Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження методів та інструментів для групування помилок журналу тестування за допомогою алгоритмів подібності рядків
(тема)

Виконав:
студент 2 курсу, групи ІПЗм-22-3

Гриб Р.В.
(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)
Тип програми освітньо-наукова

Керівник доц. Ревенчук І.А.
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри _____ З.В.Дудар
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерних наук
Кафедра програмної інженерії
Рівень вищої освіти другий (магістерський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми освітньо-наукова програма
Освітня програма Інженерія програмного забезпечення
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Грибу Роману Владиславовичу
(прізвище, ім’я, по батькові)

1. Тема роботи «Дослідження методів та інструментів для групування помилок журналу тестування за допомогою алгоритмів подібності рядків»
Затверджена наказом по університету від 29.03.2024р. № 250Ст
2. Термін подання студентом роботи до екзаменаційної комісії 20.06.2024
3. Вихідні дані до роботи опис досліджуваних алгоритмів подібності рядків, опис метрик для оцінки і порівняння алгоритмів, мова програмування JavaScript, React, технології NPM, середовище розробки Visual Studio Code
4. Перелік питань, що потрібно опрацювати в роботі
аналіз та порівняння існуючих алгоритмів подібності рядків, вибір відповідних NPM технологій, написання програмних рішень, проведення експериментів та аналіз отриманих результатів

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Отримання завдання	01.04.2024	*виконано*
2	Аналіз предметної галузі та постановка задачі	01.04 – 10.04.24	*виконано*
3	Аналіз та вибір алгоритмів для дослідження	10.04 – 14.04.24	*виконано*
4	Аналіз та моделювання предметної області	14.04 – 20.04.24	*виконано*
5	Планування експериментів	20.04 – 29.04.24	*виконано*
6	Програмна реалізація кожного з обраних для дослідження API	29.04 – 10.05.24	*виконано*
7	Експериментальні дослідження	10.05 – 14.05.24	*виконано*
8	Аналіз результатів експериментальних досліджень	14.05 – 16.05.24	*виконано*
10	Підготовка пояснювальної записки	16.05 – 25.05.24	*виконано*
11	Підготовка презентації та доповіді	25.05 – 30.05.24	*виконано*
12	Перевірка на плагіат	30.05 – 03.06.24	*виконано*
13	Нормоконтроль	03.06 – 08.06.24	*виконано*
14	Рецензування	08.06 – 12.06.24	*виконано*
15	Попередній захист	12.06.2024	*виконано*
16	Занесення диплома в електронний архів	15.06.2023	*виконано*
17	Допуск до захисту у зав. кафедри	19.06.2024	*виконано*

Дата видачі завдання 01 квітня 2024р.

Студент (ка) _____ Гриб Р.В.
(підпис)

Керівник роботи _____ доц. Ревенчук І.А.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка містить: 61 с., 35 рис., 5 табл., 9 джерел.

ТЕСТУВАННЯ, ГРУПУВАННЯ ПОМИЛОК, АЛГОРИТМИ, ПОДІБНІСТЬ РЯДКІВ, NPM ПАКЕТИ.

Об’єктом дослідження є методи та інструменти групування помилок.

Метою роботи є проведення дослідження продуктивності методів групування помилок за допомогою алгоритмів подібності рядків.

Методами розробки та проектування є аналіз проблемної області дослідження, вибір оптимального метода групування помилок шляхом вирішення багатокритеріальної задачі прийняття рішень та проведення експериментального аналізу даних.

У результаті кваліфікаційної роботи було реалізовано та застосовано п’ять методів та інструментів групування помилок, по одному для кожного з досліджуваних алгоритмів подібності рядків: алгоритм знаходження косинусної подібності, алгоритм Левенштейна, алгоритм знаходження індексу Жаккара, алгоритм знаходження подібності Джаро-Вінклера, алгоритм знаходження коефіцієнта Соренсена-Дайса.

TESTING, ERROR GROUPING, ALGORITHMS, STRING SIMILARITY, NPM PACKAGES.

The object of research is methods and tools for error grouping.

The purpose of the work is to investigate the performance of error grouping methods using string similarity algorithms.

The development and design methods are the analysis the analysis of the problem area of research, the selection of the optimal method of error grouping by solving a multi-criteria decision-making problem, and experimental data analysis.

