Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)
Навчально-науковий інститут комп'ютерних наук та інформаційних технологій
(повне найменування інституту, назва факультету (відділення))
Кафедра комп'ютерних наук
(повна назва кафедри (предметної, циклової комісії))

Магістерська кваліфікаційна робота

другий (магістерський)
(рівень вищої освіти)

на тему: «Система аналізу та оцінки популярності технологій і фреймворків»

Виконав: студент 6 курсу групи КН-61м
спеціальності 122 "Комп'ютерні науки"
(шифр і назва напряму підготовки, спеціальності)
Клебан Д. І.
(прізвище та ініціали)
Керівники: Капран І. Д., Борецька І. Б.
(прізвище та ініціали)
Рецензент: Фицд Л.О.
(прізвище та ініціали)

Львів – 2025

Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

ННІ *комп'ютерних наук та інформаційних технологій*
Кафедра *комп'ютерних наук*
Рівень вищої освіти *другий (магістерський)*
Спеціальність *122 "Комп'ютерні науки"*
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
_____ Борецька І. Б.
"10" грудня 2025 року

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Клебану Данилу Ігоровичу
(прізвище, ім'я, по батькові)

1. Тема роботи *«Система аналізу та оцінки популярності технологій і фреймворків»*
керівники роботи *Капран Ігор Дмитрович, Борецька Ірина Богданівна, к. т. н., доцент*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від *"29"квітня 2025 р.* № С-288

2. Термін подання студентом роботи *10 грудня 2025 року*

3. Вихідні дані до роботи *Огляд літературних джерел та аналіз шляхів вирішення поставлених завдань, характеристика процесу розроблення системи аналізу та оцінки популярності технологій і фреймворків, аналіз технологій для програмної реалізації, організаційна структура та графічний інтерфейс для розробленої системи, тестування розробленої системи для аналізу та оцінки популярності технологій і фреймворків.*

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити) *Вступ. Розділ 1. Стан проблемної області. Розділ 2. Інформаційне забезпечення. Розділ 3. Математичне забезпечення. Розділ 4. Програмне забезпечення. Розділ 5. Розроблення стартап-проєкту. Висновки. Список використаних джерел. Додатки.*

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Слайди для доповіді загальним обсягом 10-15 одиниць

6. Дата видачі завдання *01 травня 2025 року*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1	Огляд літературних джерел згідно досліджуваної тематики. Збір необхідних матеріалів. Формування функціональних вимог та постановка технічного завдання. Оформлення першого розділу пояснювальної записки.	01.05.2025 р. 20.06.2025 р.	***Виконано***
2	Розроблення математичного та алгоритмічного забезпечення	20.06.2025 р. 24.07.2025 р.	***Виконано***
3	Програмна реалізація та аналіз результатів	24.07.2025 р. 28.09.2025 р.	***Виконано***
4	Розроблення стартап-проєкту	28.09.2025 р. 15.11.2025 р.	***Виконано***
5	Оформлення пояснювальної записки та здача на рецензування	15.11.2025 р. 10.12.2025 р.	***Виконано***

Студент _____ Клебан Д. І.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Капран І. Д.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Борецька І. Б.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Магістерська робота містить 70 сторінок пояснювальної записки, 15 рисунків, 2 таблиці, 1 додаток та 25 джерел.

У магістерській кваліфікаційній роботі проведено аналіз сучасних методів оцінки популярності IT-технологій та фреймворків за даними GitHub. Розроблено вебзастосунок «GitHub Trend Analyzer», який дозволяє збирати дані через GitHub API, розраховувати H-індекси, порівнювати технології за активністю спільноти та відображати тренди у часовому проміжку. У роботі продемонстровано приклад аналізу RPC-фреймворків (gRPC, JSON-RPC, SOAP, Apache Thrift, Dubbo), що дозволяє визначити домінуючі технології та їхню популярність серед розробників. Практична частина включає реалізацію інтерфейсу користувача для введення запитів, отримання результатів та експорту даних для подальшого аналізу.

Ключові слова: GitHub, аналіз трендів, H-індекс, вебінструмент, програмне забезпечення.

ABSTRACT

The master's thesis consists of 70 pages of explanatory notes, 15 figures, 2 tables, 1 addition, and 25 references.

The work presents an analysis of modern methods for evaluating the popularity of IT technologies and frameworks using GitHub data. A web application, GitHub Trend Analyzer, has been developed, allowing data collection via the GitHub API, calculation of H-indices, comparison of technologies by community activity, and visualization of trends over time. The thesis demonstrates an example analysis of RPC frameworks (gRPC, JSON-RPC, SOAP, Apache Thrift, Dubbo), enabling identification of dominant technologies and their popularity among developers. The practical part includes the implementation of a user interface for query input, result retrieval, and data export for further analysis.

Keywords: GitHub, trend analysis, H-index, web tool, software.

ТЕХНІЧНЕ ЗАВДАННЯ

Необхідно дослідити та розробити вебінструмент GitHub Trend Analyzer для оцінки популярності IT-технологій та фреймворків на основі метрик GitHub (Star H-Index, Fork H-Index) та трендової активності репозиторіїв.

Завдання включає такі кроки:

1. Провести огляд літературних джерел щодо методів оцінки популярності технологій та використання H-індексів для аналітики GitHub.
2. Охарактеризувати процес збору та обробки даних через GitHub API, а також алгоритми обчислення Star та Fork H-індексів.
3. Провести аналіз технологій та інструментів, необхідних для програмної реалізації вебінструменту (бекенд, фронтенд, бази даних, бібліотеки для графіків).
4. Розробити вебінтерфейс для введення пошукових запитів, відображення результатів, порівняння технологій та побудови часових трендів.
5. Провести тестування розробленого вебінструменту, включаючи перевірку коректності обчислення H-індексів, порівняння технологій, відображення трендів та експорту даних.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ…….....8
ВСТУП 9
РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ.....11
1.1. Загальний контекст проблемної області…….....11
1.2. Індекс Хірша……...16
Висновки до розділу……..18
РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ.....19
2.1. Огляд використаних джерел даних……...20
2.2. Інструменти та середовища обробки даних…….....23
2.3. Методи аналізу популярності мов та фреймворків…….24
Висновки до розділу.....27
РОЗДІЛ 3. МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ.....28
3.1. Математична модель.....28
3.2. Формальні моделі вихідних даних……29
3.3. Моделі на основі узагальненого h-індексу……...30
Висновки до розділу 33
РОЗДІЛ 4. ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ.....34
4.1. Загальні цілі та функціональні вимоги ……34
4.2. Архітектура системи ….....35
4.3. Розроблення застосунку GitHub Trend Analyzer……..36
4.4. Основні компоненти та їхні функції ……44
4.5. Вебінтерфейс вебзастосунку GitHub Trend Analyzer……..47
Висновки до розділу……..50
РОЗДІЛ 5. РОЗРОБЛЕННЯ СТАРТАП-ПРОЄКТУ…….....51
5.1. Опис ідеї проєкту……51
5.2. Аналіз технологічних можливостей реалізації ідей проєкту…….52
5.3. Аналіз ринкових можливостей запуску стартап-проєкту …….....53
Висновки до розділу……..……56
ВИСНОВКИ…….....57

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ……..58
ДОДАТКИ……..60
ДОДАТОК А……..60

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

GitHub – платформа для розміщення та спільного розроблення програмного забезпечення, дозволяє зберігати код, керувати версіями та відстежувати активність спільноти через репозиторії, зірки (stars), форки (forks) та issues.

API (англ. Application Programming Interface) — набір протоколів та інструментів для взаємодії між програмним забезпеченням.

REST (англ. Representational State Transfer) — один зі способів отримання даних через API, заснований на запитах до ресурсів із використанням HTTP.

GraphQL (англ. Graph Query Language) — дозволяє гнучко отримувати потрібні дані з GitHub та аналізувати тільки необхідну інформацію.

RPC (англ. Remote Procedure Call) — технологія для взаємодії між програмними компонентами, прикладом є фреймворки gRPC, JSON-RPC, SOAP, Thrift, Dubbo.

gRPC (англ. Google Remote Procedure Call) — забезпечує швидкий обмін даними між сервісами та широко використовується для мікросервісної архітектури.

JSON-RPC (англ. JSON Remote Procedure Call) — дозволяє викликати віддалені процедури з обміном даними у форматі JSON.

SOAP (англ. Simple Object Access Protocol) — стандарт, який забезпечує структурований обмін даними між системами.

Thrift (англ. Apache Thrift) — дозволяє визначати типи даних і сервіси для кросплатформенного обміну інформацією.

Dubbo (англ. Apache Dubbo) — відкритий фреймворк для високопродуктивних Java-додатків.

h-індекс (англ. Hirsch index) — наукова метрика, яка показує продуктивність і цитованість дослідників; у роботі адаптована для оцінки популярності технологій через кількість репозиторіїв, зірок і форків.

CSV – формат файлу для експорту даних (англ. Comma-Separated Values) — текстовий формат для збереження таблиць з показниками та метриками.

GUI (англ. Graphical User Interface) — інтерфейс веб-інструменту для взаємодії користувача з даними.

ВСТУП

У даній магістерській кваліфікаційній роботі розглядається процес створення системи аналізу та порівняння популярності технологій програмної розробки на основі відкритих даних GitHub. Фактично йдеться про формування аналітичного сервісу, який надає можливість відстежувати активність розробницьких спільнот, виявляти тенденції та визначати реальний рівень використання фреймворків і бібліотек шляхом автоматизованої обробки метрик GitHub.

Актуальність проблеми. Створення подібної системи можна порівняти зі збиранням технічного конструктора, де кожен елемент — це окрема частина складного процесу: робота з API, обчислення показників активності, формування аналітичних звітів та побудова візуалізацій. Для отримання достовірних результатів необхідно правильно підібрати «деталі» — метрики, моделі та алгоритми — і визначити, як саме їх поєднати, щоб система могла порівнювати технології за рівнем популярності та динамікою розвитку. Саме таку «інструкцію з побудови» ефективної аналітичної системи розроблено в межах цієї магістерської роботи.

Об’єкт дослідження. Аналітична система для визначення та порівняння популярності технологій розроблення програмного забезпечення.

Предмет дослідження. Підхід до формування метрик, моделей і алгоритмів аналізу популярності технологій на основі GitHub-даних, включно з адаптованими H-індексами та показниками активності спільнот.

Мета роботи. Метою даної роботи є дослідження, розроблення та практична реалізація системи аналізу популярності технологій на основі даних GitHub із використанням спеціалізованих метрик, що дозволяють здійснювати об’єктивне порівняння фреймворків і визначення тенденцій їх розвитку.

Основні завдання, що мають бути реалізовані у роботі:

- розроблення модуля збору та обробки GitHub-даних;
- створення механізму обчислення метрик, включно зі Star H-Index, Fork H-Index та динамічними показниками активності;
- впровадження системи порівняння технологій за комплексом метрик;

- проєктування JSON-API для взаємодії із зовнішніми застосунками та сервісами;
- розроблення вебінтерфейсу для перегляду результатів аналізу та візуалізації трендів.

Програмне забезпечення реалізовано із використанням технологій REST-API, GitHub GraphQL API, методів обробки великих обсягів даних та спеціалізованих алгоритмів статистичного аналізу метрик репозиторіїв.

Новизна роботи. Запропонована система поєднує різнорідні GitHub-метрики в єдину аналітичну модель, що дає змогу оцінювати популярність технологій точніше, ніж традиційні методи. Використання адаптованих H-індексів для аналізу репозиторіїв дозволяє отримати глибше уявлення про активність спільнот та реальний рівень застосування технологій. Такий підхід формує передумови для створення сервісів, здатних автоматично інформувати компанії та розробників про зміни технологічного ринку та тенденції його розвитку.

Практичне значення одержаних результатів. Створений інструмент може бути використаний компаніями, IT-фахівцями та освітніми установами для моніторингу популярності технологій, порівняння фреймворків та планування технологічного стеку майбутніх проєктів. Система здатна бути частиною внутрішніх аналітичних платформ або використовуватися у відкритому доступі як сервіс для оцінювання актуальних трендів програмної інженерії.

РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ

1.1. Загальний контекст проблемної області

У сучасній IT-індустрії популярність технологій визначає багато стратегічних рішень — від вибору стеку для нового проєкту до оновлення навчальних програм в університетах. При цьому класичні джерела інформації, такі як Google Trends або рейтинги популярних сайтів, не завжди відображають реальну активність розробників. Наприклад, висока кількість пошукових запитів не гарантує глибокого застосування технології в проєктах, а рейтинги мов програмування часто охоплюють лише обмежений спектр даних. Google Trends - це сервіс, який дає змогу порівняти популярність термінів у часі, він показує лише запити користувачів, а не їхню реальну участь у розробці. Це зручний інструмент, який дозволяє порівняти популярність тих чи інших ключових слів серед запитів. Але результати виявилися неоднозначними [1]. За даними Google Trends SOAP та gRPC здаються приблизно однаково популярними, але це не відповідає фактичній активності на GitHub. На рисунку 1. 1 представлений графік популярності SOAP, gRPC та JSON-RPC.

Рисунок 1. 1 - Графік популярності SOAP, gRPC та JSON-RPC

Згідно з Google Trends, SOAP і gRPC сьогодні приблизно однаково популярні, а JSON-RPC значно відстає. Мій досвід підказує, що сучасні проєкти рідко вибирають SOAP для нових розробок. Більше того, остання версія SOAP вийшла у 2007 році, тоді як останній стабільний реліз gRPC був 3 тижні тому. Складається відчуття, що статистика пошукових запитів не зовсім коректно відображає популярність технологій.

Існують інші джерела, на підставі яких можна зіставити популярність RPC бібліотек. Наприклад, на BestofJS можна знайти рейтинг RPC-бібліотек для JavaScript (рис. 1. 2). Сайти з рейтингами технологій, такі платформи, як BestofJS, орієнтовані на конкретні мови (наприклад, JavaScript), і не охоплюють кросплатформні фреймворки або бібліотеки [2].

