKeyword    =    tokens.some((token)    =>
token?.type?.keyword === FUNCTION_KEYWORD);
       return    tokens.some((token)    =>    !isStructuralToken(token,
hasFunctionKeyword));
};

const fallbackCountCodeLines = (source) =>
  normalizeNewlines(source)
    .split('\n')
    .map((line) => line.trim())
     .filter((line) => line.length > 0 && line !== '{' && line !== '}'
&& line !== ';')
    .length;

// ========================================
// ПУБЛІЧНИЙ API
// ========================================

/**
 * Видаляє усі коментарі, використовуючи парсер Babel.
  * Зберігає структуру перенесень рядків, щоб номери рядків
залишались сталими.
 */
export function stripAllComments(sourceCode) {
  const normalized = normalizeNewlines(sourceCode);
  const { comments } = parseSource(normalized);

  if (!comments.length) return normalized;

  const ranges = getCommentRanges(comments);
  return removeRangesFromSource(normalized, ranges);
}

/**
  * Рахує кількість рядків коду за токенами Babel (виняток:
рядки тільки з { } ; не враховуються).
 */
export function countCodeLines(sourceCode) {
  const normalized = normalizeNewlines(sourceCode);
  const { tokens } = parseSource(normalized);

  if (!tokens.length) return fallbackCountCodeLines(normalized);

  const lineMap = collectTokensByLine(tokens);
  let count = 0;

  for (const tokensOnLine of lineMap.values()) {
    if (hasMeaningfulToken(tokensOnLine)) count += 1;
  }

  return count;
}

/**
   * Залишено для зворотної сумісності. Використовуйте
countCodeLines().
 */
```

```
export function countLines(sourceCode) {
  return countCodeLines(sourceCode);
}

/**
   * Залишено для зворотної сумісності. Використовуйте stripAllComments().
 */
export function stripBlockComments(sourceCode) {
  return stripAllComments(sourceCode);
}

/**
   * Залишено для зворотної сумісності. Використовуйте stripAllComments().
 */
export function stripLineComments(sourceCode) {
  return stripAllComments(sourceCode);
}
```

public/js/wilcoxon.js

```
const METRIC_DIRECTION = {
  mi: 1, // higher is better
  cc: -1, // lower is better
  cyclomatic: -1, // lower is better
};

const erf = (x) => {
  // Abramowitz and Stegun approximation
  const sign = x >= 0 ? 1 : -1;
  const absX = Math.abs(x);
  const a1 = 0.254829592;
  const a2 = -0.284496736;
  const a3 = 1.421413741;
  const a4 = -1.453152027;
  const a5 = 1.061405429;
  const p = 0.3275911;
  const t = 1 / (1 + p * absX);
  const y =
    1 -
     (((((a5 * t + a4) * t + a3) * t + a2) * t + a1) * t * Math.exp(-absX * absX));

  return sign * y;
};

const normalCdf = (z) => 0.5 * (1 + erf(z / Math.sqrt(2)));

const rankDifferences = (differences) => {
  const ranked = differences
    .map((diff) => ({ diff, abs: Math.abs(diff) }))
    .sort((a, b) => a.abs - b.abs);

  const tieCounts = {};
  let currentRank = 1;

  for (let i = 0; i < ranked.length; ) {
    let j = i;
       while (j + 1 < ranked.length && ranked[j + 1].abs === ranked[i].abs) {
      j += 1;
    }

    const avgRank = (currentRank + (j + 1)) / 2;
    for (let k = i; k <= j; k += 1) {
      ranked[k].rank = avgRank;
    }

    const tieSize = j - i + 1;
    if (tieSize > 1) {
      tieCounts[ranked[i].abs] = tieSize;
    }

    currentRank = j + 2;
    i = j + 1;
  }

  const WPositive = ranked.reduce(
    (sum, item) => (item.diff > 0 ? sum + item.rank : sum),
```

```
    0,
  );
  const WNegative = ranked.reduce(
    (sum, item) => (item.diff < 0 ? sum + item.rank : sum),
    0,
  );

  return { ranked, WPositive, WNegative, tieCounts };
};

export const wilcoxonTest = (pairs, metric) => {
  const direction = METRIC_DIRECTION[metric] ?? 1;
  const validPairs = (Array.isArray(pairs) ? pairs : []).filter(
    (pair) =>
                        Number.isFinite(pair?.es5?.[metric])    &&
Number.isFinite(pair?.es6?.[metric]),
  );

  const differences = validPairs
    .map((pair) => direction * (pair.es6[metric] - pair.es5[metric]))
    .filter((diff) => Number.isFinite(diff) && diff !== 0);

  const n = differences.length;
  const betterCount = differences.filter((d) => d > 0).length;
  const worseCount = differences.filter((d) => d < 0).length;

  if (n === 0) {
    return {
      W: 0,
      pValue: null,
      zScore: null,
      isSignificant: false,
      sampleSize: 0,
      direction: 0,
      betterCount: 0,
      worseCount: 0,
          interpretation: 'Немає даних для аналізу (всі різниці
дорівнюють нулю)',
    };
  }

  if (n < 5) {
    return {
      W: 0,
      pValue: null,
      zScore: null,
      isSignificant: false,
      sampleSize: n,
      direction: 0,
      betterCount,
      worseCount,
          interpretation: `Недостатньо пар для надійного тесту
(мінімум 5, знайдено ${n})`,
    };
  }

        const { WPositive, WNegative, tieCounts } =
rankDifferences(differences);
  const W = Math.min(WPositive, WNegative);

  const meanW = (n * (n + 1)) / 4;
  let varianceW = (n * (n + 1) * (2 * n + 1)) / 24;

  Object.values(tieCounts).forEach((t) => {
    varianceW -= (t * (t - 1) * (t + 1)) / 48;
  });

  const sdW = Math.sqrt(Math.max(varianceW, 0));
   const zScore = sdW === 0 ? 0 : (W - meanW - 0.5) / sdW; //
continuity correction
  const pValue = 2 * (1 - normalCdf(Math.abs(zScore)));
  const isSignificant = pValue < 0.05;
   const effectDirection = WPositive > WNegative ? 1 : WNegative
> WPositive ? -1 : 0;
  const effectSize = Math.abs(zScore) / Math.sqrt(n);

  const interpretation = isSignificant
    ? 'Різниця статистично значуща (p < 0.05)'
        : 'Статистично значущих відмінностей не виявлено (p ≥
0.05)';

  return {
    W,
    pValue,
```

```
    zScore,
    isSignificant,
    sampleSize: n,
    direction: effectDirection,
    betterCount,
    worseCount,
    effectSize,
    interpretation,
  };
};
```

src/layouts/layout.pug

```
pug
doctype html
html(lang="uk")
  head
    meta(charset="utf-8")
                meta(name="viewport"    content="width=device-width,
initial-scale=1")
    title
      block title
        | Code Complexity Lab
                                                link(rel="stylesheet",
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstra
p.min.css")
                                                link(rel="stylesheet",
href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.11.3/font/boo
tstrap-icons.min.css")
    // Prism.js для підсвітки синтаксису
                                                link(rel="stylesheet",
href="https://cdnjs.cloudflare.com/ajax/libs/prism/1.29.0/themes/pri
sm-tomorrow.min.css")
    link(rel="stylesheet", href="/css/app.css")

script(src="https://cdnjs.cloudflare.com/ajax/libs/prism/1.29.0/pris
m.min.js")

script(src="https://cdnjs.cloudflare.com/ajax/libs/prism/1.29.0/com
ponents/prism-javascript.min.js")
  body
    include ../partials/nav.pug
    main.container-fluid.py-4
      block content

script(src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/boo
tstrap.bundle.min.js")
    // Toast контейнер
        .toast-container.position-fixed.top-0.end-0.p-3(style="z-index:
1100")

#toast-mi.toast.align-items-center.text-bg-warning.border-0(role="a
lert" aria-live="assertive" aria-atomic="true")
        .d-flex
          .toast-body
            strong.d-block.mb-1 MI занизький!
            span#toast-mi-text.small
                                button.btn-close.me-2.m-auto(type="button"
data-bs-dismiss="toast" aria-label="Close")

#toast-cc.toast.align-items-center.text-bg-danger.border-0(role="ale
rt" aria-live="assertive" aria-atomic="true")
        .d-flex
          .toast-body
            strong.d-block.mb-1 CC зависока!
            span#toast-cc-text.small
                                button.btn-close.me-2.m-auto(type="button"
data-bs-dismiss="toast" aria-label="Close")
    block scripts
```

src/pages/comparison.pug

```
pug
extends ../layouts/layout.pug

block title
  | ES5 vs ES6: Порівняння · Code Complexity Lab

block content
  .container-fluid.py-4(style="max-width: 1400px")
```

```
// Header
.text-center.mb-5
  .display-1.mb-3
    i.bi.bi-arrow-left-right.text-primary
  h1.display-6.fw-bold ES5 vs ES6: Що змінилось?
    p.lead.text-muted Порівняння стандартів JavaScript та їх вплив на якість коду