As a result of the qualification work, five methods and tools for grouping errors

were implemented and applied, one for each of the studied string similarity algorithms: the cosine similarity algorithm, the Levenshtein algorithm, the Jaccard index algorithm, the Jaro-Winkler similarity algorithm, and the Sorensen-Dice coefficient algorithm.

Заява щодо самостійного виконання кваліфікаційної роботи та можливості її публікації в електронному архіві відкритого доступу ElArKhNURE.

Я, Гриб Роман Владиславович, студент гр. ІПЗм-22-3, здобувач вищої освіти на другому (магістерському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Дослідження методів та інструментів для групування помилок журналу тестування за допомогою алгоритмів подібності рядків», що буде представлена в екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElArKhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений(на) з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

ВСТУП

На сьогоднішній день, стрімко розвивається індустрія розробки програмного забезпечення (ПЗ). Динаміка зростання ПЗ як продукту на ринку сприяє постійній еволюції методів розробки та появі технологічних нововведень. Використання штучного інтелекту і машинного навчання у ПЗ стало звичайною практикою, розгортання продукту на різноманітних платформах стало необхідністю у повсякденні, а швидкість та якість розробки стали основними факторами для підвищення конкурентоспроможності на ринку.

Підвищення рівня складності програмних систем часто призводить до збільшення кількості помилок під час розробки, тому ця проблема актуальна як зараз, так і буде актуальною у майбутньому.

Для забезпечення високої якості і надійності продукту існує безліч підходів до організації життєвого циклу розробки, де на кожному етапі проводиться аналіз поставленої задачі та оцінка необхідних ресурсних витрат на її виконання. Але ключовим та незамінним етапом розробки є тестування програмного забезпечення.

Тестування ПЗ дозволяє забезпечити якість та цілісність кінцевого продукту, допомагає виявити помилки та недоліки в програмному забезпеченні, а також сприяє економії коштів і часу, адже виявлення помилок на ранніх етапах розробки коштує набагато менше ніж їх виправлення після відправки продукту на ринок.

Для ефективної класифікації та управління помилками у тестуванні використовують групування помилок за їх подібністю. Даний метод допомагає розробникам і тестувальникам виявляти закономірності виникнення помилок та вдосконалювати стратегії їх тестування, позитивно впливаючи на швидкість аналізу та вирішення проблем.

Серед найефективніших методів групування помилок – є групування за допомогою алгоритмів подібності рядків, основна ідея яких полягає у виявленні

схожих або ідентичних рядків у журналі тестування або у вихідних кодах, та об’єднання їх у групи для подальшого аналізу і зосередження уваги на виправленні загальних проблем.

Дана робота спрямована на аналіз існуючих методів групування помилок, які використовують найпопулярніші алгоритми подібності рядків.

Метою дослідження – є знаходження найкращого, з боку оптимальності використання алгоритму, який має низький обсяг використаних ресурсів і високу точність при великій кількості помилок.

Дослідження методів та інструментів групування помилок у тестуванні програмного забезпечення це важливий крок у напрямку покращення якості і надійності продукту. Результат даного дослідження має значний потенціал для вдосконалення процесів розробки та забезпечення успішного впровадження ПЗ на ринку.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

У даному розділі буде розглянуто основні цілі тестування, цінність тестування у розробці програмних застосунків, основні види тестування, відомості про журнал помилок, а також алгоритми подібності рядків, на основі чого буде поставлена задача даного дослідження.

1.1 Основні відомості про тестування

Тестування – це етап в процесі розробки програмного забезпечення (ПЗ), який є важливою частиною даного процесу і передбачає виконання ряду дій з метою визначення якості та правильності роботи програмного коду. Основна мета тестування – забезпечити високу якість програми, допомогти виявити та виправити помилки до випуску проекту в публічне чи приватне використання клієнтам.

Тестування може проводитися як власноруч, так і за допомогою автоматизованих інструментів, фреймворків, середовищ для тестування. Ефективне тестування дозволяє виявляти та усувати помилки на ранніх етапах розробки, зменшуючи витрати на їх виправлення та забезпечуючи високу якість готового продукту.

Розвиток галузі розробки програмного забезпечення тісно пов’язаний із розвитком галузі тестування, адже з самого першого етапу розробки ПЗ, необхідно було перевірити їх працездатність та результат.