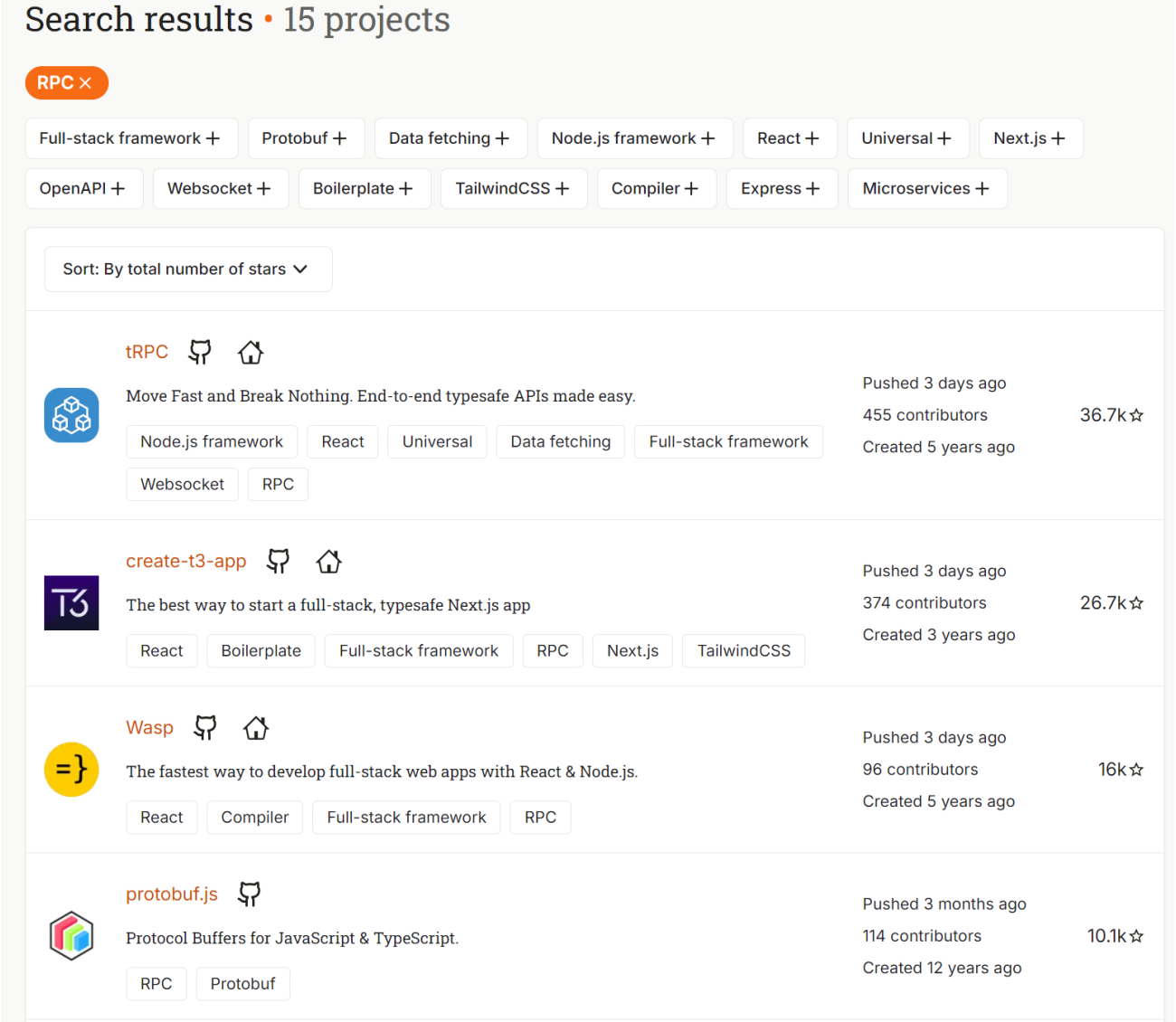


Рисунок 1. 2 - Рейтинг RPC-бібліотек для JavaScript

Тут також можна знайти проєкти, пов'язані з gRPC (зараз вони починаються з 4-го місця protobuf.js і далі за списком). Але враховуються тільки проєкти, пов'язані з JavaScript, а ми хочемо оцінити популярність кросплатформових RPC проєктів.

Якби ми розглядали мови програмування, то мали б більше доступних інструментів. Наприклад, IEEE Spectrum Language Ranking — один із найпопулярніших рейтингів популярності мов програмування. Згідно з їхньою методологією, вони аналізують 8 ключових джерел даних: Google, Stack Overflow, IEEE Xplore Digital Library, IEEE Job Site (ну природно, це рейтинг IEEE), CareerBuilder, GitHub, Trinity College Dublin Library і Discord. Підхід IEEE Spectrum дійсно ґрунтовний (рис. 1. 3). Оцінка різних джерел рейтингів, наприклад IEEE Spectrum, комбінують дані з багатьох джерел (Job-платформи, публікації, GitHub тощо), але їхнє дослідження та аналіз потребують значних ресурсів [3, 4].

Рисунок 1. 3 - IEEE Spectrum: Найкращі мови програмування 2024 року

І тут я натрапив на низку альтернативних проєктів, зокрема GitHub Language Statistics, який використовує дані Google BigQuery для аналізу активності на GitHub. Ідея аналізу GitHub здалася мені перспективною, і я вирішив рухатися у цьому напрямку [5].

Допустимо, ми звідкись знаємо, що серед найвідоміших кросплатформових RPC фреймворків можна виділити SOAP, gRPC, JSON-RPC, Apache Thrift та Dubbo. Як

порівняти їхню популярність серед open source проєктів на GitHub? Ми вирішили провести простий аналіз вручну, використовуючи пошук GitHub. Поставимо всі проєкти в рівні умови та встановимо, що ми шукаємо результати за останні, припустимо, 2 роки (з 2023 року). Виходить такий запит:

```
NAME_OF_THE_FRAMEWORK created:>2023-01-01
```

GitHub пошук показує не тільки результати пошуку, але й статистику щодо загальної кількості репозиторіїв, Issues, пул-реквестів та дискусій (рис. 1. 4).

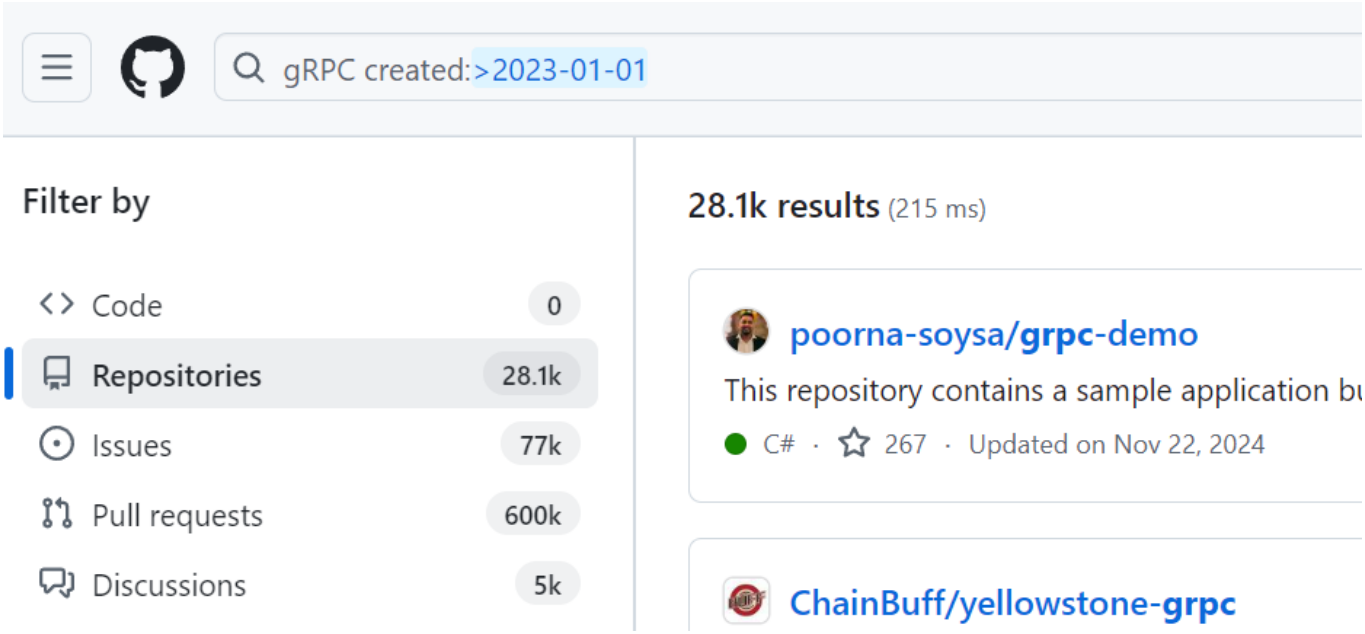


Рисунок 1. 4 - Статистика по репозиторіям, issues, pull requests та дискусіям у результатах пошуку GitHub

Пошук за описаним вище методом дає наступний результат (напівжирним виділено рядок з першим місцем за кожним критерієм; підкресленням - друге місце; похилим шрифтом - третє місце).

Таблиця 1. 1 - Оцінка кількісних показників технологій

Технологія	Репозиторії	Issues	Pull Requests	Discussions
gRPC	**28k**	**77k**	**600k**	**5k**
JSON-RPC	851	35k	98k	2k
SOAP	7.7k	*11k*	12k	*781*
Thrift	*2.5k*	8k	*14k*	320
Dubbo	983	3k	10k	163

На відміну від Google Trends, ми можемо детально оцінити кількісні показники кожної з технологій:

1. gRPC явно домінує за всіма показниками, що говорить про широке впровадження, високу динаміку співтовариства та активний розвиток.

2. SON-RPC хоч і має найменшу кількість репозиторіїв (851), але дуже високу кількість issues (35k), pull requests (98k) та обговорень (2k), що може говорити про активне використання.

3. hrift та Dubbo показують помірно-низькі показники по всіх метриках.

4. SOAP показує напрочуд велику кількість репозиторіїв (7.7k), що вказує на використання, що триває. Однак можна помітити дуже невелику кількість pull requests (12k) та обговорень (781) для такого великого числа репозиторіїв.

SOAP, незважаючи на велику кількість проєктів, просідає по pull requests та обговоренням. Складається враження, що проєкти, пов'язані з SOAP, не дуже популярні.

GitHub, як одна з найбільших платформ розробки відкритого коду, являє собою надзвичайно цінне джерело даних про справжню діяльність IT-спільнот. Через API GitHub можливо аналізувати репозиторії, пул-реквести, issues та інші події, що дозволяє побудувати більш точний аналіз популярності технологій.

Прості кількісні показники, такі як кількість репозиторіїв або зірок (“stars”) на GitHub, не враховують якість, активність або довгострокову значимість проектів. Це може призвести до переоцінки технологій, які мають масштабні, але малоактивні репозиторії [6].

1.2. Індекс Хірша

В академічному середовищі вже стикалися з подібною проблемою. Буває, що у вченого (чи університету, чи журналу) купа статей. Але важлива не лише кількість, а те, наскільки вони затребувані та цитовані.

Хорхе Хірш запропонував індекс, який тепер має його ім'я: індекс Хірша. Якщо відсортувати всі статті автора за спаданням цитованості, то h-індекс - це таке число h, що стаття номер h у цьому списку процитована як мінімум h разів. Тобто, автор з h=10 має як мінімум 10 статей, кожну з яких процитовано не менше 10 разів [7].

На рисунку 1. 5 зображено розрахунок h-індексу для 18 статей деякого автора. Якщо ми відсортуємо їх за кількістю цитат, то побачимо, що 5 статей з 18 мають по 5 або більше цитат, а решта 13 мають менше 5 цитат. Отже, при цьому для даного автора h=5.

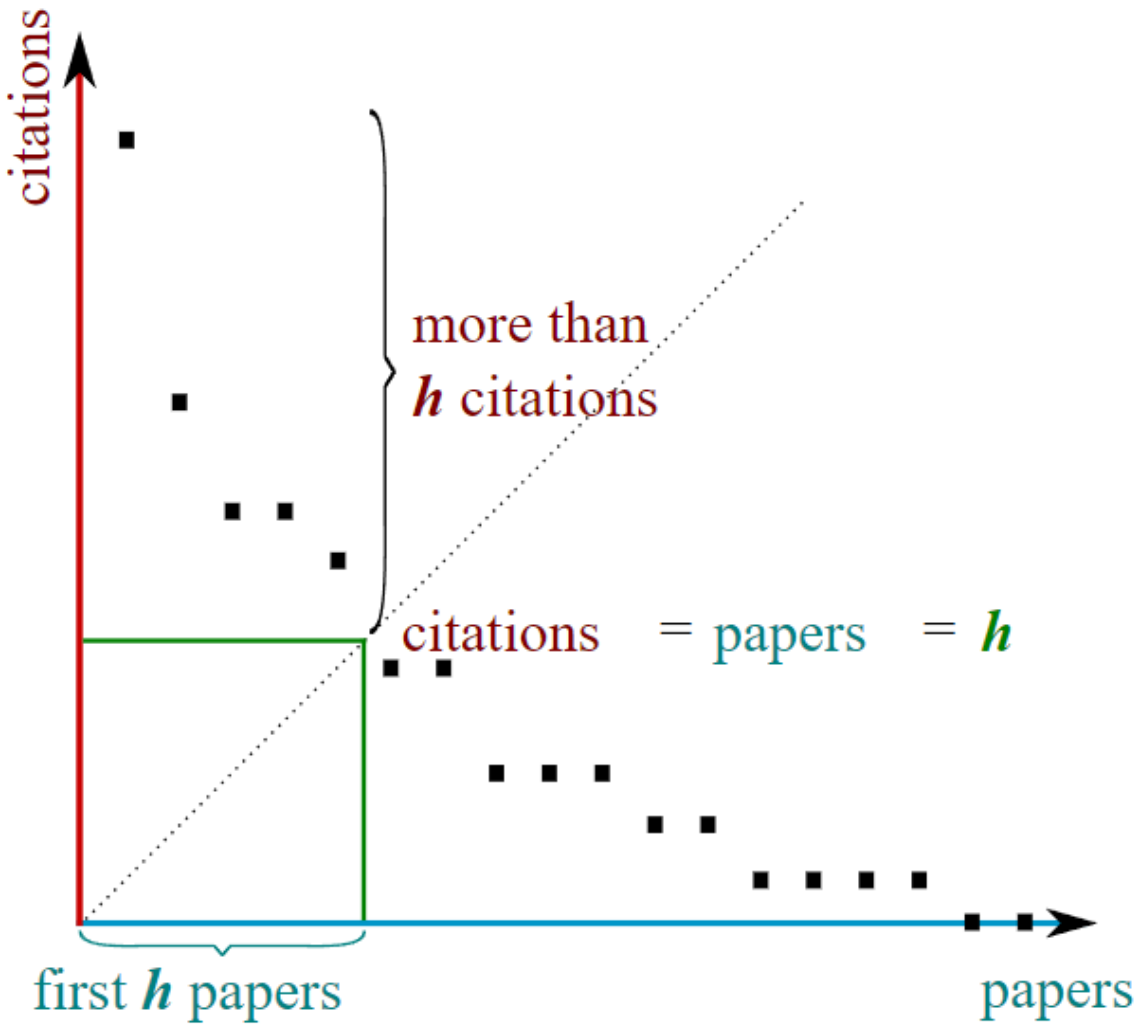


Рисунок 1. 5 - Ілюстрація розрахунку індексу Хірша

Були спроби застосування індексу Хірша та GitHub, але вони в основному обмежувалися аналізом окремих авторів [8]. Мені здалося цікавим застосувати концепцію індексу Хірша ширше. Я пропоную два нові показники для оцінки популярності технологій та фреймворків:

1. Star H-Index (h☆): Технологія має індекс h☆, якщо існує h☆ репозиторіїв за цією технологією, кожен з яких має не менше h☆ зірок.
2. Fork H-Index (h🍴): Технологія має індекс h🍴, якщо існує h🍴 репозиторіїв за цією технологією, кожен з яких має не менше h🍴 форків.