// Tabs
.card.rounded-4.shadow.mb-5
  .card-body.p-4
    h3.h5.mb-4
      i.bi.bi-code-square.me-2
      | Синтаксис: Side-by-Side

    ul.nav.nav-tabs.mb-4#comparisonTabs(role="tablist")
      li.nav-item(role="presentation")
                  button.nav-link.active(data-bs-toggle="tab" data-bs-target="#tab-variables" type="button") Змінні
      li.nav-item(role="presentation")
                  button.nav-link(data-bs-toggle="tab" data-bs-target="#tab-functions" type="button") Функції
      li.nav-item(role="presentation")
                  button.nav-link(data-bs-toggle="tab" data-bs-target="#tab-arrays" type="button") Масиви
      li.nav-item(role="presentation")
                  button.nav-link(data-bs-toggle="tab" data-bs-target="#tab-strings" type="button") Рядки
      li.nav-item(role="presentation")
                  button.nav-link(data-bs-toggle="tab" data-bs-target="#tab-destructuring" type="button") Деструктуризація
      li.nav-item(role="presentation")
                  button.nav-link(data-bs-toggle="tab" data-bs-target="#tab-spread" type="button") Spread/Rest

    .tab-content#comparisonTabsContent
      // Variables
      .tab-pane.fade.show.active#tab-variables(role="tabpanel")
        .row.g-3
          .col-md-6
            .card.border-danger.h-100
              .card-header.bg-danger.text-white
                strong ES5
                | — var (function scope, hoisting)
              .card-body.p-0
                pre.mb-0
                  code.language-javascript.
                    var name = 'John';
                    var age = 25;

                    if (true) {
                      var temp = 'test';
                    }
                    console.log(temp); // 'test'
          .col-md-6
            .card.border-success.h-100
              .card-header.bg-success.text-white
                strong ES6
                | — const/let (block scope)
              .card-body.p-0
                pre.mb-0
                  code.language-javascript.
                    const name = 'John';
                    let age = 25;

                    if (true) {
                      const temp = 'test';
                    }
                    // console.log(temp); // ReferenceError

      // Functions
      .tab-pane.fade#tab-functions(role="tabpanel")
        .row.g-3
          .col-md-6
            .card.border-danger.h-100
              .card-header.bg-danger.text-white
                strong ES5
                | — function keyword
              .card-body.p-0
                pre.mb-0
                  code.language-javascript.
                    var add = function(a, b) {
```

```
          return a + b;
        };

        users.map(function(user) {
          return user.name;
        });

        var self = this;
        setTimeout(function() {
          console.log(self.value);
        }, 1000);
    .col-md-6
      .card.border-success.h-100
        .card-header.bg-success.text-white
          strong ES6
          | — arrow functions
        .card-body.p-0
          pre.mb-0
            code.language-javascript.
              const add = (a, b) => a + b;

              users.map((user) => user.name);

              setTimeout(() => {
                console.log(this.value);
              }, 1000);
// Arrays
.tab-pane.fade#tab-arrays(role="tabpanel")
  .row.g-3
    .col-md-6
      .card.border-danger.h-100
        .card-header.bg-danger.text-white
          strong ES5
          | — ручні цикли
        .card-body.p-0
          pre.mb-0
            code.language-javascript.
              var adults = [];
              for (var i = 0; i &#60; users.length; i++) {
                if (users[i].age &#62;= 18) {
                  adults.push(users[i]);
                }
              }

              var names = [];
              for (var j = 0; j &#60; adults.length; j++) {
                names.push(adults[j].name);
              }
      .col-md-6
        .card.border-success.h-100
          .card-header.bg-success.text-white
            strong ES6
            | — map/filter/reduce
          .card-body.p-0
            pre.mb-0
              code.language-javascript.
                const adults = users.filter((user) => user.age &#62;=
18);

                const names = adults.map((user) => user.name);

// Strings
.tab-pane.fade#tab-strings(role="tabpanel")
  .row.g-3
    .col-md-6
      .card.border-danger.h-100
        .card-header.bg-danger.text-white
          strong ES5
          | — конкатенація через +
        .card-body.p-0
          pre.mb-0
            code.language-javascript.
              var greeting = 'Hello, ' + name + '!';
                var message = 'User ' + user.name + ' (' + user.email
+ ')';

              var html = '&#60;div class="user">' +
                '  &#60;h2>' + name + '&#60;/h2>' +
                '&#60;/div>';
    .col-md-6
      .card.border-success.h-100
        .card-header.bg-success.text-white
          strong ES6
          | — template literals
        .card-body.p-0
```

```
            pre.mb-0
              code.language-javascript.
                const greeting = `Hello, ${name}!`;
                                const message = `User ${user.name}
(${user.email})`;
                const html = `
                  &#60;div class="user">
                    &#60;h2>${name}&#60;/h2>
                  &#60;/div>`;


        // Destructuring
        .tab-pane.fade#tab-destructuring(role="tabpanel")
          .row.g-3
            .col-md-6
              .card.border-danger.h-100
                .card-header.bg-danger.text-white
                  strong ES5
                  | — повторення
                .card-body.p-0
                  pre.mb-0
                    code.language-javascript.
                      var name = user.name;
                      var email = user.email;
                      var age = user.age;
                      var role = user.role;

                      var first = arr[0];
                      var second = arr[1];
                      var rest = arr.slice(2);
            .col-md-6
              .card.border-success.h-100
                .card-header.bg-success.text-white
                  strong ES6
                  | — деструктуризація
                .card-body.p-0
                  pre.mb-0
                    code.language-javascript.
                      const { name, email, age, role } = user;
                      const [first, second, ...rest] = arr;


        // Spread/Rest
        .tab-pane.fade#tab-spread(role="tabpanel")
          .row.g-3
            .col-md-6
              .card.border-danger.h-100
                .card-header.bg-danger.text-white
                  strong ES5
                  | — складний синтаксис
                .card-body.p-0
                  pre.mb-0
                    code.language-javascript.
                      var arr2 = arr1.concat([4, 5, 6]);
                      var copy = arr.slice();
                      function sum() {
                        var args = Array.prototype.slice.call(arguments);
                        return args.reduce(function(acc, value) {
                          return acc + value;
                        }, 0);
                      }
            .col-md-6
              .card.border-success.h-100
                .card-header.bg-success.text-white
                  strong ES6
                  | — spread/rest
                .card-body.p-0
                  pre.mb-0
                    code.language-javascript.
                      const arr2 = [...arr1, 4, 5, 6];
                      const copy = [...arr];
                        const sum = (...args) => args.reduce((acc, value) =>
acc + value, 0);



        // Why important
        .card.rounded-4.shadow.mb-5
          .card-body.p-4
            h3.h5.mb-4
              i.bi.bi-lightbulb.me-2
              | Чому це важливо?
            .row.g-4
              .col-md-6.col-lg-3
                .text-center
                  .display-1.text-primary.mb-3
                    i.bi.bi-eye
```

```
          h6 Читабельність
            p.small.text-muted Менше boilerplate та зрозуміліший intent.
        .col-md-6.col-lg-3
          .text-center
            .display-1.text-success.mb-3
              i.bi.bi-shield-check
            h6 Підтримуваність
                p.small.text-muted Менше коду = менше помилок, легший code review.
        .col-md-6.col-lg-3
          .text-center
            .display-1.text-warning.mb-3
              i.bi.bi-speedometer2
            h6 Продуктивність
                    p.small.text-muted Map/filter використовують оптимізації JS engine.
        .col-md-6.col-lg-3
          .text-center
            .display-1.text-info.mb-3
              i.bi.bi-people
            h6 Командна робота
              p.small.text-muted Єдиний стиль = швидше онбординг та рев'ю.


block scripts
  script.
    (function() {
      const highlight = () => {
        if (window.Prism && typeof window.Prism.highlightAll === 'function') {
          window.Prism.highlightAll();
        }
      };
      highlight();

      const PREFILL_KEY = 'ccl:prefill-code';
      const loadSample = (path) => {
        fetch(path)
          .then((res) => res.text())
          .then((code) => {
            try {
              sessionStorage.setItem(PREFILL_KEY, code);
            } catch (error) {
                    console.warn('Не вдалося зберегти код для автозаповнення.', error);
            }
            window.location.href = '/#analyze';
          })
          .catch((error) => console.error('Не вдалося завантажити приклад:', error));
      };

                    window.loadES5Code = () => loadSample('/presentation_example/user_management_es5.js');
                    window.loadES6Code = () => loadSample('/presentation_example/user_management_es6.js');
    })();



src/pages/docs/cognitive.pug

pug
extends ../../layouts/layout.pug

block title
  | Cognitive Complexity · Документація

block content
  .container.py-4(style="max-width: 900px")
    // Header
    .text-center.mb-5
      .display-1.mb-3
        i.bi.bi-brain.text-primary
      h1.display-6.fw-bold Cognitive Complexity
      p.lead.text-muted Когнітивна складність коду

    // TL;DR
    .alert.alert-primary.d-flex.align-items-start.gap-3.mb-5
      i.bi.bi-lightbulb.fs-3
      div
        h5.alert-heading Що це таке?
```

```
      p.mb-0
        |
        span.badge.bg-primary-subtle.text-dark.em-75
        | CC
        |  вимірює наскільки
        strong складно зрозуміти
                |  код при читанні. Враховує не лише кількість розгалужень, а й
        strong вкладеність
        |  та довгі логічні вирази.