З плином часу, складність програм, а також їх обсяг зростали, тому з'явилася необхідність у спеціалізованих фахівцях, які займалися виключно тестуванням розробленого ПЗ. Програмісти використовували ручне тестування та писали власні тести для перевірки правильності роботи використовуваного коду.

Поява більш прогресивних технологій та розвиток комп'ютерної галузі вплинули на появу автоматизованих інструментів тестування, у результаті чого збільшилася швидкість та ефективність тестування. З'явилися нові інструменти для автоматизації, що дозволило використовувати спеціальні команди – скрипти, які запускали автоматичне виконання тестів.

У сучасних методологіях розробки, таких як Agile та DevOps, тестування стало невід'ємною частиною циклу розробки. Тестувальники активно взаємодіють із розробниками та іншими членами команди напряму, забезпечуючи регулярне та автоматизоване тестування під час всього процесу розробки [1-3].

Технології продовжують розвиватися, що призводить до збільшення загроз у сфері кібербезпеки, а тестування є важливою складовою процесу розробки захисту від кіберзагроз. Програмне забезпечення, мережі та інші цифрові ресурси перевіряються на вразливості, які потенційно можуть бути використані зловмисниками для несанкціонованого доступу, отримання конфіденційної інформації, тощо. І саме якісне тестування допоможе виявити недоліки та прогалини у системі, результат виправлення яких зміцнює безпеку програмного забезпечення.

Також розвиваються інші спеціалізовані напрямки тестування, такі як тестування продуктивності, доступності, сумісності, тестування мобільних додатків та інші.

1.2 Види тестування

Окрім напрямків, при розробці програмного забезпечення, набули розвитку підходи до тестування. Від перевірки коду на правильність розрахунків до повної реплікації дій користувача при використанні системи, що забезпечує надійний захист програми від помилок при розробці нового функціоналу. Розглянемо основні види тестування.

Модульне тестування (Unit Testing) – вид тестування, при якому перевіряються окремі функції або методи програми для визначення їх правильності роботи. Це основний підхід, який закладає фундамент при загальному тестуванні застосунку. Зазвичай, при модульному тестуванні перевіряються нетривіальні функції, для того, щоб у разі зміни кодової бази запобігти негативному впливу на повернений результат із функції або методу (див. рис. 1.1).

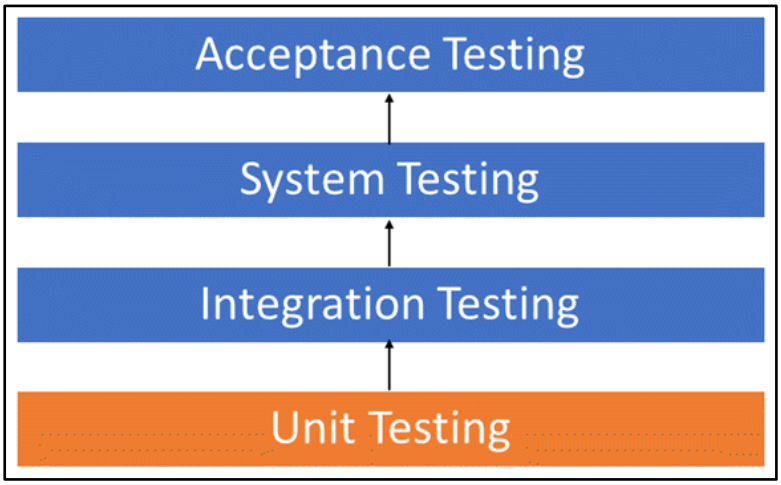


Рисунок 1.1 – Модульне тестування (за даними [3])

Інтеграційне тестування (Integration Testing) – вид тестування, який спрямований на перевірку взаємодії декількох компонентів програми які є складовими певного блоку або частини системи. Основна мета даного виду тестування – виявити та вказати на помилки взаємодії між компонентами програми. Конфлікти між програмними модулями трапляються з багатьох причин, таких як несумісність версій підсистем, конфлікт форматів даних або логіки обробки. Інтеграційне тестування виявляє ці проблеми зв'язку між програмними компонентами. Зазвичай воно відбувається після модульного тестування і перед системним тестуванням (див. рис. 1.2).