Star H-Index відображає видимість проєкту (зірки вказують на популярність), а Fork H-Index відображає активну участь спільноти (форки показують, скільки людей активно досліджують або розширюють проєкт).

Цей показник можна вимірювати як глобально, і у контексті конкретного часового проміжку. Для прикладу, розрахуємо значення h☆ і h🍴 для наших технологій з 2023 року (табл. 1. 2).

Таблиця 1. 2 - Значення h☆ та h🍴 для технологій з 2023 року

Технологія	h☆	h🍴
gRPC	58	24
SOAP	13	11
JSON-RPC	23	11
Thrift	12	9
Dubbo	19	11

Схоже, що ці метрики справді дозволяють виміряти популярність відповідних технологій у репозиторіях GitHub. gRPC підтверджує своє лідерство за обома показниками, тоді як JSON-RPC проривається на друге місце з h☆=23, що показує поточну популярність проєкту. SOAP, Thrift та Dubbo показують дуже близькі показники h☆(в діапазоні від 12 до 19) та h🍴 (в діапазоні від 9 до 11), що може говорити про їх обмежене використання у проєктах.

Висновки до розділу

У результаті проведеного огляду проблемної області було встановлено, що сучасний набір інструментів для створення та розгортання мовних агентів характеризується активним розвитком, але все ще перебуває на етапі формування єдиних стандартів. Попри наявність великої кількості бібліотек і фреймворків, таких як LangChain, Haystack, LlamaIndex та інших, розробники стикаються з низкою труднощів: фрагментованістю підходів, нечіткими межами між компонентами систем, а також різним рівнем абстракції, який пропонують інструменти.

Аналіз також показав, що впровадження мовних агентів у прикладних сферах стримується не браком технологічних можливостей, а складністю їх коректного узгодження між собою: інтеграцією інструментів, контрольованим виконанням дій, безпекою та інтерпретацією інструкцій. Саме тому виникає потреба у системах, які пропонують чіткі інтерфейси, контрольовану взаємодію агресивних компонентів та можливість гнучкої конфігурації поведінки.

Таким чином, огляд проблемної області демонструє, що існує значний запит на уніфікований підхід до побудови мовних агентів, який забезпечить надійність, повторюваність результатів та простоту інтеграції. Це підтверджує актуальність обраного напряму дослідження та обґрунтовує необхідність створення системи, що використовує сучасні підходи до розроблення агентних архітектур та здатна ефективно працювати в реальних умовах.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

Інформаційне забезпечення магістерської кваліфікаційної роботи охоплює сукупність джерел, методів, інструментів та процедур, що забезпечують отримання, обробку, аналіз і систематизацію даних, необхідних для дослідження динаміки розвитку сучасних технологій. Формування інформаційної бази здійснюється на основі відкритих даних, аналітичних матеріалів та технічних ресурсів, що дозволяють отримувати об’єктивні та статистично значущі показники активності розробницьких спільнот.

У сучасних умовах стрімкого розвитку інформаційних технологій аналіз популярності мов програмування та фреймворків є важливим завданням для науковців, розробників і бізнесу. Від вибору інструментів програмування залежить ефективність розробки, підтримка проєктів, а також конкурентоспроможність фахівців на ринку праці. З огляду на це, виникає потреба у використанні масштабованих, достовірних і репрезентативних джерел даних, які відображають реальну активність розробників у глобальному масштабі [9].

Даний розділ присвячено детальному опису інформаційного забезпечення дослідження, що базується на аналізі даних GitHub Archive із використанням платформи Google BigQuery. Описано джерела даних, інструменти та середовища обробки, методи аналізу, процедури забезпечення відтворюваності, а також етичні та правові аспекти роботи з відкритими даними. Особливу увагу приділено обґрунтуванню вибору інструментів, оцінці достовірності та обмежень даних, а також порівнянню з іншими підходами до аналізу популярності мов програмування.

2.1. Огляд використаних джерел даних

Фундаментальним джерелом даних виступає платформа GitHub, яка є наймасштабнішим світовим середовищем для спільної розробки програмного забезпечення [10]. Інформація, отримана з GitHub, включає:

- кількість та характеристику репозиторіїв за тематикою;
- індикатори активності (issues, pull requests, discussions);
- історичні дані про створення та оновлення проєктів;
- показники залученості користувачів (stars, forks).

Для підвищення достовірності аналізу та перехресної перевірки даних додатково використовуються:

- **Google Trends** — для виявлення інформаційних трендів у мережі Інтернет;
- **Best of JS** — каталог і рейтинг популярних JavaScript-бібліотек;
- **IEEE Spectrum Ranking** — багатофакторний рейтинг мов програмування;
- **GitHub Language Statistics** через **Google BigQuery** — для оцінки частки мов у загальному масиві репозиторіїв.

Комплексне використання зазначених ресурсів забезпечує широку інформаційну основу та підвищує узгодженість результатів.

Основним технічним інструментом збору даних є GitHub API у форматах REST та GraphQL, що дозволяє отримувати розширені та структуровані відомості про репозиторії. Авторизація за допомогою персонального токена доступу дає змогу виконувати запити підвищеної складності, а також забезпечує отримання повного набору метрик [11].

Зібрані дані автоматизовано класифікуються за тематичними, часовими та кількісними показниками, що дає змогу сформувати інформаційний масив, придатний для подальшого аналітичного опрацювання.

Аналітична обробка даних спирається на застосування як традиційних статистичних методів, так і спеціалізованих метрик, адаптованих до специфіки open-source-проєктів. Центральною метрикою є модифікований індекс Хірша, використаний у двох різновидах:

- **Star H-Index** — кількісно характеризує рівень популярності технології,
- **Fork H-Index** — визначає глибину залученості спільноти у розвиток проєктів.

Використання H-індексу дозволяє мінімізувати вплив випадкових або надмірно популярних репозиторіїв, забезпечуючи більш стійку та об’єктивну оцінку.

Крім того, застосовуються:

- аналіз часових рядів для виявлення динаміки розвитку;
- порівняльний аналіз різних технологій і фреймворків;
- статистична оцінка активності користувачів;
- графічна інтерпретація результатів.

Інформаційний потік у межах дослідження сформовано як логічно впорядковану послідовність:

 формування пошукових запитів до API;

 автоматизований збір показників;

 первинне структурування інформації;

 обчислення індексів і статистичних метрик;

 побудова графічних моделей;

 систематизація результатів для наукового аналізу;

 архівація даних для подальшої роботи.

Така організація забезпечує цілісність інформаційної системи та узгодженість етапів дослідження.

Застосування відкритих даних GitHub є методологічно обґрунтованим, оскільки ресурс характеризується високою репрезентативністю, регулярністю оновлення та широким спектром охоплення технологічних напрямів. Отримані дані є достатньо валідними для аналітичних висновків.

Разом з тим, інформаційне забезпечення має низку обмежень:

- аналіз охоплює лише публічні репозиторії;
- можливе потрапляння до вибірки незначущих або навчальних проєктів;
- робота з API підпорядковується технічним обмеженням платформи.

Попри зазначені фактори, інформаційна база відповідає вимогам дослідження та забезпечує достатній рівень об’єктивності.

GitHub Archive — це відкритий проєкт, який фіксує всі публічні події, що відбуваються на платформі GitHub, починаючи з 2011 року [12]. До таких подій належать створення репозиторіїв, коміти, відкриття та закриття issues, pull requests, додавання учасників до проєктів, а також інші дії розробників. Дані агрегуються у вигляді JSON-файлів, які оновлюються щогодини та містять понад 15 типів подій.
Кожен запис у GitHub Archive містить структуровану інформацію про тип події, час, учасника, репозиторій, а також додаткові параметри у полі payload. Це дозволяє здійснювати глибокий аналіз активності розробників, відстежувати динаміку використання мов програмування, фреймворків, бібліотек, а також виявляти тренди у розвитку open source-екосистеми.

Важливою перевагою GitHub Archive є його відкритість, регулярне оновлення та масштабованість. Дані охоплюють глобальну спільноту розробників, що забезпечує репрезентативність аналізу. Однак слід враховувати, що GitHub Archive містить лише публічні репозиторії, а приватна активність залишається поза межами дослідження [1].

Google BigQuery — це хмарна аналітична платформа, яка дозволяє виконувати SQL-запити до великих обсягів даних у режимі реального часу. BigQuery підтримує обробку терабайтів і навіть петабайтів даних за лічені хвилини, що робить її ідеальним інструментом для аналізу масивних датасетів, таких як GitHub Archive [2, 3, 4].
GitHub Archive доступний як публічний датасет у BigQuery, що дає змогу дослідникам і аналітикам виконувати складні SQL-запити без необхідності завантаження та локального зберігання даних. Датасет автоматично оновлюється щогодини, а структура таблиць дозволяє агрегувати дані за роками, місяцями, днями та типами подій [1, 5].

BigQuery також підтримує інтеграцію з іншими сервісами Google Cloud, такими як Data Studio, Looker, Colab Enterprise, а також сторонніми BI-інструментами (Tableau, Power BI тощо) для подальшої візуалізації та аналізу результатів [6, 7].

2.2 Інструменти та середовища обробки даних

Для обробки, аналізу та візуалізації даних у дослідженні використовуються такі основні інструменти:

• SQL (GoogleSQL) — основна мова для формування запитів у BigQuery. Підтримує стандарт SQL:2011, а також розширення для роботи з JSON, геоданими, машинним навчанням тощо [2, 3].

• Python — використовується для попередньої обробки даних, автоматизації ETL-процесів, аналізу результатів та побудови візуалізацій. Основні бібліотеки:

-pandas — для роботи з табличними даними;

- matplotlib, seaborn — для побудови графіків і діаграм;

- google-cloud-bigquery, pandas-gbq — для інтеграції з BigQuery та виконання SQL-запитів із Python-оточення [8, 9, 10].

• Jupyter Notebook / Colab Enterprise — інтерактивне середовище для розробки, тестування та документування аналітичних сценаріїв, що дозволяє поєднувати код, текстові пояснення та візуалізації в одному документі [11, 12, 13].

Дані GitHub Archive доступні для завантаження у вигляді стиснутих JSON-файлів або безпосередньо через публічний датасет у BigQuery. Для ефективної роботи з великими обсягами інформації використовується імпорт у BigQuery, що дозволяє уникнути дублювання даних і забезпечує швидкий доступ до актуальної інформації.

Перед аналізом дані проходять етапи очищення, які включають:

- **Видалення дублікатів** — використання SQL-функцій (наприклад, ROW_NUMBER() OVER (PARTITION BY ...)) для ідентифікації та усунення повторюваних записів, особливо при обробці подій, що можуть дублюватися через технічні причини або помилки імпорту.
- **Обробка пропущених значень** — фільтрація або заповнення відсутніх даних, наприклад, у полях мови програмування, опису репозиторію тощо.
- **Нормалізація форматів дат і текстових полів** — приведення дат до єдиного формату (UTC), очищення тексту від зайвих символів, кодування тощо.

- **Фільтрація релевантних подій** — відбір лише тих типів подій, які мають значення для аналізу популярності мов (наприклад, створення репозиторіїв, коміти, pull requests, issues), ігнорування службових або неінформативних подій (наприклад, додавання учасників, зміна налаштувань).

Для підвищення інформативності аналізу здійснюється збагачення даних шляхом:

- Витягування вкладених полів із JSON — використання SQL-функцій (JSON_EXTRACT, JSON_EXTRACT_SCALAR) для отримання значень з вкладених структур, наприклад, визначення мови програмування репозиторію, імені фреймворку тощо.

- Агрегація за часовими періодами — групування даних за роками, місяцями, кварталами для побудови часових рядів і трендових графіків.

- Злиття з іншими датасетами — при необхідності, інтеграція з додатковими джерелами (наприклад, Stack Overflow, GHTorrent, RedMonk) для порівняння результатів і підвищення достовірності висновків.

2. 3 Методи аналізу популярності мов та фреймворків

Основним підходом до оцінки популярності мов програмування та фреймворків є підрахунок кількості релевантних подій, пов’язаних із певною мовою або технологією:

Кількість створених репозиторіїв — показник активності спільноти навколо мови або фреймворку.

Кількість комітів, pull requests, issues — відображає рівень підтримки, розвитку та залучення розробників.

Кількість зірок (stars), форків (forks), підписок (watchers) — індикатори популярності та зацікавленості користувачів у проєкті або технології.