    // Шкала
    .card.rounded-4.shadow.mb-5
      .card-body.p-4
        h3.h5.mb-4
          i.bi.bi-speedometer2.me-2
          | Шкала оцінювання

              .position-relative.mb-4(style="height: 40px; background: linear-gradient(to right, #38ef7d 0%, #f2994a 50%, #eb3349 80%); border-radius: 20px")
          .position-absolute.top-50.start-0.translate-middle-y.ms-2
            span.badge.bg-dark 0
                .position-absolute.top-50.translate-middle-y(style="left: 40%")
            span.badge.bg-dark 10
                .position-absolute.top-50.translate-middle-y(style="left: 65%")
            span.badge.bg-dark 20
          .position-absolute.top-50.end-0.translate-middle-y.me-2
            span.badge.bg-dark 30+

        .row.g-3
          .col-md-4
            .d-flex.align-items-center.gap-2
                  .badge.bg-success.rounded-circle(style="width: 12px; height: 12px")
              div
                strong &le; 10:
                | Добре
          .col-md-4
            .d-flex.align-items-center.gap-2
                  .badge.bg-warning.rounded-circle(style="width: 12px; height: 12px")
              div
                strong 11-20:
                | Прийнятно
          .col-md-4
            .d-flex.align-items-center.gap-2
                  .badge.bg-danger.rounded-circle(style="width: 12px; height: 12px")
              div
                strong &gt; 20:
                | Потребує рефакторингу

    // Правила підрахунку
    .card.rounded-4.shadow.mb-5
      .card-body.p-4
        h3.h5.mb-4
          i.bi.bi-list-check.me-2
          | Що додає складність?

        .row.g-4
          .col-md-6
            .card.border-primary
              .card-body
                h6.text-primary
                  i.bi.bi-plus-circle.me-1
                  | +1 за структуру
                ul.small.mb-0
                  li
                    code if, else if, else
                  li
                    code for, while, do-while
                  li
                    code switch
                  li
                    code catch
                  li
                    code ? :
                    |  (тернарний)

          .col-md-6
            .card.border-warning
```

```
          .card-body
            h6.text-warning
              i.bi.bi-layers.me-1
              | +N за вкладеність
            p.small.mb-0
              | Кожен рівень вкладеності додає
              strong +1
              | до базового інкременту.
            .bg-light.p-2.rounded.mt-2
              code.small.
                if (...) {        // +1
                  if (...) {      // +2 (nesting)
                    for (...) {   // +3 (nesting)

        button.btn.btn-sm.btn-outline-secondary.mt-3(type="button" data-bs-toggle="collapse" data-bs-target="#cc-extra")
          i.bi.bi-info-circle.me-1
          | Детально про специфікацію SonarSource
        #cc-extra.collapse.mt-3
          .bg-light.border.rounded-3.p-3
            p.small.mb-2
              | Наша реалізація слідує специфікації SonarSource v1.7: структури отримують
              code +1
              | , а кожна глибина додає ще
              code +1
                    | . Рекурсію наразі позначаємо як примітку, без автоматичного інкременту.
            ul.small.mb-0
              li(data-bs-toggle="tooltip" title="switch легший за довгі if")
                strong switch
                | коштує 1 незалежно від кількості
                code case
              li(data-bs-toggle="tooltip" title="Рекурсію можна додати окремим проходом")
                    | Якщо метод бере участь у рекурсії — додаємо фундаментальний
                code +1
                | вручну
                      li(data-bs-toggle="tooltip" title="Коли операнди змішуються")
                        | Зміна
                        code && ↔ ||
                        | стартує нову послідовність і дає ще
                        code +1


// Приклад
.card.rounded-4.shadow.mb-5
  .card-body.p-4
    h3.h5.mb-3
      i.bi.bi-code-slash.me-2
      | Приклад підрахунку


    .row.g-3
      .col-md-6
        h6.text-danger
          | CC = 6
          small.text-muted (складно)
        pre.bg-light.p-3.rounded.small
          code.
            function check(x, y) {
              if (x &#62; 0) {          // +1
                if (y &#62; 0) {        // +2 (base +1, nest +1)
                  return true;
                }
              }
              for (let i = 0; ...) { // +1
                if (i % 2) {        // +2 (base +1, nest +1)
                  break;
                }
              }
            }
            // Всього: 1 + 2 + 1 + 2 = 6

      .col-md-6
        h6.text-success
          | CC = 2
          small.text-muted (просто)
        pre.bg-light.p-3.rounded.small
          code.
            const isPositive = (n) =&#62; n &#62; 0;

            const check = (x, y) =&#62;
```

```
            isPositive(x) &&      // +1 (&&)
            isPositive(y);        // +1 (&&)

          // Всього: 1 + 1 = 2


  // Відмінності від Cyclomatic
  .card.rounded-4.shadow.bg-light.mb-5
    .card-body.p-4
      h3.h6.mb-3
        i.bi.bi-question-circle.me-2
        | Cognitive vs Cyclomatic — в чому різниця?

      table.table.table-sm.mb-0
        thead
          tr
            th
            th.text-center Cognitive
            th.text-center Cyclomatic
        tbody
          tr
                td(data-bs-toggle="tooltip" title="Штраф за вкладені
блоки") Враховує вкладеність
            td.text-center
              i.bi.bi-check-circle-fill.text-success
            td.text-center
              i.bi.bi-x-circle-fill.text-danger
          tr
                  td(data-bs-toggle="tooltip" title="Довгі ланцюжки
&&/||") Штрафує за довгі логічні вирази
            td.text-center
              i.bi.bi-check-circle-fill.text-success
            td.text-center
              i.bi.bi-dash-circle-fill.text-muted


  // Посилання
  .card.rounded-4.shadow.bg-light
    .card-body.p-4
      h3.h6.mb-3 Дивіться також:
      .d-flex.gap-2.flex-wrap
        a.btn.btn-sm.btn-outline-primary(href="/docs/mi")
          i.bi.bi-gear.me-1
          | Maintainability Index
        a.btn.btn-sm.btn-outline-primary(href="/")
          i.bi.bi-house.me-1
          | На головну
```

src/pages/docs/index.pug

```
pug
extends ../../layouts/layout.pug

block title
  | Документація · Code Complexity Lab

block content
  h1.h4 Документація
    p.text-muted Тут зібрані пояснення метрик та приклади
покрокового обчислення.
  .list-group

a.list-group-item.list-group-item-action(href="/docs/cognitive.html"
)
      strong Когнітивна складність (Cognitive Complexity)
      | — як рахується та як зменшити
    a.list-group-item.list-group-item-action(href="/docs/mi.html")
      strong Індекс підтримуваності (Maintainability Index)
      | — формула, Halstead, цикломатика, LOC
```

src/pages/docs/mi.pug

```
pug
extends ../../layouts/layout.pug

block title
  | Maintainability Index · Документація

block content
  .container.py-4(style="max-width: 900px")
    // Header з іконкою
    .text-center.mb-5
```

```
.display-1.mb-3
  i.bi.bi-gear-fill.text-success
 h1.display-6.fw-bold Maintainability Index
 p.lead.text-muted Індекс підтримуваності коду

// Швидке пояснення (TL;DR)
.alert.alert-success.d-flex.align-items-start.gap-3.mb-5
 i.bi.bi-lightbulb.fs-3
 div
  h5.alert-heading Що це таке?
  p.mb-0
   | MI показує наскільки
   strong легко підтримувати
   |  ваш код. Чим вище значення — тим краще.
   | Враховує обсяг коду, складність та структуру.

// Шкала з візуалізацією
.card.rounded-4.shadow.mb-5
 .card-body.p-4
  h3.h5.mb-4
   i.bi.bi-speedometer2.me-2
   | Шкала оцінювання

  // Пояснення рівнів
  .row.g-3
   .col-md-4
    .d-flex.align-items-center.gap-2
     .badge.bg-danger.rounded-circle(style="width: 12px; height: 12px")
     div
      strong 0-9:
      | Низька підтримуваність
   .col-md-4
    .d-flex.align-items-center.gap-2
     .badge.bg-warning.rounded-circle(style="width: 12px; height: 12px")
     div
      strong 10-19:
      | Середня
   .col-md-4
    .d-flex.align-items-center.gap-2
     .badge.bg-success.rounded-circle(style="width: 12px; height: 12px")
     div
      strong 20-100:
      | Висока

 // Як рахується (collapse)
 .card.rounded-4.shadow.mb-5
  .card-body.p-4
   h3.h5.mb-3
    i.bi.bi-calculator.me-2
    | Як це працює?

   p.mb-3
    |

span.badge.bg-secondary-subtle.text-dark.em-75(data-bs-toggle="tooltip" title="Maintainability Index")
     | MI
     |  обчислюється за формулою Microsoft Visual Studio (нормована до 0-100):

   .bg-light.p-3.rounded.mb-3
    code.text-dark.small
     | MI = max(0, (171 − 5.2·ln(V) − 0.23·G − 16.2·ln(LOC)) · 100 / 171)