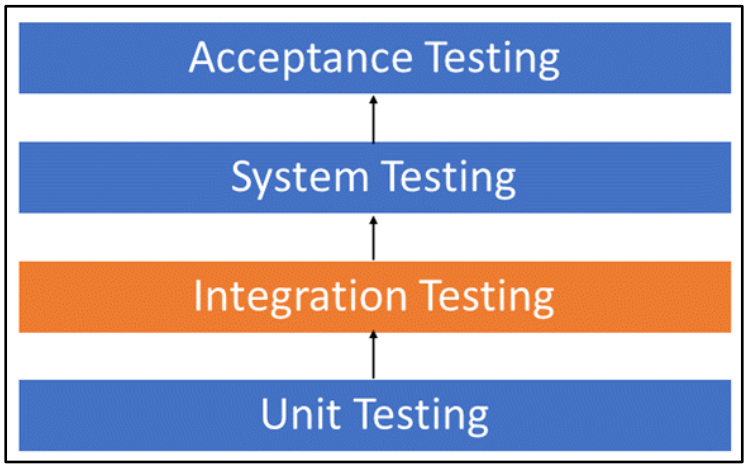


Рисунок 1.2 – Інтеграційне тестування (за даними [3])

Програмне забезпечення складається з багатьох окремих програмних компонентів або модулів. Ці модулі можуть успішно проходити модульне

тестування і чудово працювати окремо, але з різних причин викликати помилки, коли їх використовують разом.

Найбільш поширена помилка – неузгодженість логіки коду. Виникає у тому випадку, якщо два (або більше) різних програмісти взаємодіють із взаємопов’язаними модулями системи, змінюючи логіку поведінки модуля та застосовуючи свій підхід до розробки, тому при інтеграції модулі викликають функціональні проблеми.

Клієнти часто змінюють свої вимоги. Модифікація коду одного модуля для адаптації до нових вимог іноді призводить до повної зміни логіки роботи коду, що впливає на весь додаток в цілому. Ці зміни не завжди відображаються в модульному тестуванні, тому виникає потреба в інтеграційному тестуванні для виявлення відсутніх дефектів.

Дані можуть змінюватися під час передачі між модулями. Якщо вони не мають належного формату під час перенесення, дані не зможуть бути прочитані та оброблені, що також призводить до помилок. Оскільки дані можуть змінюватися у процесі передачі, API (Application Programming Interface) та сторонні сервіси, які використовуються у системі, можуть отримувати помилкові дані і генерувати неправильні відповіді. Іноді це навіть призводить до повної заморозки системи.

Помилки також можуть виникати через несумісність між програмним та апаратним забезпеченням, що можна легко виявити за допомогою належного інтеграційного тестування.

Системне тестування (System Testing) – це вид тестування, при якому перевіряється, чи відповідає система встановленим вимогам та специфікаціям. Зазвичай, програмне забезпечення є лише однією частиною великої комп’ютерної системи та взаємодіє з іншими програмно-апаратними системами. Системне тестування включає в себе серію різних тестів, єдиною метою яких є перевірка всієї комп’ютерної системи. Це перед-останній етап тестування програмної системи, який допомагає переконатися в можливості та готовності системи до випуску та використання в реальних умовах (див. рис. 1.3).