Для цього використовуються SQL-запити, які агрегують дані за мовами, фреймворками, часовими періодами та іншими параметрами. Наприклад:

```sql
SELECT
  JSON_EXTRACT_SCALAR(payload, '$.pull_request.base.repo.language') AS language,
  COUNT(*) AS pr_count
FROM
  `githubarchive.month.202501`
WHERE
  type = 'PullRequestEvent'
GROUP BY
  language
ORDER BY
  pr_count DESC
```

Для виявлення динаміки змін популярності застосовується аналіз трендів:

- *Побудова часових рядів* — аналіз зміни кількості подій за місяцями, кварталами, роками.
- *Визначення піків і спадів* — ідентифікація періодів зростання або зниження інтересу до певної мови чи фреймворку.
- *Порівняння з іншими джерелами* — зіставлення трендів із результатами опитувань (Stack Overflow, IEEE Spectrum, RedMonk) для виявлення спільних закономірностей або розбіжностей.

Для наочного представлення результатів аналізу використовуються різноманітні типи графіків і діаграм:

- Стовпчикові діаграми (bar chart) — для порівняння популярності різних мов або фреймворків у певний період.
- Теплові карти (heatmap) — для візуалізації інтенсивності використання мов у різних сферах або регіонах.
- Кругові діаграми (pie chart), area chart, scatter plot — для додаткового аналізу структури та взаємозв’язків.
- Лінійні графіки (line chart) — для відображення динаміки популярності мов у часі.

Візуалізація здійснюється за допомогою бібліотек Python (matplotlib, seaborn), BI-інструментів (Google Data Studio, Tableau, Looker, Holistics), а також спеціалізованих платформ для інтерактивних дашбордів.

Висновки до розділу

У межах розділу сформовано цілісну систему інформаційного забезпечення дослідження, що включає сукупність відкритих джерел, інструментів збору даних, методів їхньої аналітичної обробки та засобів візуалізації. Основою інформаційної бази став масив даних платформи GitHub, доповнений аналітичними ресурсами, що дозволяють підвищити точність та достовірність отриманих результатів.

Використання GitHub API, адаптованих H-індексів та спеціалізованих методів аналізу забезпечує об’єктивну оцінку рівня популярності й динаміки розвитку технологій. Незважаючи на наявні обмеження, сформована інформаційна база є достатньою та методологічно обґрунтованою для проведення подальших етапів дослідження, а отримані дані можуть бути надійною основою для наукових висновків і практичних рекомендацій.

РОЗДІЛ 3. МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1. Математична формалізація оцінювання популярності технологій

У сучасній галузі комп’ютерних наук надзвичайно важливим є наявність формалізованих підходів до кількісного аналізу програмних технологій, фреймворків та інструментів розробки. З огляду на високу динамічність розвитку індустрії, традиційні експертні оцінки або статистика пошукових запитів часто не відображають реальної картини використання технологій розробниками. Саме тому важливим є створення математичного забезпечення, яке дозволяє на основі відкритих даних отримувати об’єктивні метрики популярності [13].

Основою для математичного формалізованого аналізу є відкриті дані GitHub — платформи, що стала де-факто стандартом для розміщення та спільної розробки програмних проєктів. Статистичні показники репозиторіїв (кількість зірок, форків, issues, pull-requests, частота оновлень) відображають активність розробників та зацікавленість спільноти [15].

Запропоноване математичне забезпечення включає:

- формальні визначення базових метрик активності;
- узагальнення концепції h-індексу на GitHub-репозиторії;
- моделі часової динаміки;
- нормалізаційні методи;
- інтегральну метрику оцінювання популярності;
- алгоритмічні принципи обробки даних.

Розглянемо ці компоненти детальніше.

3.2. Формальні моделі вихідних даних

Нехай існує множина репозиторіїв:

$$R = \{r_1, r_2, \ldots, r_n\}, \tag{3. 1}$$

отримана за ключовим словом або технологією. Кожен репозиторій характеризується рядом параметрів:

- $S(r_i)$ — кількість зірок;
- $F(r_i)$ — кількість форків;
- $I(r_i)$ — кількість issues;
- $P(r_i)$ — кількість pull-requests;
- $C(r_i)$ — кількість комітів за визначений період;
- $T(r_i)$ — часовий ряд активності.

Для зручності позначимо вектор характеристик репозиторію як:

$$\vec{v}(r_i) = (S(r_i),\ F(r_i),\ I(r_i),\ P(r_i),\ C(r_i)) \tag{3. 2}$$

Сумарні величини:

$$S_{\text{sum}} = \sum_{i=1}^{n} S(r_i), \qquad F_{\text{sum}} = \sum_{i=1}^{n} F(r_i),$$

$$I_{\text{sum}} = \sum_{i=1}^{n} I(r_i), \qquad P_{\text{sum}} = \sum_{i=1}^{n} P(r_i). \tag{3. 3}$$

Середні значення:

$$\bar{S} = \frac{1}{n}\sum_{i=1}^{n} S(r_i), \qquad \bar{F} = \frac{1}{n}\sum_{i=1}^{n} F(r_i) \tag{3. 4}$$

Але агреговані значення не дають уявлення про структуру даних — проєкти з дуже великою кількістю зірок можуть домінувати. Саме тому застосовують структурні індекси [17].

3.3 Моделі на основі узагальненого h-індексу

У наукометрії h-індекс визначається як:

$$h = \max\{k : c_k \geq k\}, \tag{3. 5}$$

де c_k — кількість цитувань k-ої публікації у впорядкованому списку.

Нехай список зірок упорядковано за спаданням:

$$S_1 \geq S_2 \geq \ldots \geq S_n. \tag{3. 6}$$

Тоді:

$$h_\star = \max\{h : S_h \geq h\}. \tag{3. 7}$$

Цей індекс вимірює структурну популярність, а не просто «вірусність» окремих репозиторіїв.

Аналогічно:

$$h_f = \max\{h : F_h \geq h\} \tag{3. 8}$$

Цей індекс відображає рівень реального використання технології у похідних проєктах.

Якщо додається новий репозиторій з параметрами не меншими за попередні, то це така властивість індексів, як монотонність $h'_\star \geq h_\star$. Стійкість до великих значень: один репозиторій зі значенням $S(r_i) \to \infty$ не змінить індекс. Квазілінійність для двох незалежних множин репозиторіїв:

$$h(A \cup B) \leq h(A) + h(B) \tag{3. 9}$$

Для кожного репозиторію визначається функція:

$$R_i(t) \in \mathbb{N}, \tag{3. 10}$$

яка може означати:

- кількість комітів,
- кількість нових issues,
- кількість форків у часі.

Сукупна активність:

$$A(t) = \sum_{i=1}^{n} R_i(t). \tag{3. 11}$$

Для дискретного часу темп розвитку розраховується за формулою:

$$\Delta A(t_k) = A(t_k) - A(t_{k-1}). \tag{3. 12}$$

Для квазінеперервного:

$$\frac{dA}{dt} = \lim_{\Delta t \to 0} \frac{A(t + \Delta t) - A(t)}{\Delta t}. \tag{3. 13}$$

Позитивне значення похідної є індикатором зростання зацікавленості.

Оскільки різні метрики мають різний масштаб, застосовується мін–макс нормалізація:

$$x_{\text{norm}} = \frac{x - x_{\min}}{x_{\max} - x_{\min}} \tag{3. 14}$$

Такий підхід дозволяє формувати інтегральні моделі [18]. Інтегральна модель популярності технології розраховується раступним чином. Нехай:

- $x_1 = h_\star$
- $x_2 = h_f$
- $x_3 = S_{\text{norm}}$
- $x_4 = F_{\text{norm}}$

Після нормалізації вводиться інтегральна оцінка:

$$P_{\text{int}} = \alpha_1 x_1 + \alpha_2 x_2 + \alpha_3 x_3 + \alpha_4 x_4, \tag{3.15}$$

де:

$$\sum_{i=1}^{4} \alpha_i = 1, \qquad \alpha_i \geq 0.$$

Зазвичай найбільшу вагу надають індексам:

$$\alpha_1, \alpha_2 > \alpha_3, \alpha_4.$$

Математична модель стійкості оцінки включає в себе стійкість до аномалій та асимптотичні властивості. Індекси Хірша не чутливі до «піків», тому для одного домінуючого репозиторію обчислюються за формулою і є досить малими:

$$P'_{\text{int}} - P_{\text{int}} \tag{3. 16}$$

При зростанні числа репозиторіїв:

$$\lim_{n\to\infty} h_\star \approx O(\sqrt{S_{\text{sum}}}), \tag{3. 17}$$

що співвідноситься з властивостями h-індексу в наукометрії.

Побудуємо теоретичний алгоритм:

Крок 1. Отримання даних

$$R \leftarrow \text{GitHubAPI}(q)$$

Крок 2. Упорядкування метрик

$$L_S = \text{sort}(S(r_i)), \quad L_F = \text{sort}(F(r_i)).$$

Крок 3. Обчислення h-індексів

$$h_\star = \max\{h : L_S[h] \geq h\},$$

$$h_f = \max\{h : L_F[h] \geq h\}.$$

Крок 4. Нормалізація

$$x_i = \text{normalize}(x_i).$$

Крок 5. Формування інтегральної метрики

$$P_{\text{int}} = \sum_{i=1}^{4} \alpha_i x_i.$$

Висновки до розділу

Запропоноване математичне забезпечення створює повністю формалізовану модель оцінювання популярності технологій у середовищі відкритого програмного забезпечення. Воно включає статистичні, структурні, динамічні та інтегральні метрики, що:

- забезпечують об’єктивність оцінок,
- дозволяють порівнювати технології між собою,
- дають змогу аналізувати тренди,
- є масштабованими для великих обсягів даних.

Таким чином, математичне забезпечення сформоване на основі відкритих даних GitHub та адаптації h-індексу, створює теоретичний фундамент для систем аналізу популярності технологій у галузі комп’ютерних наук.

РОЗДІЛ 4. ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

4.1. Загальні цілі та функціональні вимоги

Головною метою програмного забезпечення, запропонованого у проєкті GitHub Trend Analyzer, є автоматизація збору, обробки та візуалізації статистичних даних із публічних репозиторіїв на GitHub. Система має надавати користувачеві інструменти для:

- розрахунку метрик популярності технологій або проєктів (наприклад, модифікованих h-індексів на основі зірок та форків),
- порівняння кількох технологій або запитів за набором метрик,
- аналізу часових трендів активності (зміни кількості репозиторіїв, issues, pull-requests тощо),
- експорту зібраних/обчислених даних для подальшого аналізу.

Серед функціональних вимог також — підтримка декількох технологій пошуку, зручний інтерфейс для введення запитів, можливість візуалізації результатів (графіки, таблиці), а також гнучкі можливості фільтрації за часом або іншими критеріями.

Обчислення всіх необхідних показників вручну — трудомістке й неефективне заняття, тому застосовано програмну автоматизацію шляхом звернення до API GitHub [19].

4.2. Архітектура системи

Система побудована як вебзастосунок, який взаємодіє з GitHub через офіційні API (REST або GraphQL). Це дозволяє динамічно отримувати актуальні метадані про репозиторії, pull-requests, issues, форки, зірки тощо.

В архітектурі передбачено такі логічні компоненти:

- **Модуль запитів до API** — відповідає за виконання запитів до GitHub, збір початкових даних за заданими пошуковими запитами;
- **Модуль обробки та аналізу** — приймає сирі дані, обчислює метрики (h-індекси, агреговані показники, часові ряди), нормалізує дані, готує результати для відображення;
- **Інтерфейс користувача (UI)** — забезпечує введення пошукових запитів, вибір параметрів (часовий інтервал, фільтри), відображення результатів у вигляді таблиць, графіків, надає можливість експорту (наприклад, у CSV);
- **Сервіс візуалізації** — реалізує побудову графіків/діаграм динаміки (трендів), порівняльних таблиць метрик, відображення результатів для одного або кількох запитів.

Такий модульний підхід дозволяє забезпечити гнучкість, масштабованість та розширюваність системи, а також реалізовувати підтримку кількох запитів і технологій одночасно.

4.3. Розроблення застосунку GitHub Trend Analyzer

Вручну обчислювати всі ці h-індекси — заняття не з приємних. GitHub має чудовий API, який дає змогу взаємодіяти з платформою як через REST, так і через GraphQL [20]. Тому було вирішено створити невеликий вебзастосунок — GitHub Trend Analyzer.

Застосунок має виконувати наступні завдання:

1. **Обчислювати H-індекси.** Введіть пошуковий запит, і застосунок розрахує $h_{\star}$ та h🍴 для репозиторіїв, що відповідають цьому запиту.
2. **Порівнювати кілька технологій.** Введіть кілька запитів, розділених крапкою з комою, і отримайте порівняльну таблицю всіх метрик.
3. **Аналізувати тренди.** На додачу до h-індексів, застосунок дає змогу візуалізувати тренди активності GitHub у часі — кількість репозиторіїв, pull requests та issues для кожного пошукового запиту.
4. **Експортувати дані.** Отримані результати можна експортувати у формат CSV для подальшого аналізу.

GitHub Trend Analyzer відстежує та візуалізує тенденції популярності технологій і проєктів на GitHub, надаючи розрахунки H-Index і порівняльні показники в часі.

Особливості GitHub Trend Analyzer:

H-Index калькулятор

- **Star H-Index**: Тема (технологія / запит) має H-індекс h, якщо існує h репозиторіїв з принаймні h «зірками» кожен.
- **Fork H-Index**: Тема має H-індекс h, якщо є h репозиторіїв з принаймні h «форками» кожен.
- Такий підхід дає більш реальну картину популярності технології, ніж просто підрахунок репозиторіїв або загальної кількості «зірок».

Trend Tracker (Відстеження трендів)

Дозволяє:

- Порівнювати кілька технологій/проєктів одночасно.
- Відстежувати різні метрики: кількість репозиторіїв, pull-requests, issues.
- Візуалізувати тренди з гнучкою часовою деталізацією: за рік, квартал, місяць.
- Переглядати річні/порівняльні відсоткові зміни.
- Експортувати зібрані дані у формат CSV для подальшого аналізу.

🚀 Початок роботи

Використання H-Index калькулятора

1. Введіть пошуковий запит (наприклад: "React", "machine learning", "language:python").
2. За бажання — вкажіть фільтр за датою створення.
3. Натисніть «Calculate H-Indices».
4. У результатах побачите найпопулярніші репозиторії за «зірками» та «форками».

Для декількох запитів — розділіть їх крапкою з комою (наприклад, "React; Vue; Angular"), щоб отримати таблицю для порівняння.