   // Accordion з деталями
   #mi-formula-accordion.accordion
    // Halstead Volume
    .accordion-item
     h2.accordion-header
      button.accordion-button.collapsed(
       type="button"
       data-bs-toggle="collapse"
       data-bs-target="#collapse-v"
      )
       strong(data-bs-toggle="tooltip" title="Обсяг програми через оператори/операнди") V
       | — Halstead Volume (обсяг програми)

#collapse-v.accordion-collapse.collapse(data-bs-parent="#mi-formula-accordion")
```

```
        .accordion-body
          p
                    | Вимірює "розмір" програми через токени (оператори та операнди).
          ul.mb-0
            li
              code n1
              | — унікальні оператори (
              code if, +, =
              | )
            li
              code n2
              | — унікальні операнди (змінні, літерали)
            li
              code N1, N2
              | — загальна кількість
            li
              strong V = (N1 + N2) · log₂(n1 + n2)

      // Cyclomatic
      .accordion-item
        h2.accordion-header
          button.accordion-button.collapsed(
            type="button"
            data-bs-toggle="collapse"
            data-bs-target="#collapse-g"
          )
                    strong(data-bs-toggle="tooltip"  title="Класична цикломатична складність McCabe") G
          | — Cyclomatic Complexity (розгалуження)

#collapse-g.accordion-collapse.collapse(data-bs-parent="#mi-formula-accordion")
          .accordion-body
            p
              | Рахує кількість точок прийняття рішень у коді.
            ul.mb-0
              li
                code if/else
                | → +1
              li
                code for/while
                | → +1
              li
                code &&/||
                | → +1
              li
                code case
                | в switch → +1

      // LOC
      .accordion-item
        h2.accordion-header
          button.accordion-button.collapsed(
            type="button"
            data-bs-toggle="collapse"
            data-bs-target="#collapse-loc"
          )
            strong(data-bs-toggle="tooltip" title="Lines of Code без порожніх рядків") LOC
          | — Lines of Code (рядки коду)

#collapse-loc.accordion-collapse.collapse(data-bs-parent="#mi-formula-accordion")
          .accordion-body
            p.mb-0
                | Кількість непорожніх рядків коду (без коментарів та порожніх рядків).

    // Як покращити MI
    .card.rounded-4.shadow.border-success.mb-5
      .card-body.p-4
        h3.h5.mb-3
          i.bi.bi-arrow-up-circle.text-success.me-2
          | Як покращити MI?

        .row.g-3
          .col-md-6
            .d-flex.gap-2
              i.bi.bi-check-circle-fill.text-success
              div
                strong Розбивайте функції
                p.small.text-muted.mb-0 Менші функції = вищий MI
```

```
        .col-md-6
          .d-flex.gap-2
            i.bi.bi-check-circle-fill.text-success
            div
              strong Спрощуйте логіку
                    p.small.text-muted.mb-0 Менше if/for = менша
складність

        .col-md-6
          .d-flex.gap-2
            i.bi.bi-check-circle-fill.text-success
            div
              strong Видаляйте дублювання
              p.small.text-muted.mb-0 Менше коду = кращий MI

        .col-md-6
          .d-flex.gap-2
            i.bi.bi-check-circle-fill.text-success
            div
              strong Використовуйте ES6+
                    p.small.text-muted.mb-0 Сучасний синтаксис =
читабельніший код

    // Посилання на інші метрики
    .card.rounded-4.shadow.bg-light
      .card-body.p-4
        h3.h6.mb-3 Дивіться також:
        .d-flex.gap-2.flex-wrap
          a.btn.btn-sm.btn-outline-primary(href="/docs/cognitive")
            i.bi.bi-brain.me-1
            | Cognitive Complexity
          a.btn.btn-sm.btn-outline-primary(href="/")
            i.bi.bi-house.me-1
            | На головну



src/pages/index.pug

pug
extends ../layouts/layout.pug

block title
  | Головна · Code Complexity Lab

block content
  // hero-зона: компактне введення та миттєвий аналіз

.container.d-flex.flex-column.align-items-center.justify-content-cent
er.min-vh-100.py-5
    // Лого+заголовок
    .text-center.mb-4
      h1.display-6.fw-semibold.mb-2 Code Complexity Lab
       p.text-muted.mb-0 Вставте JavaScript-код і отримайте оцінку
читабельності та підтримуваності

    // Центральний інпут (компактний)
              .card.shadow.border.rounded-4.w-100(style="max-width:
1200px")
      .card-body.p-4
        .input-group.input-group-lg
          span.input-group-text.border-0.bg-light
            i.bi.bi-code-square

textarea#code.form-control.border-0.rounded-end.shadow-sm.font-
monospace(
            rows="3"
              placeholder="Вставте JavaScript-код сюди… (Ctrl+Enter
для аналізу)"
            style="resize: vertical; min-height: 100px"
          )

        .d-flex.gap-2.mt-3.flex-wrap.align-items-center

button#analyze.btn.btn-primary.btn-lg.flex-grow-1.d-flex.align-ite
ms-center.justify-content-center.gap-2(type="button")

span#analyze-spinner.spinner-border.spinner-border-sm.d-none(role
="status" aria-hidden="true")
            i.bi.bi-lightning-charge
            span#analyze-text Аналізувати код

          .btn-group(role="group")
```

```
button#sample.btn.btn-outline-secondary.btn-lg(type="button"
data-bs-toggle="tooltip" title="Завантажити приклад коду")
          i.bi.bi-file-code

button#upload.btn.btn-outline-secondary.btn-lg(type="button"
data-bs-toggle="tooltip" title="Завантажити файл .js")
          i.bi.bi-upload
                                    input#file-input.d-none(type="file"
accept=".js,.jsx,.ts,.tsx")
            button#clear.btn.btn-outline-danger.btn-lg(type="button"
data-bs-toggle="tooltip" title="Очистити поле")
          i.bi.bi-trash

      .mt-2
        small#input-status.text-muted.d-flex.align-items-center.gap-2
          i.bi.bi-info-circle
          span Готово до аналізу

  // Прев'ю коду (показується після аналізу)
    #code-preview.mt-4.d-none.w-100(style="max-width:  1200px;
margin: 0 auto")
    .card.code-preview-card.rounded-4.border-0.shadow

.card-header.code-preview-header.d-flex.justify-content-between.ali
gn-items-center.px-4.py-3
        .d-flex.align-items-center.gap-2
          i.bi.bi-code-slash.text-white-50
          h3.h6.mb-0.text-white Проаналізований код

span.badge.bg-white.bg-opacity-10.text-white-50.small#code-lines
0 рядків

        .btn-group(role="group")

button.btn.btn-sm.btn-outline-light.btn-icon#copy-code(type="butto
n"          data-bs-toggle="tooltip"          data-bs-placement="top"
title="Копіювати код")
          i.bi.bi-clipboard

button.btn.btn-sm.btn-outline-light.btn-icon#fullscreen-code(type="
button"        data-bs-toggle="tooltip"        data-bs-placement="top"
title="Повноекранний режим")
            i.bi.bi-arrows-fullscreen

      .card-body.code-preview-body.p-0
        pre.mb-0
          code#code-display.language-javascript

  // Блок результатів — миттєві метрики
    .section-results.w-100.mt-5(style="max-width: 1200px; margin:
0 auto")
      .row.g-4.mb-4
        .col-12.col-lg-6
          .card.metric-card.mi-card.rounded-4.border-0.shadow.h-100
            .card-body.p-4
              .d-flex.justify-content-between.align-items-start.mb-3
                div
                        .text-uppercase.fw-semibold.small.text-white.mb-1
Підтримуваність
                  h2.display-4.fw-bold.text-white.mb-0#mi-chip 0
                  p.text-white.small.mb-0 із 100
                .metric-icon
                  i.bi.bi-gear-fill

              .progress.mb-3(style="height: 8px")

.progress-bar.progress-bar-animated#mi-progress(role="progressbar
" style="width: 0%" aria-valuenow="0" aria-valuemin="0"
aria-valuemax="100")

              .d-flex.align-items-center.gap-2

span#mi-status-icon.badge.rounded-pill.px-3.bg-light.text-dark
Статус
                        span#mi-human-label.text-white.small  Статус:
Обчислюємо…

              .mt-3.pt-3.border-top.border-white-25
                .row.row-cols-2.g-3.small
                  .col

.d-flex.justify-content-between.text-white.align-items-center
                    span.d-inline-flex.align-items-center.gap-1
                      span Halstead V
```

```
                i.bi.bi-question-circle.opacity-75.cursor-pointer(
                  data-bs-toggle="popover"
                  data-bs-trigger="hover focus"
                  data-bs-placement="top"
                  data-bs-custom-class="metric-popover"
                  data-bs-title="Halstead Volume"
                   data-bs-content="Оцінює розмір програми через унікальні оператори й операнди. Високе значення означає насиченість конструкціями. Типово: 100-500 для функцій."
                )
              span.text-white.fw-semibold#mi-vol 0
            .col