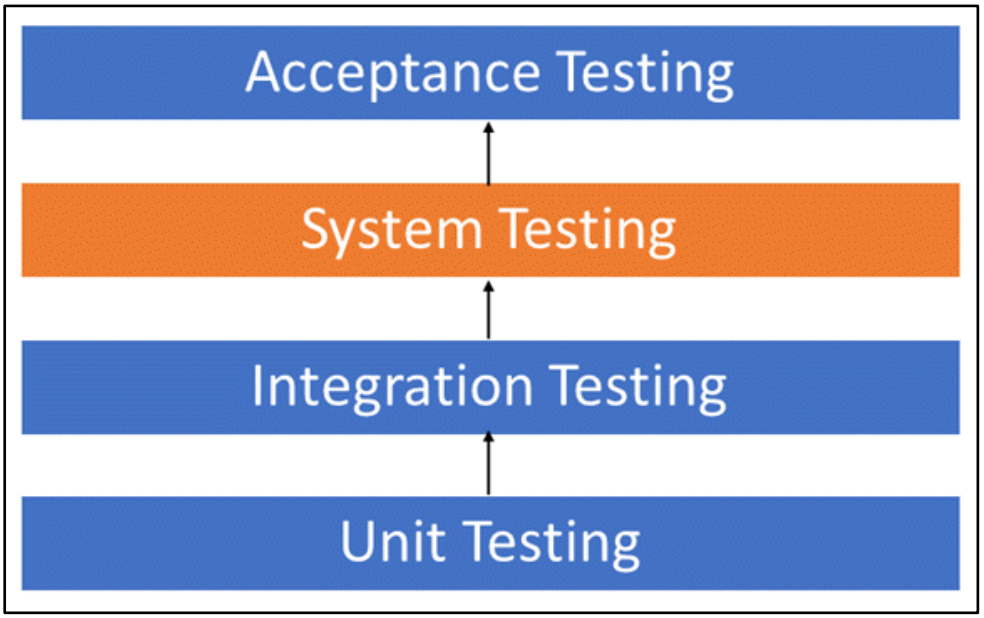


Рисунок 1.3 – Системне тестування (за даними [3])

Приймальне тестування (Acceptance Testing) – це вид тестування, при якому систему оцінюють на відповідність бізнес-вимогам та готовність до випуску. Даний вид тестування допомагає проекту проаналізувати подальші вимоги користувачів безпосередньо, адже під час приймального тестування залучаються реальні користувачі. Це останній етап тестування ПЗ, який виконується після системного тестування та перед релізом системи для використання (див. рис. 1.4).

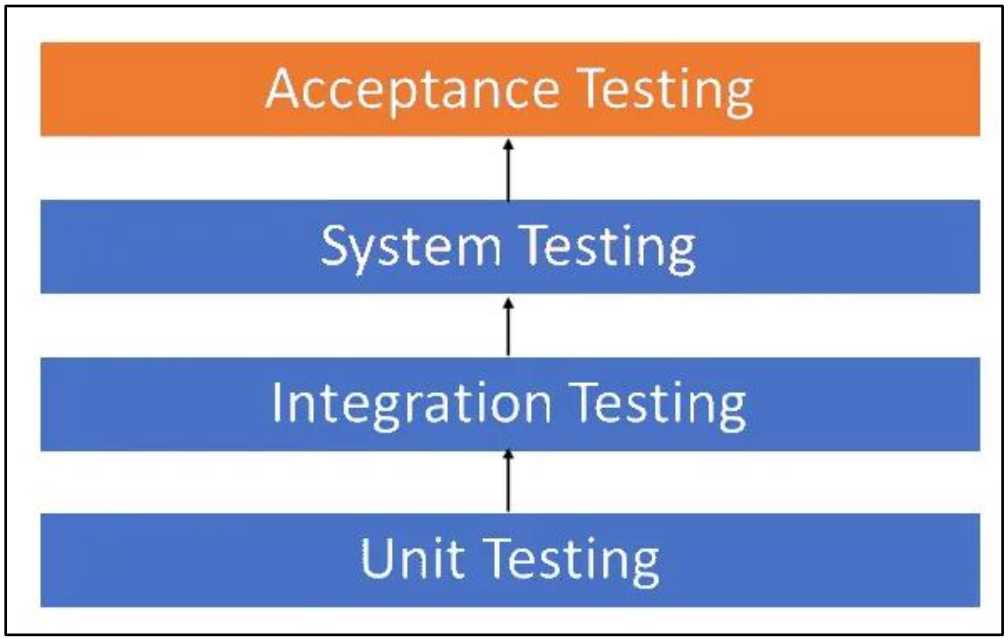


Рисунок 1.4 – Приймальне тестування (за даними [3])

Приймальне тестування забезпечує впевненість і задоволення клієнтам, адже вони мають змогу перевірити роботу системи, вказати на недоліки або переваги, навіть описати свої вимоги до майже готового продукту.

1.3 Журнал помилок

Під час тестування програмного забезпечення розробник може зіткнутися із програмними або апаратними помилками. Наприклад, під час зміни логіки в одному з модулів системи згідно з новими вимогами користувача «А», він може не помітити, як ця зміна логіки вплине на попередні вимоги користувача «Б». У результаті програма може працювати з помилкою для користувача «Б», тоді як користувач «А» буде задоволений, оскільки його вимоги були виконані.