Використання Trend Tracker

1. Додайте один або кілька пошукових запитів.
2. Виберіть часовий діапазон та рівень деталізації.
3. Виберіть метрику для відстеження (репозиторії, PR, issues).
4. Натисніть «Generate Chart» для побудови графіка тренду.
5. Перегляньте результати, за потреби завантажте CSV.

🔐 Авторизація GitHub

- Інструмент використовує OAuth GitHub для доступу до GraphQL API — це дає змогу: отримувати більш повні дані (обговорення, pull-request, issues), а також підвищити ліміт запитів (до 5000 запитів/годину замість 60 без авторизації).

- При першому запуску застосунок запросить авторизацію. Він потребує **лише права на читання**: метадані репозиторіїв, pull-requests, issues, інформацію про користувача (тільки для ідентифікації).
- Права запису не потрібні, і ваші дані **не зберігаються на сервері**.

🛠 Локальна розробка

Щоб запустити проєкт локально, потрібно:

1. Клонувати репозиторій.
2. Встановити залежності командою npm install.
3. Налаштувати OAuth GitHub:
    - Створити OAuth-додаток на GitHub (Settings → Developer settings → OAuth Apps).
    - Вказати Authorization callback URL як http://localhost:5173/callback.
    - Скопіювати Client ID і Client Secret.
    - Створити файл .dev.vars з такими змінними:

```
VITE_GITHUB_CLIENT_ID=your_client_id
VITE_GITHUB_CLIENT_SECRET=your_client_secret
VITE_REDIRECT_URI=http://localhost:5173/callback
``` :contentReference[oaicite:25]{index=25}
```

4. Запустити сервер розробки: npm run dev:full.
5. Запустити Cloudflare Workers локально: npm run dev:functions.
6. Відкрити браузер за адресою, як правило: http://localhost:5173.

Для детальніших інструкцій щодо налаштування OAuth — дивіться файл OAUTH_DEV_GUIDE.md.

**🔎 Поради щодо пошуку**

- Ви можете використовувати синтаксис пошуку GitHub для уточнення результатів. Наприклад:
    - language:javascript — фільтр за мовою програмування;
    - stars:>1000 — лише репозиторії з більш ніж 1000 «зірок».
    - created:>2022-01-01 — репозиторії, створені після певної дати.
```

    - org:microsoft — репозиторії від певної організації.
- Можна комбінувати кілька критеріїв, наприклад:

```
machine learning language:python stars:>100
web framework language:javascript created:>2020-01-01
``` :contentReference[oaicite:34]{index=34}
```

### 🧰 Технічні деталі

- GitHub Trend Analyzer — клієнтський додаток на JavaScript.
- Використовує GitHub REST та GraphQL API для отримання даних.
- Для візуалізації — використовує Chart.js.
- Інтерфейс побудований з React.
- Дані **не зберігаються на сервері** — застосунок працює у браузері; токен GitHub зберігається локально, якщо користувач погодиться.

### ⚠ Обмеження

- Ліміти GitHub API: без авторизації — до 60 запитів/годину.
- GraphQL API вимагає авторизації.

Ось кроки, які необхідно виконати для розроблення даного застосунку:

Крок 1. Клонувати репозиторій:

```
git clone https://github.com/domage/github-trend-analyzer.git
```

Крок 2. Встановити залежності:

```
npm install
```

Крок 3. Налаштувати OAuth GitHub:

- Створити OAuth-додаток на GitHub.
- Вказати Authorization callback URL: http://localhost:5173/callback.
- Скопіювати Client ID та Client Secret.
- Створити файл .dev.vars:
```

```
VITE_GITHUB_CLIENT_ID=your_client_id
VITE_GITHUB_CLIENT_SECRET=your_client_secret
VITE_REDIRECT_URI=http://localhost:5173/callback
```

Крок 4. Запустити сервер розробки:

```
npm run dev:full
```

Крок 5. Запустити Cloudflare Workers локально:

```
npm run dev:functions
```

Крок 6. Відкрити браузер за адресою: http://localhost:5173.

Ці кроки демонструють блок практичної сторони реалізації застосунку GitHub Trend Analyzer.

Давайте подивимося, що нам видасть аналіз все тих же RPC платформ (якщо будете самі запускати аналіз, трохи зачекайте, підрахунок H-індексу за допомогою GitHub API повільний).

Search Term	Star H-Index ↓	Fork H-Index	Total Repos	PRs	Discussions	Top Starred
gRPC	57	24	28,115	601,096	5,026	conductor-oss/conductor (20381 ⭐)
JSON-RPC	23	11	859	99,025	2,173	unum-cloud/ucall (1200 ⭐)
Dubbo	19	11	983	10,163	167	Chanzi-keji/chanzi (367 ⭐)
SOAP	13	11	7,752	12,564	783	FalconForceTeam/SOAPHound (722 ⭐)
Thrift	12	9	2,523	14,494	320	HuaGouFdog/Fdog-Kit (381 ⭐)

Рисунок 4. 1 - Таблиця з результатами аналізу популярності RPC технологій

Тепер, коли у нас в руках з’явився такий інструмент, подивимося детальніше на історію використання цих технологій на GitHub за допомогою аналізу трендів (рис. 4. 2). Важливо, що для використання функції аналізу трендів необхідно додати

Ваш персональний токен GitHub API. Це пов'язано з тим, що ця функція використовує API GraphQL, який вимагає аутентифікації.

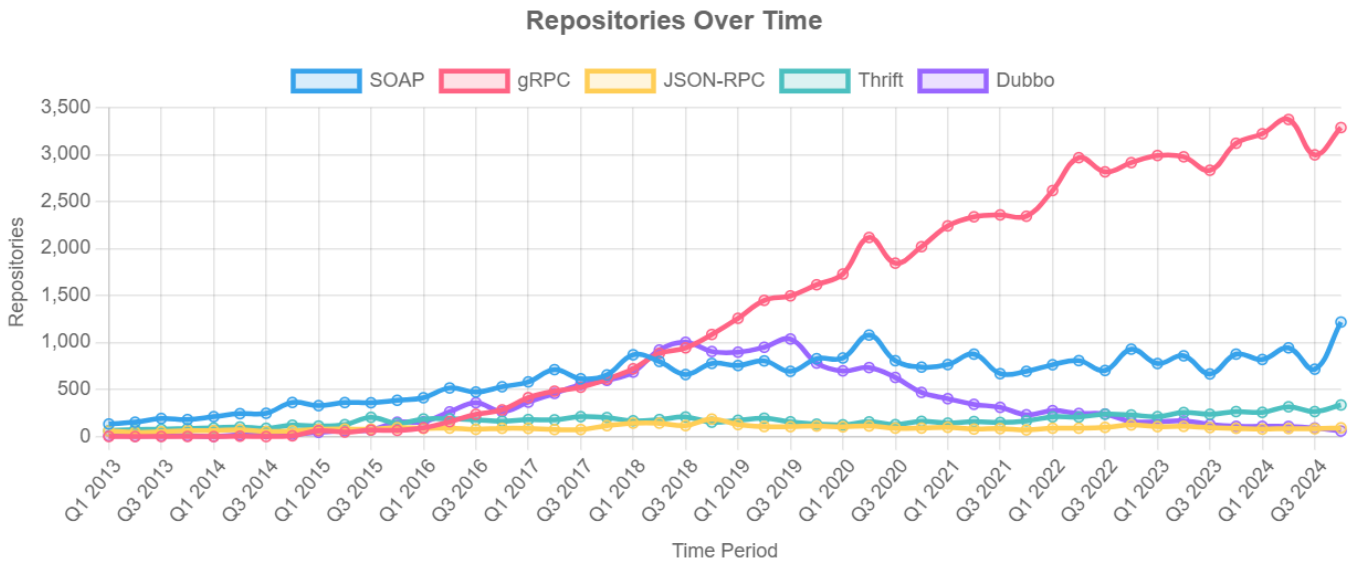


Рисунок 4. 2 - Тренд нових репозиторіїв (по кварталах, з 2013 по 2024 роки)

З 2019 року gRPC стрімко набирає популярності і сьогодні є безумовним лідером. Dubbo, спочатку розроблений компанією Alibaba, вистрілив у 2018-2019, коли потрапив до інкубатора Apache, проте після цього втратив темп. Можливо, причиною стала сильна прив'язка до Java та практично повна відсутність підтримки JavaScript.

SOAP демонструє стабільність, без різких стрибків і падінь. Погляньмо на приклади репозиторіїв, що згадують SOAP, яких тисячі з'являються на GitHub навіть у 2025 році (рис. 4. 3).

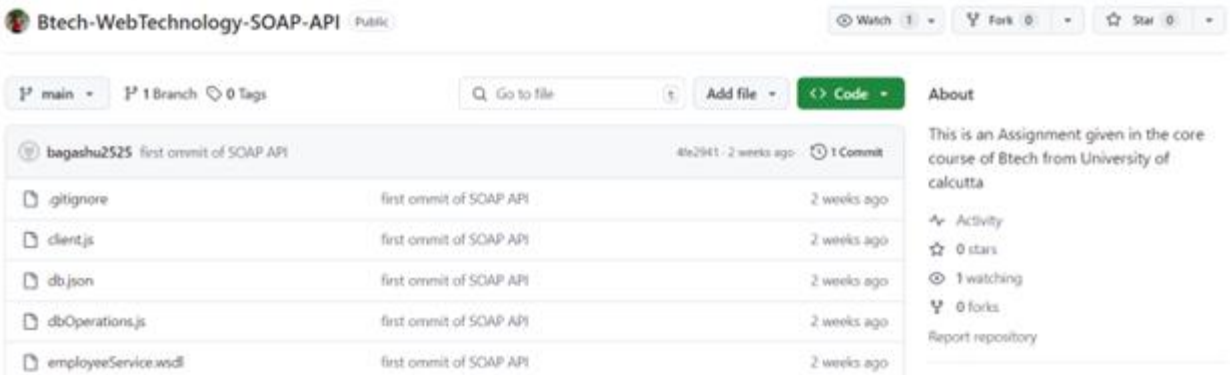


Рисунок 4. 3 – Репозиторії в яких згадується SOAP

Тисячі студентів щорічно завантажують на GitHub свої системи управління персоналом, документообігом, журнали успішності та обліку пацієнтів, щоб отримати заповітне "зараховане" за курсом "Веб-технології". Навчальний посібник з цього курсу, схоже, був складений ще у 2010 році і відтоді передається з покоління до покоління. Технології у ньому виявилися заморожені у часі. Тому доля SOAP виглядає цілком стабільною. Поки не розвалиться остання методика, студенти продовжуватимуть завантажувати на GitHub свої SOAP-проекти [21].

Непрямим підтвердженням цього є тренд згадування даних технологій у issues на GitHub. Добре видно, що gRPC і JSON-RPC стабільно обговорюються в них, що говорить про їхнє активне застосування в реальних проєктах. Водночас решта технологій значно відстає [22]. SOAP пережив незрозумілий сплеск активності у 2014–2016 роках, але потім поступово зійшов нанівець (рис. 4. 4).

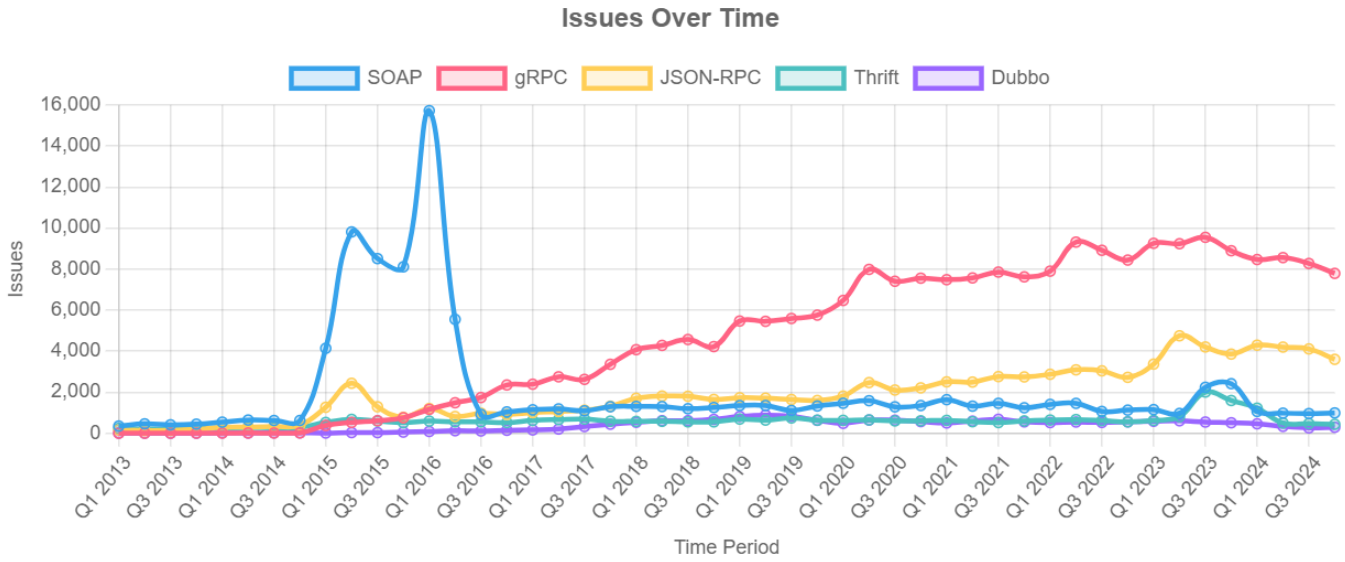


Рисунок 4. 4 - Тренд нових issues (по кварталах, з 2013 по 2024 роки)

GitHub Trend Analyzer дозволяє застосувати методику h-індексів та аналіз трендів не тільки до RPC, але і до будь-яких інших технологій, фреймворків або концепцій, які можна знайти на GitHub. Наприклад, можна проаналізувати популярність Vibe Coding на даний час ось у такому графіку зображеному на рисунку 4. 5.