.d-flex.justify-content-between.text-white.align-items-center
              span.d-inline-flex.align-items-center.gap-1
                span Cyclomatic
                i.bi.bi-question-circle.opacity-75.cursor-pointer(
                  data-bs-toggle="popover"
                  data-bs-trigger="hover focus"
                  data-bs-placement="top"
                  data-bs-custom-class="metric-popover"
                  data-bs-title="Cyclomatic Complexity (G)"
                     data-bs-content="Кількість незалежних шляхів через код. Рівна кількості тестів для повного покриття. Добре: ≤10, прийнятно: 11–20, критично: >20."
                )
              span.text-white.fw-semibold#mi-cyc 0
            .col

.d-flex.justify-content-between.text-white.align-items-center
              span.d-inline-flex.align-items-center.gap-1
                span LOC
                i.bi.bi-question-circle.opacity-75.cursor-pointer(
                  data-bs-toggle="popover"
                  data-bs-trigger="hover focus"
                  data-bs-placement="top"
                  data-bs-custom-class="metric-popover"
                  data-bs-title="Lines of Code"
                        data-bs-content="Логічні рядки коду без порожніх дужок і коментарів. Слугують базою для нормалізації інших метрик. Оптимально: 10-50 LOC для окремої функції."
                )
              span.text-white.fw-semibold#mi-loc 0
            .col

.d-flex.justify-content-between.text-white.align-items-center
              span.d-inline-flex.align-items-center.gap-1
                span Коментарі
                i.bi.bi-question-circle.opacity-75.cursor-pointer(
                  data-bs-toggle="popover"
                  data-bs-trigger="hover focus"
                  data-bs-placement="top"
                  data-bs-custom-class="metric-popover"
                  data-bs-title="Щільність коментарів"
                     data-bs-content="Частка рядків із коментарями відносно всіх рядків. Рекомендовано: 10–30%. Менше — код може бути неочевидним, занадто багато — сигнал складності."
                )
              span.text-white.fw-semibold#mi-comments 0%

      .col-12.col-lg-6
        .card.metric-card.cc-card.rounded-4.border-0.shadow.h-100
          .card-body.p-4
            .d-flex.justify-content-between.align-items-start.mb-3
              div
                    .text-uppercase.fw-semibold.small.text-white.mb-1 Когнітивна складність
                h2.display-4.fw-bold.text-white.mb-0#cc-chip 0
                p.text-white.small.mb-0 рівень складності
              .metric-icon
                i.bi.bi-lightbulb

            .progress.mb-3(style="height: 8px")

.progress-bar.progress-bar-striped.progress-bar-animated#cc-progress(role="progressbar" style="width: 0%" aria-valuenow="0" aria-valuemin="0" aria-valuemax="100")

            .d-flex.align-items-center.gap-2

span#cc-status-icon.badge.rounded-pill.px-3.bg-light.text-dark Рівень
                    span#cc-human-label.text-white.small Рівень: Обчислюємо…
```

```
.mt-3.pt-3.border-top.border-white-25
  .row.row-cols-3.g-3.small
    .col
      .d-flex.align-items-center.gap-2
        .metric-dot.bg-success
        div
          .text-white.d-flex.align-items-center.gap-1
            span Добре
            i.bi.bi-question-circle.opacity-75.cursor-pointer(
              data-bs-toggle="popover"
              data-bs-trigger="hover focus"
              data-bs-placement="top"
              data-bs-custom-class="metric-popover"
              data-bs-title="Низька когнітивна складність"
                    data-bs-content="Код легко читати й утримувати в голові. Вкладеність контрольована, розгалуження прості."
            )
          .text-white.fw-semibold ≤10
    .col
      .d-flex.align-items-center.gap-2
        .metric-dot.bg-warning
        div
          .text-white.d-flex.align-items-center.gap-1
            span Прийнятно
            i.bi.bi-question-circle.opacity-75.cursor-pointer(
              data-bs-toggle="popover"
              data-bs-trigger="hover focus"
              data-bs-placement="top"
              data-bs-custom-class="metric-popover"
              data-bs-title="Підвищена складність"
                    data-bs-content="Код ще читабельний, але варто уникати додаткової вкладеності та довгих умов."
            )
          .text-white.fw-semibold 11–20
    .col
      .d-flex.align-items-center.gap-2
        .metric-dot.bg-danger
        div
          .text-white.d-flex.align-items-center.gap-1
            span Високий
            i.bi.bi-question-circle.opacity-75.cursor-pointer(
              data-bs-toggle="popover"
              data-bs-trigger="hover focus"
              data-bs-placement="top"
              data-bs-custom-class="metric-popover"
              data-bs-title="Висока когнітивна складність"
                    data-bs-content="Код важко читати через глибоку вкладеність та складні логічні ланцюжки. Рекомендовано рефакторинг."
            )
          .text-white.fw-semibold 20+

  // Історія аналізів
  #history-section.w-100.mt-4.d-none(style="max-width: 1200px; margin: 0 auto")
    .card.rounded-4.border-0.shadow

      .card-header.bg-light.d-flex.justify-content-between.align-items-center.px-4.py-3
        .d-flex.align-items-center.gap-2
          i.bi.bi-clock-history
          h3.h6.mb-0 Історія аналізів
          span.badge.bg-secondary.rounded-pill#history-count 0

        button.btn.btn-sm.btn-outline-danger#clear-history(type="button")
          i.bi.bi-trash.me-1
          | Очистити
      .card-body.p-3
        #history-list.d-flex.flex-column.gap-3
          .text-center.text-muted.py-4#history-empty
            i.bi.bi-inbox.fs-2.d-block.mb-2
                  p.mb-0 Історія порожня. Проаналізуйте код щоб почати.
    #tips-section.d-none

      .alert.alert-warning.rounded-4.border-0.d-flex.gap-3.align-items-start(role="alert")
        i.bi.bi-lightbulb.fs-4.flex-shrink-0
        div
          h4.alert-heading.h6.mb-2 Рекомендації
          ul.mb-0.small#tips-list
```

```
block scripts
  script(type="module" src="/js/main.js")



src/pages/stats.pug

pug
extends ../layouts/layout.pug

block title
  | Статистика ES5 vs ES6 — Code Complexity Lab

block content
  .container-fluid.py-4
    .text-center.mb-5
      h1.display-5.fw-bold Порівняння ES5 vs ES6
      p.lead.text-muted
        | Аналіз метрик складності для
        span#file-pair-count 0
        |  пар файлів з однаковою бізнес-логікою
        br
        small#file-total-count.text-muted Опрацьовано 0 файлів



#stats-status.alert.alert-info.d-flex.align-items-center.gap-2(role="alert")
      i.bi.bi-cloud-arrow-down-fill
      span Оновлюємо дані…

    #summary-cards.row.g-4.mb-5

    section.statistical-analysis.mt-5
      h2.h4 📊 Статистична значущість
      p.small.text-muted Імовірність того, що різниця випадкова
          p.text-muted#wilcoxon-sample-info Тест Вілкоксона для парних вибірок

      .row.g-4#wilcoxon-cards
        .col-12
          .text-muted.small Дані завантажуються...
        .mt-4
          h5.mb-2 Що означає p-value?
          p.text-muted.mb-0
            | p &lt; 0.05 означає, що з 95% впевненістю різниця не є випадковою.
            br
            | Наші результати показують p &lt; 0.05 для всіх ключових показників.
          details.mt-2
            summary.text-primary.cursor-pointer.small Як працює тест Вілкоксона?
            .mt-2.small.text-muted
              ol.mb-0.ps-3
                li Для кожної пари файлів рахується різниця (ES6 - ES5)
                li Різниці сортуються за абсолютним значенням і ранжуються
                li Ранги сумуються окремо для позитивних та негативних різниць
                li Обчислюється статистика W та p-value через Z-апроксимацію
                li Якщо p &lt; 0.05 → різниця статистично значуща ✅

    .row.g-4.mb-5
      .col-12
        h2.h4.mb-3 Графіки порівняння

      .col-12
        .card.rounded-4.shadow.h-100
          .card-body
            h3.h5.mb-3
              i.bi.bi-gear.text-success.me-2
              | Maintainability Index (Індекс підтримуваності)
            #chart-mi.chart-container(style="min-height: 400px")

      .col-12
        .card.rounded-4.shadow.h-100
          .card-body
            h3.h5.mb-3
              i.bi.bi-brain.text-primary.me-2
              | Cognitive Complexity (Когнітивна складність)
            #chart-cc.chart-container(style="min-height: 400px")
```

```
    .col-12
      .card.rounded-4.shadow.h-100
        .card-body
          h3.h5.mb-3
            i.bi.bi-diagram-3.text-warning.me-2
            | Cyclomatic Complexity (Цикломатична складність)
                #chart-cyclomatic.chart-container(style="min-height:
400px")

    .col-12
      .card.rounded-4.shadow.h-100
        .card-body
          h3.h5.mb-3
            i.bi.bi-file-code.text-info.me-2
            | Lines of Code (Кількість рядків коду)
          #chart-loc.chart-container(style="min-height: 400px")

  .card.rounded-4.shadow.mb-5
    .card-body

.d-flex.flex-wrap.align-items-center.justify-content-between.gap-3.
mb-4
        h3.h5.mb-0
          i.bi.bi-table.me-2
          | Детальне порівняння файлів

button#export-csv-btn.btn.btn-outline-primary.btn-sm(type="button
" disabled)
          i.bi.bi-download.me-2
          | Експорт у CSV