Тести запобігають таким ситуаціям та повідомляють розробника, що один з протестованих раніше модулів змінився і працює некоректно. Це повідомлення відображається у спеціальному вікні у вигляді помилок – в спеціальному вікні у середовищі розробки програмного забезпечення. Загальновживано, це вікно називають «журналом помилок» (Error Log).

Журнал помилок тестування є відправною точкою, з якої можливе виявлення першопричини проблеми. Він допомагає виявити проблеми, пов’язані з продуктивністю, пам’яттю, низькою пропускною здатністю тощо. У даному випадку розробник бачить помилки, пов’язані з вимогами користувача «Б» у журналі помилок, і за допомогою методів групування помилок збирає та аналізує їх, що в результаті допомагає при вирішенні даної проблеми.

На протязі усієї історії тестування програмного забезпечення програмісти шукали ефективні методи та інструменти для забезпечення якості програм та виявлення помилок під час розробки проектів. Це сприяло появі великої кількості методів і алгоритмів для вирішення даної проблеми, а також розв’язанню другорядних проблем під час процесу тестування. Одним із них є групування помилок за певним критерієм. У даному випадку це – групування помилок за знаходженням подібності між рядками тестових одиниць.

1.4 Основні алгоритми подібності рядків

Для того, щоб врахувати схожість між різними записами у журналі тестування, а також для ідентифікації закономірностей чи типових проблем, існує безліч алгоритмів подібності рядків. Визначимо основні та найпоширеніші алгоритми, які використовуються для групування помилок.

Алгоритм подібності рядків на основі косинусної подібності (cosine similarity) використовується для оцінки ступеня подібності між двома векторами у просторі. В контексті журналу тестування, записи помилок можуть бути представлені як вектори, де кожен компонент вектору відповідає певному терміну або слову, а значення компоненту представляє частоту вживання цього терміну у даному записі.

Основні кроки алгоритму:

- представлення даних у векторній формі: спочатку кожен запис у журналі тестування представляється як вектор;
- визначення векторів: створюються вектори для кожного запису, де вектор відповідає терміну, а значення визначається частотою вживання терміну у записі;
- обчислення косинусної подібності: знаходиться косинусна подібність між двома векторами, обчислюючи косинус кута між ними;
- визначення ступеня подібності: аналізуємо ступінь подібності, і чим більше значення, тим більше подібність;
- групування: в залежності від вимог та потреб, отримані значення використовуються для групування записів за ступенем подібності.

Основну формулу для обчислення косинусної подібності між двома векторами A і B, можна описати за формулою 1.1:

$$Cosine\ Similarity = \frac{A * B}{\|A\| * \|B\|} \tag{1.1}$$

де $A*B$ – скалярний добуток векторів;

$\|A\| * \|B\|$ – їхні довжини.

Слід зазначити, що під час реалізації алгоритму важливо вирішити питання нормалізації векторів та врахувати ваги важливості термінів, для досягнення точних результатів при обчисленні косинусної подібності між векторами.

Алгоритм подібності рядків за допомогою редакційної відстані (Алгоритм Левенштейна) використовується для вимірювання схожості між двома рядками, обчислюючи мінімальну кількість операцій редагування, необхідних для того, щоб перетворити один рядок у інший.

Основні операції редагування, які враховуються алгоритмом Левенштейна, включають вставку, видалення та заміну символу. Відстань редагування – це кількість таких операцій, які потрібно виконати, щоб один рядок перетворився в інший, досягаючи знаходження подібності.

Основні кроки алгоритму:

- ініціалізація матриці: створюється матриця розміром (m+1)x(n+1), де m та n – довжини рядків (помилок), які порівнюються. Також ініціалізуються перший рядок і перший стовпець значеннями від 0 до m та від 0 до n;
- обчислення відстані Левенштейна: здійснюється проходження матриці та заповнення кожної клітинки значенням, яке представляє мінімальну кількість операцій редагування для досягнення відповідного підрядка;
- оптимізація простору: оптимізується використання пам’яті, адже для обчислення поточного рядка нам потрібно лише попередній рядок, необхідно зберігати лише два рядки матриці;
- отримання відстані: остання клітинка матриці і буде відстанню Левенштейна між двома рядками;
- відновлення шляху редагування (не обов’язковий крок): за необхідності, можна відновити шлях отримання операцій редагування для більш детального опису редагування рядків.