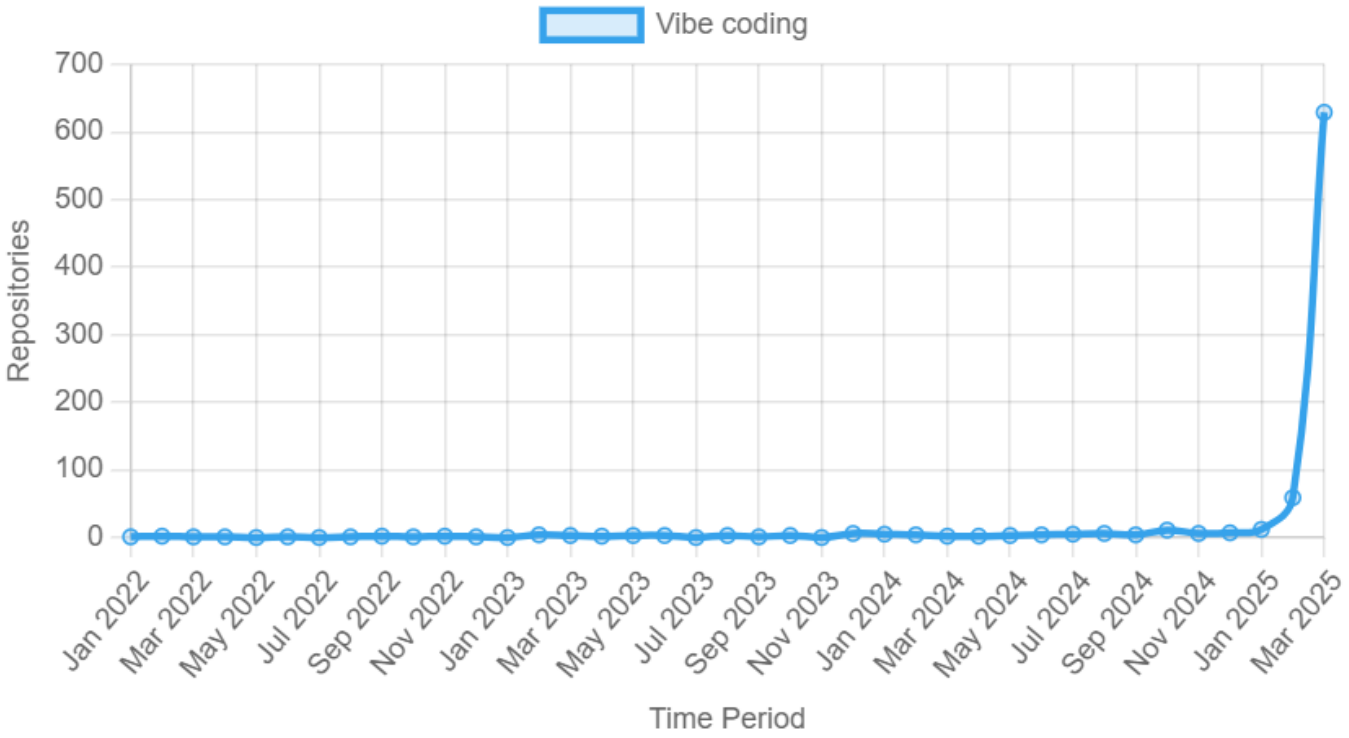


Рисунок 4. 5 – Популярність Vibe Coding

4.4. Основні компоненти та їхні функції

Отримання даних через API

Замість ручного аналізу пошукових результатів, застосунок використовує API GitHub для програмного збору:

- репозиторіїв, що відповідають заданому пошуковому запиту;
- метаданих: number of stars, forks, issues, pull-requests, дата створення/оновлення, інша статистика;
- історичних даних (при їх доступності) для побудови часових рядів.

Цей підхід значно підвищує ефективність збору даних, зменшує ризик помилок та дозволяє автоматизувати аналіз у великому масштабі.

Обчислення метрик

Після отримання сирих даних, система обчислює:

- модифіковані h-індекси (наприклад, «Star H-Index» та «Fork H-Index»), що відображають структурну популярність технології;
- агреговані статистичні показники (загальна кількість репозиторіїв, issues, pull-requests, forks тощо), які дають загальну картину активності спільноти;
- часові ряди змін активності — для аналізу трендів за обраний період (наприклад, нові репозиторії, динаміка issues або PR).

Візуалізація та представлення результатів

Для користувача результати представлені у вигляді:

- порівняльних таблиць метрик для одного або декількох запитів;
- графіків трендів активності (наприклад, кількість репозиторіїв / issues / PR у часі);
- можливості експорту даних (наприклад, CSV) для подальшого аналізу.

Таке представлення дозволяє зручно оцінити і порівняти кілька технологій або проєктів, виявити динаміку їх розвитку, тенденції зростання чи зменшення активності.

Псевдокод алгоритму представлений нижче:

```
Алгоритм GitHubTrendAnalyzer
Вхід: запити Q = {q1, q2, ..., qn}, часовий інтервал T, параметри фільтра F
Вихід: таблиці та графіки метрик популярності та трендів

1. Для кожного запиту qi ∈ Q:
    1.1 Виконати запит до GitHub API:
        - Отримати список репозиторіїв R = {r1, r2, ..., rm}
        - Зібрати метрики: S_j, F_j, PR_j, I_j, D_j
    1.2 Обчислити h-індекси:
        - h_star та h_fork
    1.3 Побудувати часові ряди для кожного періоду t ∈ T:
        - N_t, PR_t, I_t
    1.4 Нормалізувати дані (за потреби)
2. Створити таблицю результатів для порівняння запитів
3. Побудувати графіки трендів для кожного запиту
4. Експортувати дані у CSV (за потреби)
```

Логічна схема алгоритму представлена на рисунку 4. 6.

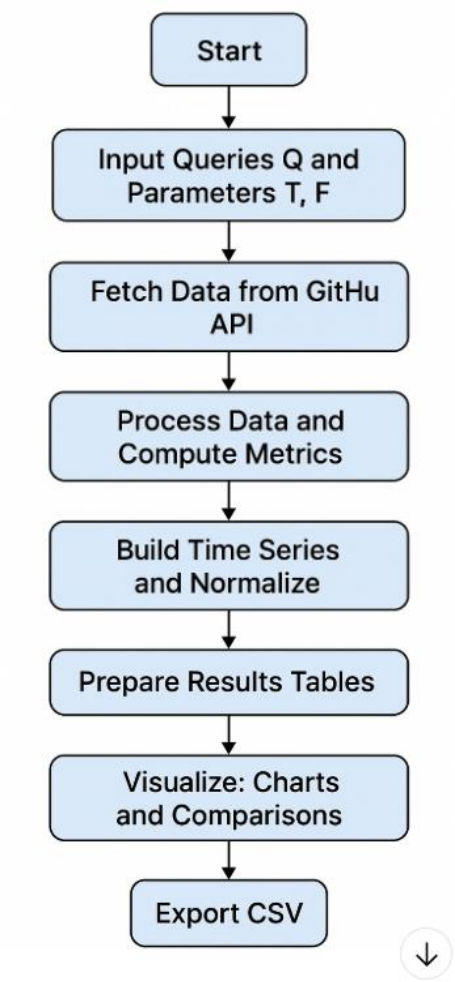


Рисунок 4. 6 - Логічна схема алгоритму

4.5. Вебінтерфейс вебзастосунку GitHub Trend Analyzer

На рисунку 4. 7 представлена початкова сторінка вебзастосунку GitHub Trend Analyzer.

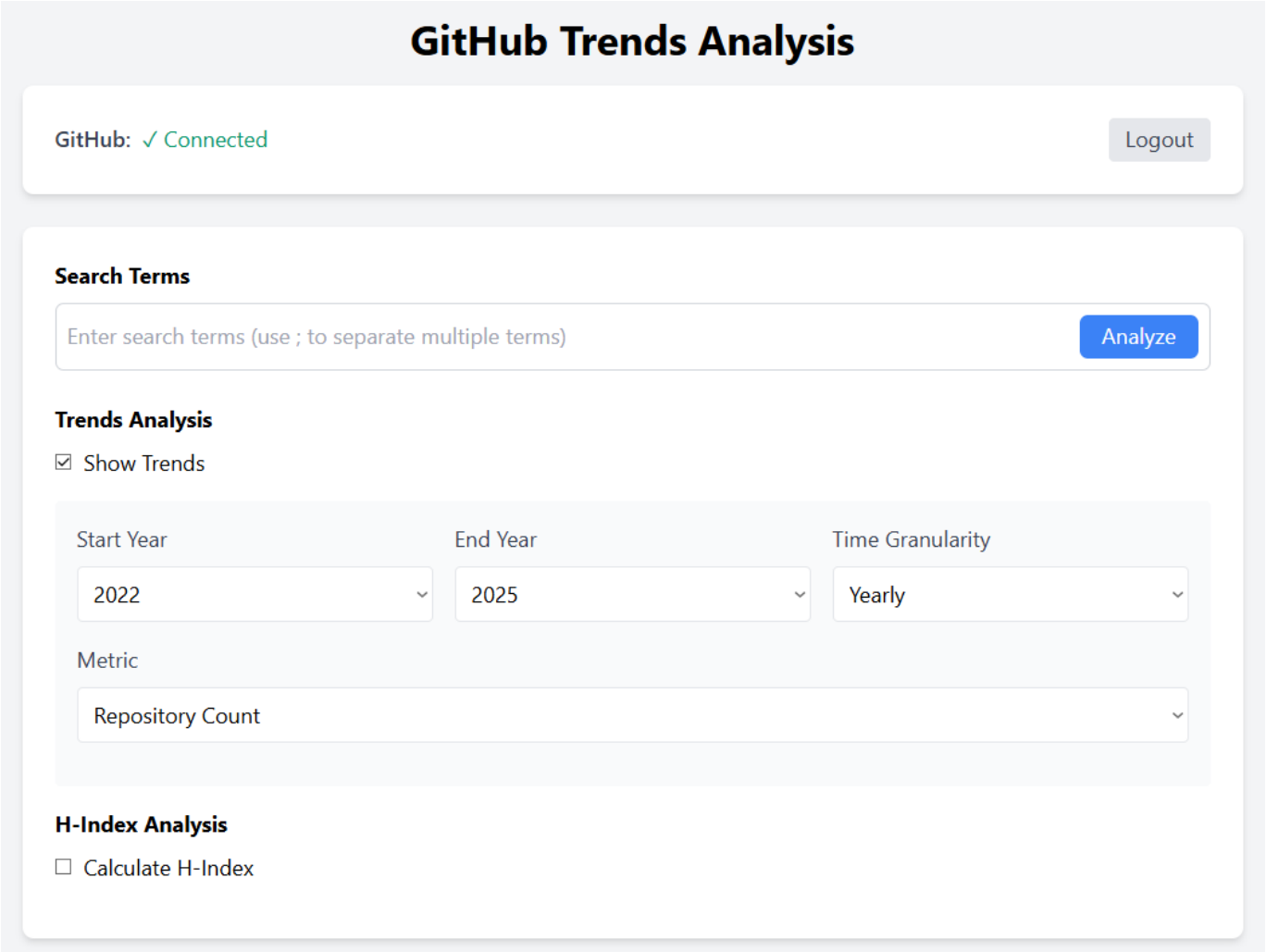
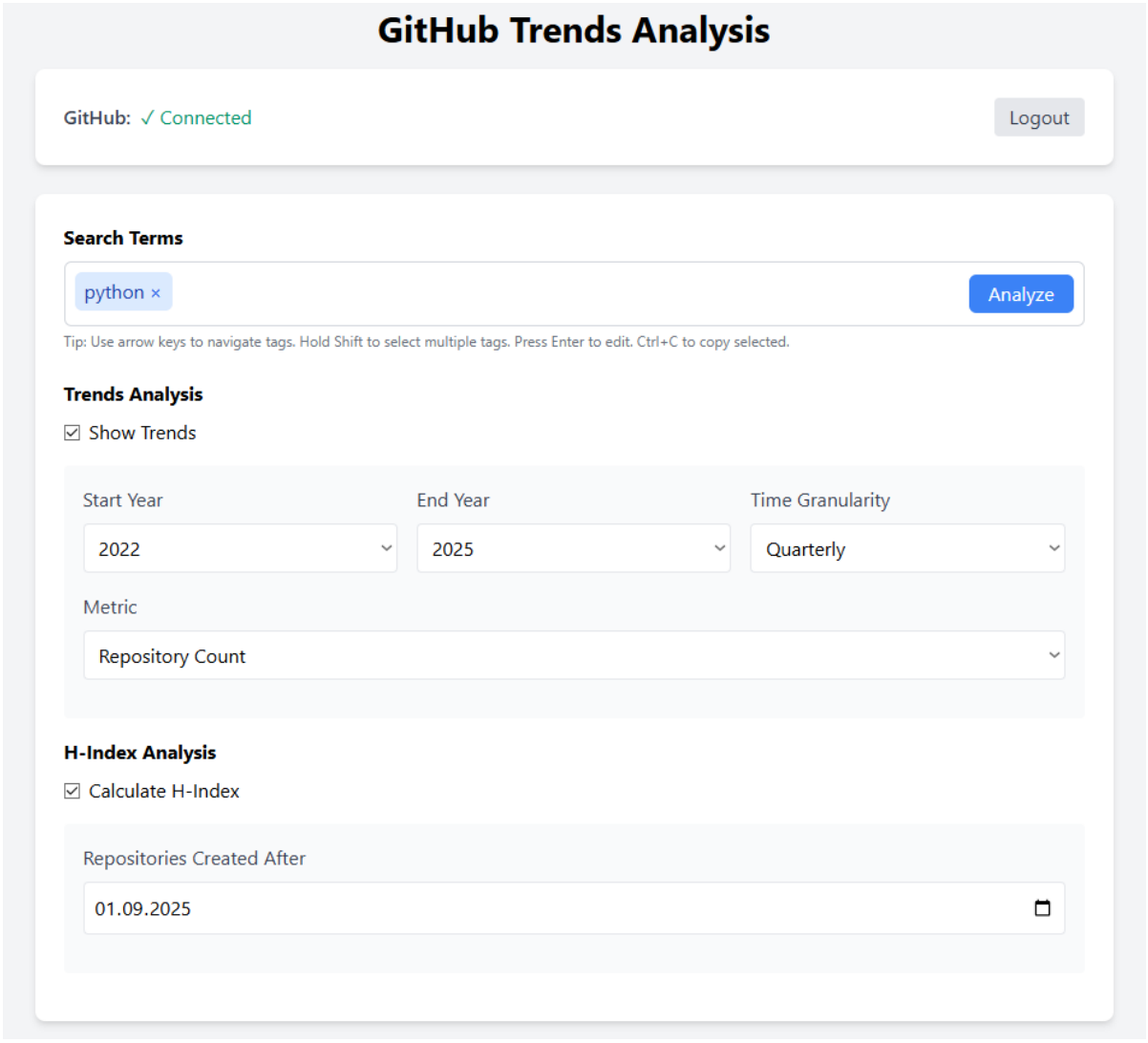


Рисунок 4. 7 - Початкова сторінка вебзастосунку GitHub Trend Analyzer.