      .alert.alert-info.d-flex.align-items-start.gap-3.mb-4
        i.bi.bi-info-circle.fs-5
        div
          strong Як читати таблицю:
          ul.mb-0.mt-2.small
            li
              strong MI (Maintainability Index):
              |  Вище = краще.
              span.badge.bg-success.mx-1 20-100
              | високий,
              span.badge.bg-warning.text-dark.mx-1 10-19
              | середній,
              span.badge.bg-danger.mx-1 0-9
              | низький
            li
              strong CC (Cognitive Complexity):
              |  Нижче = краще.
              span.badge.bg-success.mx-1 ≤10
              | добре,
              span.badge.bg-warning.text-dark.mx-1 11-20
              | прийнятно,
              span.badge.bg-danger.mx-1 >20
              | погано
            li
              strong Cyclomatic Complexity:
              |  Нижче = краще. Рекомендовано
              span.badge.bg-success.mx-1 ≤10
            li
              strong LOC (Lines of Code):
              |  Нижче = краще (менше коду = простіше підтримка)
            li
              i.bi.bi-arrow-up.text-success
              |  покращення,
              i.bi.bi-arrow-down.text-danger.ms-2
              |  погіршення,
              i.bi.bi-dash.text-muted.ms-2
              |  без змін

.table-responsive
  table.table.table-hover.align-middle#comparison-table
    thead.table-light
      tr
        th(style="width: 20%") Файл
        th.text-center(style="width: 20%")
          | MI
          br
          small.text-muted (вище = краще)
        th.text-center(style="width: 20%")
          | Cognitive
          br
          small.text-muted (нижче = краще)
        th.text-center(style="width: 20%")
```

```
            | Cyclomatic
            br
            small.text-muted (нижче = краще)
          th.text-center(style="width: 20%")
            | LOC
            br
            small.text-muted (нижче = краще)
        tbody
          tr
                td(colspan="5").text-center.text-muted.small Дані
завантажуються...

  .row.g-4.mb-5
    .col-12
      h2.h4.mb-3 Статистичний аналіз

    .col-12.col-lg-6
      .card.rounded-4.shadow
        .card-body
          h5.card-title ES5 — Статистика
          #stats-es5-table.text-muted.small Дані завантажуються...

    .col-12.col-lg-6
      .card.rounded-4.shadow
        .card-body
          h5.card-title ES6 — Статистика
          #stats-es6-table.text-muted.small Дані завантажуються...

block scripts
  script(src="https://code.highcharts.com/highcharts.js")
  script(src="https://code.highcharts.com/modules/exporting.js")
  script(type="module", src="/js/stats.js")
```

src/partials/nav.pug