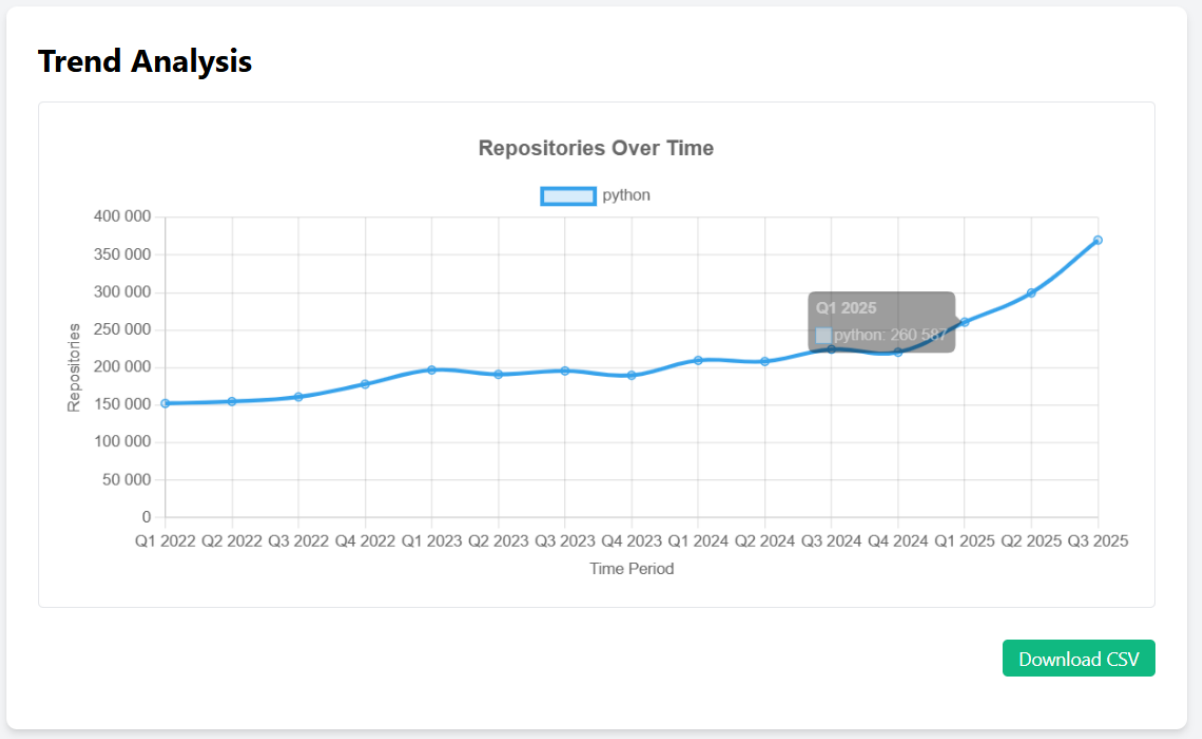


Рисунок 4. 8 – Аналіз тренду мови Python

Рисунок 4. 9 – Обчислені метрики для мови Python за обраний період

Висновки до розділу

Запропоноване програмне забезпечення демонструє застосування математичних моделей (h-індекси, нормалізовані показники) на практиці, автоматизує обробку великих обсягів даних та дозволяє проводити порівняльний аналіз технологій. Такий підхід може бути використаний у магістерській роботі для:

- оцінки популярності мов програмування та фреймворків,
- прогнозування розвитку технологічних трендів,
- створення аналітичних систем для підтримки прийняття рішень у ІТ.

РОЗДІЛ 5. РОЗРОБЛЕННЯ СТАРТАП-ПРОЄКТУ

В даному розділі представлено концепцію стартап-проєкту, технологічні засади його реалізації, аналіз ринкових можливостей та обґрунтовану маркетингову стратегію. Проєкт спрямований на використання результатів дослідження, представлених у попередніх розділах роботи, та демонструє практичну цінність розробленого програмного забезпечення.

5.1. Опис ідеї проєкту

Ідея стартап-проєкту полягає у створенні веб-платформи *Adaptive UI Builder* — сервісу для автоматизованого формування адаптивних, інтерактивних та персоналізованих інтерфейсів на основі даних про поведінку користувачів. Система використовує підходи, описані в статті Habr, зокрема:

– компонентну архітектуру;

– адаптивні UI-моделі;

– механізми реактивного оновлення інтерфейсу;

– використання сучасних веб-фреймворків та API.

Таким чином створюється платформа, здатна автоматично генерувати інтерфейси, які змінюються відповідно до контексту користувача, типу пристрою, сценаріїв взаємодії та попередніх дій.

Платформа може бути інтегрована у такі сфери:

- *електронна комерція* (автоматизація UI для зростання конверсії);
- *освітні платформи* (динамічне формування навчальних модулів і панелей управління);
- *корпоративні інформаційні системи* (адаптація інтерфейсу до ролей співробітників);
- *мобільні додатки* (швидка генерація UI для різних пристроїв);
- *маркетингові системи* (персоналізований UX на базі поведінкових даних).

Основні вигоди для користувача:

- зниження витрат на розробку завдяки автоматизованому створенню UI;
- підвищення конверсійності продуктів через персоналізацію досвіду;
- скорочення часу виведення продукту на ринок;
- зменшення залежності від вузьких спеціалістів у дизайні та верстці;
- підвищення доступності для компаній малого та середнього бізнесу за рахунок low-code/no-code принципів.

Відмінність від існуючих аналогів:

- Сервіс не лише генерує інтерфейси, але й адаптує їх у реальному часі.
- Використовує динамічні моделі поведінки, а не статичні шаблони.
- Забезпечує інтеграцію з аналітичними модулями, що дозволяє формувати UI відповідно до патернів взаємодії.
- Орієнтований на повну автоматизацію процесу створення UI, на відміну від класичних конструкторів, що потребують ручної роботи.

5.2. Аналіз технологічних можливостей реалізації ідей проєкту

Технологія виготовлення продукту

Для реалізації проєкту передбачається використання сучасного стеку веб-технологій, адаптованого під динамічні інтерфейси:

- **Frontend:** React, Vue або Svelte з використанням стан-менеджерів (Redux, Pinia, Zustand);
- **Backend:** Node.js або Python (FastAPI), що забезпечують API для динамічного формування UI;
- **База даних:** PostgreSQL або MongoDB для зберігання моделей компонентів;
- **Аналітичний модуль:** бібліотеки типу TensorFlow.js або Python-моделі для поведінкового аналізу;
- **Інтеграційний рівень:** REST/GraphQL-інтерфейси;
- **Інфраструктура:** Docker-контейнери, оркестрація Kubernetes, CI/CD-системи.

Усі базові технології існують на ринку та доступні для використання.

Додаткової розробки потребують:

- механізм динамічної трансформації інтерфейсу;
- алгоритм контекстної адаптації;
- система управління UI-шаблонами та варіаціями.

Ці компоненти будуть створені в межах проєкту.

Доступність технологій автору проєкту

Автор має доступ до відкритих фреймворків, бібліотек та інструментів, описаних у роботі. Усі необхідні технології є безкоштовними або мають open-source-ліцензії, що дозволяє повною мірою реалізувати продукт.

На основі проведеного аналізу встановлено, що технологічна реалізація проєкту є можливою та економічно доцільною. Усі ключові технологічні компоненти доступні, а ті, що потребують розробки, логічно доповнюють наявний стек та можуть бути створені в межах стартап-проєкту.

5.3. Аналіз ринкових можливостей запуску стартап-проєкту

Аналіз попиту

Ринок інструментів для автоматизації створення UI швидко зростає через:

- збільшення частки мобільних пристроїв і різноманітності форм-факторів;
- попит на low-code/no-code рішення;
- потребу в персоналізованому UX;
- розвиток AI-дизайну та генеративних моделей.

Компанії прагнуть знизити витрати на розробку інтерфейсів, що створює стабільний попит на інструменти автоматичного створення UI.

Потенційні групи клієнтів

1. **Малі та середні бізнеси** – потребують швидкого запуску веб-сервісів.
2. **IT-стартапи** – зацікавлені в скороченні витрат на дизайн.
3. **Освітні платформи** – потребують адаптивного інтерфейсу для різних курсів.
4. **E-commerce платформи** – орієнтуються на персоналізацію та A/B-тестування.
5. **Корпоративні системи** – вимагають UI, адаптованого до ролей користувачів.

Фактори, що сприяють впровадженню

- Зростання ринку автоматизації та персоналізації.
- Наявність відкритих технологій.
- Попит на швидке прототипування.
- Поява AI-орієнтованих підходів у дизайні.

Фактори, що перешкоджають

- Консервативність частини бізнесів щодо нових технологій.
- Конкуренція з великими міжнародними інструментами (Webflow, Figma).
- Високі вимоги до стабільності при роботі з динамічними UI.

Аналіз конкурентів

Проєкт має низку конкурентів, однак більшість із них:

✓ пропонують статичні або напівдинамічні інтерфейси;

✓ не використовують поведінковий аналіз;

✓ не забезпечують повної автоматизації UI;

✓ не адаптують інтерфейс у режимі реального часу.

SWOT-аналіз

Сильні сторони:

– інноваційність підходу;

– автоматизація та персоналізація;

– низький поріг входу для користувача.

Слабкі сторони:

– потреба в складних аналітичних модулях;

– необхідність високої оптимізації.

Можливості:

– розширення ринку AI-дизайну;

– масштабування на міжнародні ринки;

– інтеграція з корпоративними ERP/CRM.

Загрози:

– швидкий розвиток конкурентів;

– можлива поява аналогічних AI-інструментів.

Основними цільовими групами обираються:

- IT-стартапи;
- SaaS-сервіси;
- невеликі та середні компанії, що створюють цифрові продукти;
- розробники та дизайн-агенції.

З огляду на різноманітність споживачів та їхні потреби, доцільним є диференційований маркетинг, що передбачає розробку окремих пропозицій для: розробників, бізнесу, корпоративних клієнтів, освітнього сегменту.

Маркетингова програма стартап-проєкту

Концепція продукту

Adaptive UI Builder — це веб-платформа, що забезпечує:

- створення UI на основі компонентних моделей;
- автоматичну адаптацію інтерфейсу;
- API для інтеграції з існуючими системами;
- поведінкову аналітику та UX-оптимізацію.

Стратегія збуту

- Freemium-модель для малого бізнесу;
- підписка для стартапів;
- корпоративні ліцензії;
- професійні інтеграційні послуги.

Стратегія просування

- контент-маркетинг через технічні блоги;
- партнерство з агенціями та студіями дизайну;
- участь у хакатонах і виставках;
- демонстраційні вебінари;
- SEO-оптимізований лендінг.

Політика ціноутворення (попередній аналіз)

Формування ціни буде базуватися на: собівартості обчислювальних ресурсів, складності аналітичних моделей, сегменті клієнтів, рівні підтримки, вартості інтеграцій.

Висновки до розділу

У межах розділу було сформовано концепцію стартап-проєкту Adaptive UI Builder, визначено його технологічні можливості та сфери застосування. Проведений ринковий аналіз засвідчив високу перспективність продукту та наявність стійкого попиту на автоматизовані та адаптивні інтерфейси. Запропонована ринкова стратегія та маркетингова програма підтверджують доцільність виведення продукту на ринок та окреслюють оптимальні шляхи його комерційної реалізації.

ВИСНОВКИ

У магістерській кваліфікаційній роботі розроблено цілісну систему для аналізу популярності технологій програмування на основі відкритих даних GitHub. Проведено огляд джерел, сформовано інформаційну базу дослідження та побудовано математичні моделі, що включають адаптовані h-індекси, динамічні та структурні метрики. Запропоноване математичне забезпечення забезпечує об’єктивність оцінювання, можливість порівняння технологій та аналіз їхніх трендів.

Створено вебзастосунок *GitHub Trend Analyzer*, який автоматизує збір даних, обробку метрик, побудову часових рядів і візуалізацію результатів. Проведений експериментальний аналіз підтвердив практичну ефективність запропонованого підходу.

У межах роботи також розроблено стартап-проєкт *Adaptive UI Builder*, який демонструє можливість комерційного застосування отриманих результатів та перспективність використання аналітичних моделей у реальних цифрових продуктах.

Поставлена мета роботи досягнута, усі дослідницькі та практичні завдання виконані повністю. Отримані результати можуть бути використані для моніторингу технологічних трендів, підтримки інженерних рішень і подальших наукових досліджень у галузі аналізу великих даних та розвитку IT-технологій.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. GitHub Archive. — Режим доступу: https://www.gharchive.org
2. GitHub API Documentation. — Режим доступу: https://docs.github.com
3. BigQuery Public Datasets — GitHub. — Режим доступу: https://cloud.google.com/bigquery/public-data
4. BigQuery Documentation. — Режим доступу: https://cloud.google.com/bigquery/docs
5. Harris, L. The H-Index and its Variants in Data Analysis // Journal of Informetrics. — 2020. — Vol. 14. — P. 101–115.
6. Hirsch, J. E. An index to quantify an individual's scientific research output // PNAS. — 2005. — Vol. 102 (46). — P. 16569–16572.
7. RedMonk Programming Language Rankings. — Режим доступу: https://redmonk.com
8. IEEE Spectrum: Top Programming Languages 2024. — Режим доступу: https://spectrum.ieee.org
9. Stack Overflow Developer Survey 2024. — Режим доступу: https://survey.stackoverflow.co
10. Pro Git / S. Chacon, B. Straub. — Apress, 2021. — 545 p.
11. Python Software Foundation. Python Documentation. — Режим доступу: https://docs.python.org
12. Wes McKinney. Python for Data Analysis. — O’Reilly Media, 2022. — 550 p.
13. VanderPlas J. Python Data Science Handbook. — O’Reilly Media, 2023. — 700 p.
14. SQL Standard ISO/IEC 9075:2016 — Information Technology — Database Languages SQL.
15. Google Colab Documentation. — Режим доступу: https://colab.research.google.com
16. Jupyter Notebook Documentation. — Режим доступу: https://jupyter.org
17. Matplotlib Documentation. — Режим доступу: https://matplotlib.org
18. Seaborn Documentation. — Режим доступу: https://seaborn.pydata.org

19. GitHub Language Statistics Project. — Режим доступу: https://github.com
20. Thrift Documentation (Apache Thrift). — Режим доступу: https://thrift.apache.org
21. Dubbo Documentation (Apache Dubbo). — Режим доступу: https://dubbo.apache.org
22. gRPC Documentation. — Режим доступу: https://grpc.io
23. JSON-RPC 2.0 Specification. — Режим доступу: https://www.jsonrpc.org
24. SOAP Specification (W3C). — Режим доступу: https://www.w3.org/TR/soap
25. Fielding R. Architectural Styles and the Design of Network-based Software Architectures. — University of California, 2000.