```
pug
nav.navbar.navbar-expand-lg.navbar-dark.bg-dark
  .container-fluid
    a.navbar-brand(href="/")
      i.bi.bi-code-square.me-2
      | Code Complexity Lab
      button.navbar-toggler(type="button" data-bs-toggle="collapse"
data-bs-target="#navbarNav")
      span.navbar-toggler-icon
    #navbarNav.collapse.navbar-collapse
      ul.navbar-nav.ms-auto
        li.nav-item
          a.nav-link(href="/")
            i.bi.bi-house.me-1
            | Аналіз
        li.nav-item
          a.nav-link(href="/comparison")
            i.bi.bi-arrow-left-right.me-1
            | ES5 vs ES6
        li.nav-item
          a.nav-link(href="/stats")
            i.bi.bi-graph-up.me-1
            | Статистика
        li.nav-item.dropdown
                a.nav-link.dropdown-toggle(href="#" role="button"
data-bs-toggle="dropdown" aria-expanded="false")
            i.bi.bi-journal-text.me-1
            | Документація
          ul.dropdown-menu.dropdown-menu-end
            li
              a.dropdown-item(href="/docs/mi") Maintainability Index
            li
                a.dropdown-item(href="/docs/cognitive") Cognitive
Complexity
```

ДОДАТОК В

Керівництво користувача

ЗАТВЕРДЖУЮ
Перший проректор Українського
державного
університету науки і технологій
Анатолій РАДКЕВИЧ

“CODE COMPLEXITY LAB”

Керівництво користувача

ЛИСТ ЗАТВЕРДЖЕННЯ
44165850.1553– 01 І3 01

Завідувач кафедри КІТ
_____Вадим ГОРЯЧКІН
Керівник розробки
_____Віктор ШАРАВАРА
Виконавець
_____Михайло СВИРИДОВ
Нормоконтролер
_____Світлана ВОЛКОВА

2026

ЗАТВЕРДЖЕНО

44165850.1553– 01 І3 01

“CODE COMPLEXITY LAB”

Керівництво користувача

Листів 7

2026

ЗМІСТ

1 Вступ

1.1 Сфера застосування

Веб-застосунок "Code Complexity Lab" призначено для автоматизованого аналізу коду JavaScript з метою оцінювання показників супроводжуваності. Застосунок забезпечує порівняльний аналіз реалізацій коду за стандартами ES5 та ES6.

Основною сферою застосування є науково-дослідна та освітня діяльність, зокрема проведення емпіричних досліджень характеристик коду та навчання принципам написання коду.

1.2 Короткий опис можливостей

Застосунок надає такі можливості:

— аналіз коду JavaScript у режимі реального часу безпосередньо у веб-браузері;

— розрахунок показників супроводжуваності (індекс супроводжуваності, когнітивна та цикломатична складність, показники Холстеда, кількість рядків коду).

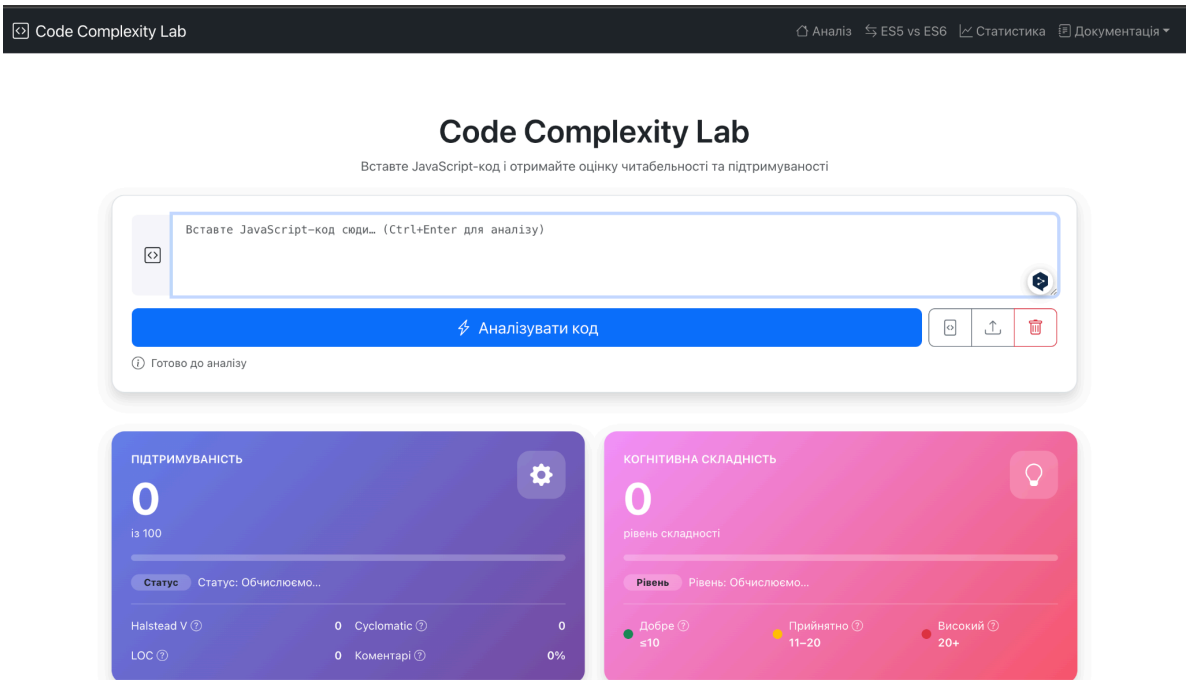


Рисунок 1.1 – Головна сторінка

1.3 Рівень підготовки користувача

Користувач повинен мати базові навички роботи з комп’ютером та інтернетом. Для користувачів, які здійснюють введення та перевірку даних, достатньо базового рівня знань з роботи з ПК.

2 Призначення та умови застосування

2.1 Види діяльності та функції

— проведення наукових досліджень у галузі якості програмного забезпечення;

— навчальна діяльність у процесі викладання дисциплін з розроблення ПЗ;

— оцінювання якості програмного коду при розробленні.

2.2 Умови застосування

— Доступність: Система є веб-застосунком і доступна через інтернет. Немає необхідності в установці додаткового програмного забезпечення.

— Платформа: Потрібен сучасний веб-браузер (Google Chrome 90+, Mozilla Firefox 88+, Safari 14+, Microsoft Edge 90+).

— Технічні вимоги: Процесор 1 ГГц, оперативна пам'ять 2 ГБ, роздільна здатність екрана не менше 1280×720 пікселів.

3 Підготовка до роботи

Перевірте доступ: Увійдіть до застосунку через веб-браузер за наданою адресою.

Перевірка функціональності: Виконайте тестовий аналіз простого фрагмента коду (наприклад: function sum(a, b) { return a + b; }). Натисніть "Аналізувати код" та переконайтеся у відображенні числових значень показників.

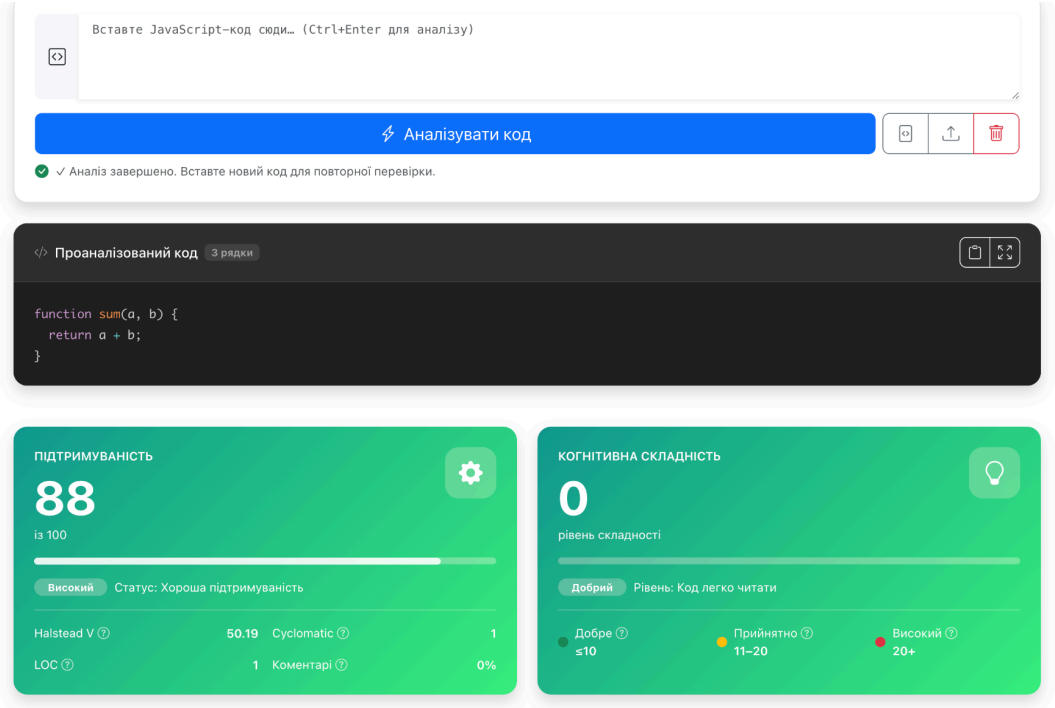


Рисунок 3.1 – Результат тестового фрагмента коду на головній сторінці

4 Опис операцій

Аналіз коду

Введіть JavaScript-код через веб-інтерфейс безпосередньо у текстове поле або завантажте файл кнопкою "Завантажити файл". Натисніть "Аналізувати код". Система автоматично розрахує та відобразить показники супроводжуваності.

Очищення результатів

Натисніть кнопку очищення для видалення введеного коду та результатів аналізу. Це дозволяє почати новий аналіз з чистого аркуша.

5 Операції технологічного процесу обробки даних

Аналіз коду за показниками супроводжуваності:

— найменування: Розрахунок показників супроводжуваності JavaScript-коду

— умови: Наявність синтаксично коректного JavaScript-коду

— підготовчі дії: Введіть код у текстове поле або завантажте файл

— основні дії: Натисніть кнопку "Аналізувати код" для початку розрахунків

— заключні дії: Перегляньте та проаналізуйте результати

— необхідні ресурси: Веб-браузер з підтримкою сучасних стандартів, інтернет-з'єднання

6 Аварійні ситуації

6.1 Дії на випадок недотримання умов виконання технологічного процесу

Перезавантаження застосунку: У разі некоректної роботи веб-застосунку оновіть сторінку у браузері за допомогою клавіші F5 або комбінації Ctrl+R. Це дозволить перезавантажити клієнтську частину та відновити нормальне функціонування.

Діагностика інтернет-підключення: Коли виникають труднощі з доступом до застосунку, переконайтеся у стабільності з'єднання з мережею. Спробуйте відкрити інші веб-сайти для перевірки. За потреби зверніться до постачальника інтернет-послуг.

6.2 Дії з відновлення програм і/або даних у разі відмови магнітних носіїв або виявлення помилок у даних

Відновлення втрачених даних: Оскільки застосунок працює на стороні клієнта без серверного збереження, у випадку втрати введеного коду скористайтеся локальними копіями файлів, які ви могли зберегти на своєму комп'ютері.

Перевірка коректності введених даних: Виконайте контрольний аналіз тестового прикладу для підтвердження працездатності системи розрахунків. Порівняйте отримані показники з еталонними значеннями, наведеними в документації.

6.3 Дії у випадках виявлення несанкціонованого втручання в дані

Повідомлення про інцидент безпеки: При виявленні підозрілої поведінки застосунку або незрозумілих змін у результатах аналізу зверніться до технічної підтримки. Опишіть характер проблеми та обставини її виникнення.

Аналіз інциденту: Технічна підтримка проведе дослідження ситуації та встановить причину виникнення проблеми. За результатами перевірки будуть вжиті відповідні заходи для усунення вразливостей.

Поновлення коректного стану: Після виявлення причини проблеми очистіть кеш браузера, видаліть cookies пов'язані із застосунком та перезавантажте систему для повернення до нормального режиму роботи.

6.4 Дії в інших аварійних ситуаціях

Усунення програмних збоїв: При виникненні непередбачуваних помилок у роботі застосунку спробуйте використати альтернативний веб-браузер. Переконайтеся, що версія вашого браузера відповідає мінімальним технічним вимогам системи.

Актуалізація програмного середовища: Регулярно встановлюйте оновлення для веб-браузера та операційної системи. Сучасні версії програмного забезпечення містять виправлення помилок та покращення продуктивності, що позитивно впливає на роботу застосунку.

7 Рекомендації щодо засвоєння

7.1 Рекомендації щодо засвоєння та експлуатації

Ознайомтеся з експлуатаційною документацією: Перед початком роботи уважно ознайомтеся з керівництвом користувача.

Виконання підготовчих дій: Переконайтеся у відповідності браузера мінімальним технічним вимогам та наявності стабільного інтернет-з'єднання.

Поетапне засвоєння: Почніть з аналізу простих функцій (10-20 рядків), поступово переходьте до більш складних фрагментів коду. Вивчіть довідкову інформацію про кожен показник.

7.2 Опис контрольного прикладу

Введення тестового коду:

```
function filterEvenNumbers(numbers) {
 var result = [];
 for (var i = 0; i < numbers.length; i++) {
   if (numbers[i] % 2 === 0) {
     result.push(numbers[i]);
   }
 }
 return result;
```

```
}
```

Виконання аналізу: Вставте код у текстове поле. Натисніть "Аналізувати код" та переконайтеся у коректності результатів.

Очікувані результати:

— Індекс супроводжуваності: приблизно 67 балів

— Когнітивна складність: 3 одиниць

— Цикломатична складність: 3 одиниці

— Кількість рядків коду: 8 рядків

— Показники Холстеда: відображають обсяг та складність програми

Перевірка довідкової інформації: Ознайомтеся з результатами показників та детальним описом для кращого розуміння отриманих результатів.

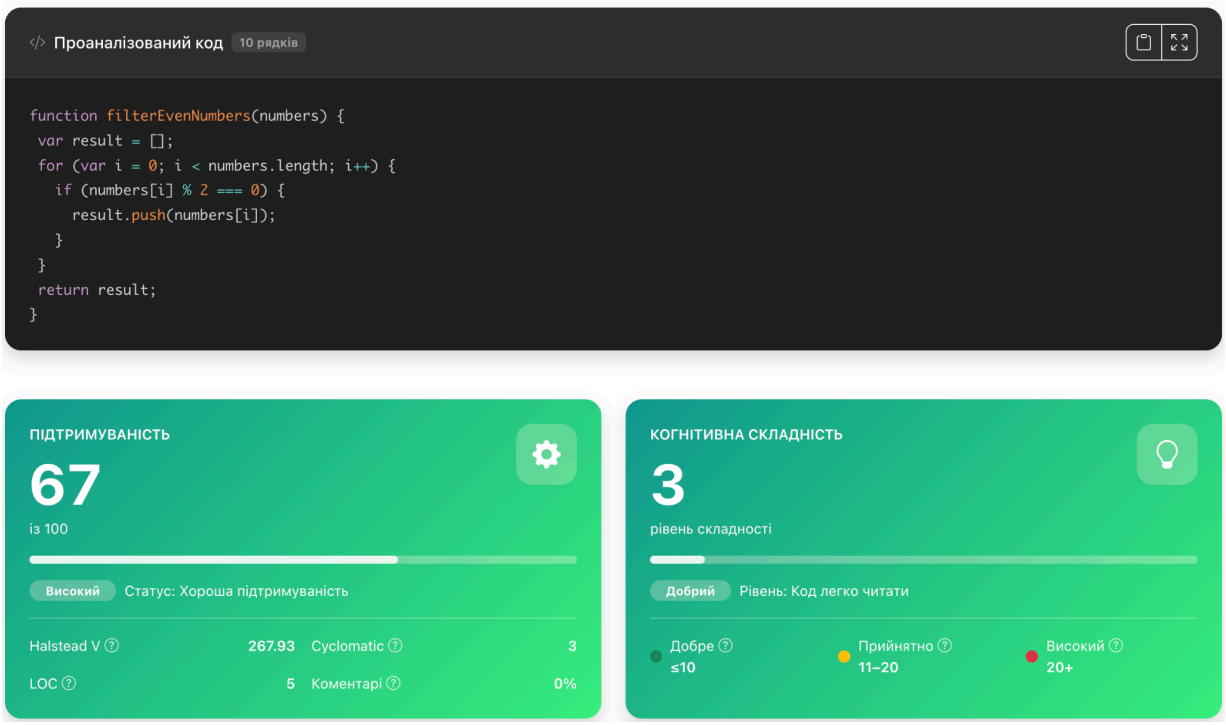


Рисунок 7.1 – Результат контрольного прикладу коду

ДОДАТОК Г

ТЕЗИ ДОПОВІДІ ДЛЯ КОНФЕРЕНЦІЇ

ЗАТВЕРДЖУЮ
Перший проректор Українського
державного
університету науки і технологій
Анатолій РАДКЕВИЧ

“ДОСЛІДЖЕННЯ ВПЛИВУ СТАНДАРТУ ES6 У ПРОГРАМУВАННІ МОВОЮ JAVASCRIPT НА МОЖЛИВОСТІ СУПРОВОДУ КОДУ”

ТЕЗИ ДОПОВІДІ ДЛЯ КОНФЕРЕНЦІЇ

ЛИСТ ЗАТВЕРДЖЕННЯ
44165850.1553–01 90 01–ЛЗ

Завідувач кафедри КІТ
_____Вадим ГОРЯЧКІН
Керівник розробки
_____Віктор ШАРАВАРА
Виконавець
_____Михайло СВИРИДОВ
Нормоконтролер
_____Світлана ВОЛКОВА

2026

ЗАТВЕРДЖЕНО

44165850.1553–01 90 01–ЛЗ

“ДОСЛІДЖЕННЯ ВПЛИВУ СТАНДАРТУ ES6 У ПРОГРАМУВАННІ МОВОЮ JAVASCRIPT НА МОЖЛИВОСТІ СУПРОВОДУ КОДУ”

ТЕЗИ ДОПОВІДІ ДЛЯ КОНФЕРЕНЦІЇ

Листів 6

2026


МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
УКРАЇНСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ НАУКИ І ТЕХНОЛОГІЙ
ABSTRACTS
OF THE XIX INTERNATIONAL CONFERENCE
«MODERN INFORMATION AND COMMUNICATION TECHNOLOGIES ON A TRANSPORT, IN INDUSTRY AND EDUCATION»
18-19, December, 2025
СУЧАСНІ ІНФОРМАЦІЙНІ ТА КОМУНІКАЦІЙНІ ТЕХНОЛОГІЇ НА ТРАНСПОРТІ, В ПРОМИСЛОВОСТІ І ОСВІТІ
ПРИСВЯЧЕНО ПАМ'ЯТІ ПРОФЕСОРА ІГОРЯ ЖУКОВИЦЬКОГО
ТЕЗИ
XIX МІЖНАРОДНОЇ НАУКОВО-ПРАКТИЧНОЇ КОНФЕРЕНЦІЇ
18-19 ГРУДНЯ 2025
ДНІПРО
2025

Міністерство освіти і науки України

Український державний університет науки і технологій

ТЕЗИ
XIX Міжнародної науково-практичної конференції
«СУЧАСНІ ІНФОРМАЦІЙНІ ТА КОМУНІКАЦІЙНІ ТЕХНОЛОГІЇ НА ТРАНСПОРТІ, В ПРОМИСЛОВОСТІ І ОСВІТІ»
Присвячено пам'яті Ігоря ЖУКОВИЦЬКОГО

ABSTRACTS
of the XIX International Conference
«MODERN INFORMATION AND COMMUNICATION TECHNOLOGIES ON A TRANSPORT, IN INDUSTRY AND EDUCATION»
Dedicated to the memory of Igor ZHUKOVYTSKY

18.12.2025 – 19.12.2025

Дніпро
2025

УДК 658.512.2:681.3.06

Сучасні інформаційні та комунікаційні технології на транспорті, в промисловості і освіті: Тези XIX Міжнародної науково-практичної конференції (Дніпро, 18-19 грудня 2025 р.). – Д.: УДУНТ, 2025. – 172 с.

У збірнику представлені тези доповідей XIX Міжнародної науково-практичної конференції «Сучасні інформаційні та комунікаційні технології на транспорті, в промисловості і освіті», яка відбулася 18-19 грудня 2025 року в Українському державному університеті науки та технологій в онлайн форматі. Конференцію присвячено пам'яті Ігоря ЖУКОВИЦЬКОГО, доктора технічних наук, професора кафедри електронних обчислювальних машин (УДУНТ, м. Дніпро). Розглянуто результати теоретичних і експериментальних досліджень, а також проблемні питання функціонування та перспективи розвитку інформаційних технологій транспорту, промисловості й освіти.

Збірник призначений для науково-технічних працівників залізниць, підприємств транспорту, викладачів вищих навчальних закладів, докторантів, аспірантів і студентів.

Адреса редакційної колегії:
49010, м. Дніпро, вул. Лазаряна, 2, УДУНТ

Тези доповідей друкуються мовою оригіналу в редакції авторів.

Дослідження впливу стандарту ES6 у програмуванні мовою JavaScript на можливості супроводу коду

Свиридов М.О., Шаравара В.В, Український державний університет науки і технологій, Україна

У сучасній веб-розробці JavaScript є однією з основних мов програмування, а тому тривалість життєвого циклу веб-додатків залежить, зокрема, від якості та підтримуваності коду, написаного цією мовою. Запровадження стандарту ECMAScript 2015 (ES6) додало до мови низку нових конструкцій (стрілкові функції, let/const, деструктуризацію, модулі, колекції тощо), які інтуїтивно сприймаються як зручніші для розробника, проте їхній вплив на показники підтримуваності коду потребує кількісного обґрунтування.

Метою дослідження є кількісне порівняння варіантів JavaScript-коду, реалізованих на основі стандартів ES5 та ES6, за показниками підтримуваності. Для досягнення цієї мети розроблено веб-додаток, що виконує автоматизований аналіз вихідного коду без залучення серверної обробки та забезпечує відтворюваність експериментальних результатів.

У веб-додатку реалізовано побудову абстрактного синтаксичного дерева та розрахунок показників підтримуваності. Як основні показники було обрано Maintainability Index (за формулою Visual Studio) та Cognitive Complexity (за специфікацією SonarSource). Як допоміжні використовувалися Cyclomatic Complexity, обсяг Halstead та кількість логічних рядків коду без коментарів. Передбачено збереження історії аналізів і візуалізацію зведених даних, що дає змогу виконувати серію порівняльних експериментів над різними варіантами коду.

Експериментальна частина роботи базується на порівнянні 100 пар фрагментів коду ES5 та ES6, які реалізують однакові алгоритми, характерні для типових задач веб-розробки (обробка масивів, форматування тексту, валідація даних, робота зі структурами колекцій). Версії ES5 отримані шляхом послідовного рефакторингу вихідного ES6-коду із збереженням функціональної еквівалентності. Пакетний режим роботи веб-додатка забезпечує автоматичний розрахунок показників для кожного файлу та формування узагальненої статистики.

Середні значення показників демонструють системну перевагу варіантів, що використовують ES6. Maintainability Index зріс з 43,63 до 47,33 бала (+8,48 %), Cognitive Complexity зменшилася з 21,45 до 16,58 (–22,7 %), Cyclomatic Complexity — з 14,59 до 10,30 (–29,4 %), а кількість логічних рядків коду скоротилася з 37,83 до 32,70 (–13,56 %). Це свідчить про спрощення структури розгалужень і зменшення обсягу коду без зміни алгоритмічної суті програм.

Для оцінки статистичної значущості відмінностей застосовано непараметричний тест Вілкоксона для парних вибірок. Для Maintainability Index, Cognitive Complexity та Cyclomatic Complexity отримано p-value < 0,001, що відповідає рівню довіри 99,9 %. Розрахований розмір ефекту для Maintainability Index та Cyclomatic Complexity перевищує 0,8, що інтерпретується як дуже великий практичний ефект і підтверджує неможливість пояснити отримані відмінності випадковими коливаннями.

Узагальнюючи результати, можна стверджувати, що використання конструкцій ES6 у типових задачах веб-розробки сприяє покращенню показників підтримуваності JavaScript-коду. Розроблений веб-додаток дає змогу не лише кількісно оцінювати вплив рефакторингу та переходу від ES5 до ES6, але й використовувати його як навчальний інструмент для формування у розробників практик написання більш зрозумілого, компактного та структуровано простішого коду.