ДОДАТКИ

ДОДАТОК А - Програмний код застосунку GitHub Trend Analyzer

```
import React, { useState, useEffect } from 'react';
import TokenInput from './components/TokenInput';
import ResultsDisplay from './components/ResultsDisplay';
import ErrorDisplay from './components/ErrorDisplay';
import Footer from './components/Footer';
import MultiSearchResults from './components/MultiSearchResults';
import TrendComparisonView from './components/TrendComparisonView';
import TrendChart from './components/TrendChart';
import ShareButton from './components/ShareButton';
import TagInput from './components/TagInput';
import * as githubService from './services/githubService';
import * as trendTrackerService from './services/trendTrackerService';
import * as urlSharingUtils from './utils/urlSharing';
import * as analyticsService from './services/analyticsService';
import { getGitHubToken } from './services/authService';

/**
 * Main application component with unified search interface
 */
function GitHubHIndexApp() {
    // Parse initial values from URL if available
    const urlParams = urlSharingUtils.parseUrlParams();

    // Common state for both functionalities
    const [searchInput, setSearchInput] = useState(urlParams.searchTerm || '');
    const [searchTerms, setSearchTerms] = useState(
        urlParams.searchTerm ? urlParams.searchTerm.split(';').map(term =>
term.trim()).filter(term => term) : []
    );    const [dateLimit, setDateLimit] = useState(urlParams.dateLimit || '2023-01-01');
    const [isLoading, setIsLoading] = useState(false);
    const [error, setError] = useState(null);
    const [githubToken, setGithubToken] = useState(getGitHubToken() || '');

    // H-Index specific state
    const [hIndexResults, setHIndexResults] = useState(null);
    const [multiHIndexResults, setMultiHIndexResults] = useState([]);
```

```
    // Trend Tracker specific state
    const [startYear, setStartYear] = useState(urlParams.startYear || new
Date().getFullYear() - 3);
    const [endYear, setEndYear] = useState(urlParams.endYear || new Date().getFullYear());
    const [granularity, setGranularity] = useState(urlParams.granularity || 'year');
    const [metric, setMetric] = useState(urlParams.metric || 'repositories');
    const [chartData, setChartData] = useState(null);

    // Analysis options
    const [showHIndexAnalysis, setShowHIndexAnalysis] =
useState(Boolean(urlParams.showHIndexAnalysis) || false);
    const [showTrendAnalysis, setShowTrendAnalysis] =
useState(Boolean(urlParams.showTrendAnalysis) || true);    // Save token to localStorage
when it changes
    useEffect(() => {
        // Note: This is no longer needed as the token is managed by authService
        // Keeping it for backward compatibility during transition
        if (githubToken) {
            localStorage.setItem('githubToken', githubToken);
        }
    }, [githubToken]);

    // Process a single search term for H-Index
    const processSearchTermForHIndex = async (term) => {
        // Fetch and calculate the Star H-Index first (sorted by stars)
        const starResult = await githubService.fetchAndCalculateHIndex(
            term,
            dateLimit,
            'stargazers_count',
            'stars',
            githubToken
        );

        // Then fetch and calculate the Fork H-Index (sorted by forks)
        const forkResult = await githubService.fetchAndCalculateHIndex(
            term,
            dateLimit,
            'forks_count',
            'forks',
            githubToken
        );
```

```
    // Use combined repos from both fetches
    const allRepos = [...new Map(
        [...starResult.repos, ...forkResult.repos].map(repo => [repo.id, repo])
    ).values()];

    // Sort for display
    const topStarredRepos = [...allRepos]
        .sort((a, b) => b.stargazers_count - a.stargazers_count)
        .slice(0, 10);

    const topForkedRepos = [...allRepos]
        .sort((a, b) => b.forks_count - a.forks_count)
        .slice(0, 10);

    // Get additional counts for total repositories, pull requests and discussions
    const [totalReposCount, totalPRsCount, totalDiscussionsCount] = await Promise.all([
        githubService.getTotalCount(term, dateLimit, 'repositories', githubToken),
        githubService.getTotalCount(term, dateLimit, 'pull_requests', githubToken),
        githubService.getTotalCount(term, dateLimit, 'discussions', githubToken)
    ]);

    return {
        searchTerm: term,
        dateLimit,
        starHIndex: starResult.hIndex,
        forkHIndex: forkResult.hIndex,
        totalRepos: totalReposCount,
        analyzedRepos: allRepos.length,
        totalPRs: totalPRsCount,
        totalDiscussions: totalDiscussionsCount,
        topStarredRepos,
        topForkedRepos,
        githubToken: !!githubToken // Include token availability flag
    };
};    // Handle form submission
const handleSearch = async (terms = searchTerms) => {
    if (terms.length === 0) {
        setError('Please add at least one search term');
        return;
    }
```

```
// Check if authentication is required but not provided
if ((showTrendAnalysis || showHIndexAnalysis) && !githubToken) {
    setError('GitHub authentication is required for the selected analysis options');
    return;
}

setSearchTerms(terms);
setIsLoading(true);
setError(null);

// Start progress bar
if (window.NProgress) window.NProgress.start();

// Update URL with search parameters
urlSharingUtils.updateSearchUrl(
    searchTerms.join(';'),
    dateLimit,
    startYear,
    endYear,
    granularity,
    metric,
    showHIndexAnalysis,
    showTrendAnalysis
);

// Log search event for analytics
analyticsService.logSearch({
    searchTerms: terms,
    dateLimit,
    startYear,
    endYear,
    granularity,
    metric,
    showHIndexAnalysis,
    showTrendAnalysis
});

try {
    // Perform H-Index calculation if enabled
    if (showHIndexAnalysis) {
```

```
    if (terms.length > 1) {
        // Handle multiple search terms
        const results = [];

        for (const term of terms) {
            try {
                const result = await processSearchTermForHIndex(term);
                results.push(result);
            } catch (err) {
                console.error(`Error processing term "${term}":`, err);
                // Continue with other terms even if one fails
            }                   }

        setMultiHIndexResults(results);
        setHIndexResults(null); // Clear single result when showing multiple
    } else {
        // Handle single search term
        const result = await processSearchTermForHIndex(terms[0]);
        setHIndexResults(result);
        setMultiHIndexResults([]); // Clear multiple results when showing single
    }
}

// Perform trend analysis if enabled
if (showTrendAnalysis && githubToken) {
    const timeWindows = trendTrackerService.generateTimeWindows(
        startYear,
        endYear,
        granularity
    );

    // Fetch regular metric data
    const data = await trendTrackerService.compareSearchTerms(
        terms,
        timeWindows,
        metric,
        githubToken
    );

    setChartData(data);
}
```

```
        } catch (err) {
            // Log error to analytics
            analyticsService.logError('search_error', err.message);
            setError(err.message);
        } finally {
            setIsLoading(false);
            if (window.NProgress) window.NProgress.done();
        }
    };

    return (
        <div className="max-w-4xl mx-auto">
            <h1 className="text-3xl font-bold text-center mb-4">
                GitHub Trends Analysis
            </h1>

            {/* Token Input */}
              <div className="bg-white p-6 rounded-lg shadow-md mb-6">
                <TokenInput githubToken={githubToken} setGithubToken={setGithubToken} />
            </div>

            {/* Token warning for trend analysis */}
            {showTrendAnalysis && !githubToken && (
                <div className="bg-yellow-100 border-l-4 border-yellow-500 text-yellow-700
p-4 mb-4">
                    <p>GitHub Authentication is required for trend analysis. Please sign in
with GitHub above.</p>
                </div>
            )}

            {/* Unified Search Interface */}
            <fieldset disabled={isLoading}>
            <div className="bg-white p-6 rounded-lg shadow-md mb-6">

                {/* Search terms input - using the new TagInput component */}
                <h3 className="font-bold mb-2">Search Terms</h3>
                <div className="mb-4">
                    <TagInput
                        tags={searchTerms}
                        setTags={setSearchTerms}
                        onAnalyze={handleSearch}
```

```
                    isLoading={isLoading}
                />
            </div>
            {/* Analysis Options */}
            <div className="mb-4">
                <h3 className="font-bold mb-2">Trends Analysis</h3>
                <div className="flex flex-wrap gap-4">
                    <div className="flex items-center">
                        <input
                            type="checkbox"
                            id="showTrendAnalysis"
                            className="mr-2"
                            checked={showTrendAnalysis}
                            onChange={(e) => setShowTrendAnalysis(e.target.checked)}
                        />
                        <label htmlFor="showTrendAnalysis">Show Trends</label>
                    </div>
                </div>
            </div>

            {/* Conditional Trend Settings */}
            {showTrendAnalysis && (
                <div className="bg-gray-50 p-4 rounded mb-4">
                    <div className="grid grid-cols-1 md:grid-cols-3 gap-4 mb-4">
                        {/* Start Year */}
                        <div>
                            <label className="block text-gray-700 mb-2">Start
Year</label>
                            <select
                                className="w-full p-2 border rounded"
                                value={startYear}
                                onChange={(e) => setStartYear(e.target.value)}
                            >
                                {Array.from({ length: 15 }, (_, i) => new
Date().getFullYear() - 14 + i).map(year => (
                                    <option key={`start-${year}`}
value={year}>{year}</option>
                                ))}
                            </select>
                        </div>
```

```
                            {/* End Year */}
                            <div>
                                <label className="block text-gray-700 mb-2">End Year</label>
                                <select
                                    className="w-full p-2 border rounded"
                                    value={endYear}
                                    onChange={(e) => setEndYear(e.target.value)}
                                >
                                    {Array.from({ length: 15 }, (_, i) => new
Date().getFullYear() - 14 + i).map(year => (
                                        <option key={`end-${year}`}
value={year}>{year}</option>
                                    ))}
                                </select>
                            </div>

                            {/* Granularity */}
                            <div>
                                <label className="block text-gray-700 mb-2">Time
Granularity</label>
                                <select
                                    className="w-full p-2 border rounded"
                                    value={granularity}
                                    onChange={(e) => setGranularity(e.target.value)}
                                >
                                    <option value="year">Yearly</option>
                                    <option value="quarter">Quarterly</option>
                                    <option value="month">Monthly</option>
                                </select>
                            </div>
                        </div>

                        {/* Metric selection */}
                        <div className="mb-4">
                            <label className="block text-gray-700 mb-2">Metric</label>
                            <select
                                className="w-full p-2 border rounded"
                                value={metric}
                                onChange={(e) => setMetric(e.target.value)}
                            >
                                <option value="repositories">Repository Count</option>
```

```
                <option value="prs">Pull Request Count</option>
                <option value="issues">Issue Count</option>
            </select>
        </div>
    </div>
)}

    {/* H-Index Options */}
    <div className="mb-4">
    <h3 className="font-bold mb-2">H-Index Analysis</h3>
    <div className="flex flex-wrap gap-4">
        <div className="flex items-center">
            <input
                type="checkbox"
                id="showHIndexAnalysis"
                className="mr-2"
                checked={showHIndexAnalysis}
                onChange={(e) => setShowHIndexAnalysis(e.target.checked)}
            />
            <label htmlFor="showHIndexAnalysis">Calculate H-Index</label>
        </div>
    </div>
</div>

{/* Basic date limit filter (common for both analyses) */}
{showHIndexAnalysis && (
    <div className="bg-gray-50 p-4 rounded mb-4">
    <div className="mb-4">
    <label className="block text-gray-700 mb-2" htmlFor="dateLimit">
        Repositories Created After
    </label>
    <input
        id="dateLimit"
        type="date"
        className="w-full p-2 border rounded"
        value={dateLimit}
        onChange={(e) => setDateLimit(e.target.value)}
    />
</div>
</div>
)}
```

```
</div>
</fieldset>
{/* Error display */}
<ErrorDisplay error={error} />

{/* Results Section */}
{/* Trend Chart if trend analysis is enabled */}
{showTrendAnalysis && chartData && (
    <div className="bg-white p-6 rounded-lg shadow-md mb-6">
        <h2 className="text-2xl font-bold mb-4">Trend Analysis</h2>

        {/* Use the TrendChart component */}
        <TrendChart
            chartData={chartData}
            metric={metric}
        />

        {/* Trend comparison table */}
        <TrendComparisonView
            chartData={chartData}
            metric={metric}
            useWindowedHIndex={false}
        />
    </div>
)}

{/* H-Index results if H-Index analysis is enabled */}
{showHIndexAnalysis && (
    <>
        {multiHIndexResults.length > 0 ? (
            <MultiSearchResults multiResults={multiHIndexResults} />
        ) : (
            <ResultsDisplay results={hIndexResults} />
        )}
    </>
)}

{/* Share Button for both functionalities */}
{(showHIndexAnalysis && (hIndexResults || multiHIndexResults.length > 0)) ||
 (showTrendAnalysis && chartData) ? (
    <div className="flex justify-center my-6">
```

```
                    <ShareButton
                        settings={{
                            searchTerm: searchTerms.join(';'),
                            dateLimit,
                            startYear,
                            endYear,
                            granularity,
                            metric,
                            showHIndexAnalysis,
                            showTrendAnalysis
                        }}
                    />
                </div>
            ) : null}

            <Footer />
        </div>
    );
}

export default GitHubHIndexApp;
```