Для обчислення значень в матриці використаємо формулу 1.2:

$$d[i,j] = \min\{d[i-1,j]+1, d[i,j-1]+1, d[i-1,j-1]+cost(s[i],t[j])\} \quad (1.2)$$

де *d[i, j]* – значення в клітинці (i, j) матриці;

s[i] та *t[j]* – символи на відповідних позиціях у порівнюваних рядках;

cost(a,b) – функція визначення вартості операції редагування між символами a та b.

Наприклад маємо матрицю знаходження подібності між двома словами «Sunday» та «Saturday», і 3 – це відстань Левенштейна (див. рис. 1.5).

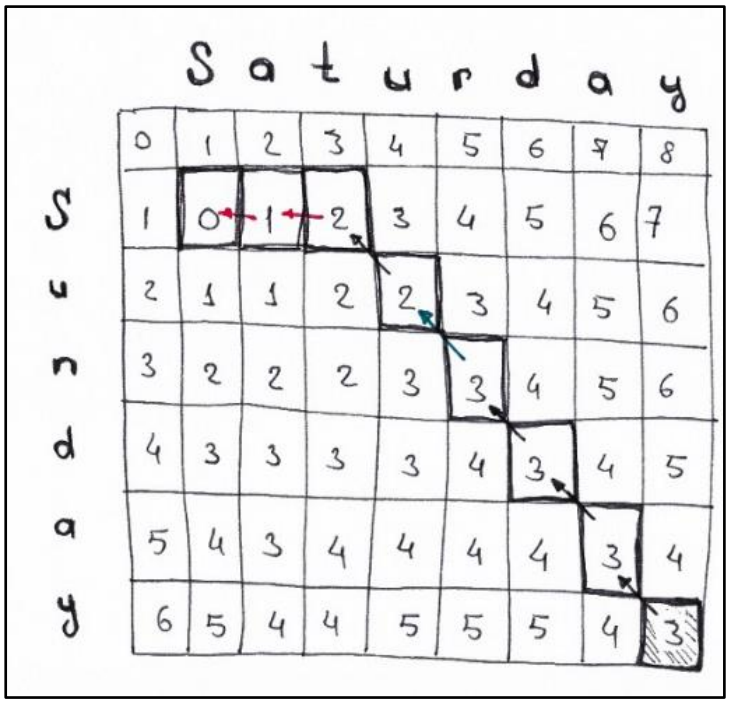


Рисунок 1.5 – Відстань Левенштейна (за даними [4])

Алгоритм Левенштейна становить ефективний інструмент для знаходження подібності у журналі помилок та визначення мінімальних операцій для їхнього перетворення один в один.

Використання нейронних мереж для розв’язку задачі групування та знаходження подібності між помилками під час тестування може збільшити точність знаходження подібності, при цьому, зменшуючи швидкість обчислювання подібності. При використанні цього методу поєднуються методи машинного навчання та традиційного аналізу тексту.

Основні кроки алгоритму:

- підготовка даних: спочатку збираються дані з журналу тестування, де кожен запис містить інформацію про помилку. Проводиться попередня обробка даних для покращення якості вхідних даних;
- векторне представлення тексту: отримані дані перетворюємо у векторне представлення, яке може бути використане нейронною мережею. Можна використовувати техніки, такі як TF-IDF (Term Frequency Inverse Document Frequency) або векторизацію словників;
- аналіз подібності рядків: використовуємо алгоритми подібності рядків, такі як косинусна подібність або алгоритм Левенштейна, для визначення ступеня подібності між рядками;
- навчання нейронної мережі: проводимо навчання мережі на позначених даних, де помилки вже мають призначені категорії чи мітки груп;
- об'єднання результатів: комбінуємо результати від нейронної мережі та алгоритмів подібності рядків для отримання комплексного підходу до групування помилок;
- оцінка та налаштування: оцінюємо ефективність системи групування за допомогою метрик точності;
- виправлення та перевірка: перевіряємо результати групування та проводимо виправлення невірно класифікованих помилок. Це може включати додавання нових категорій або корекцію існуючих.

Для групування помилок у журналі тестування також використовують відомий метод для вимірювання схожості між рядками – індекс Жаккара. Це міра подібності, яка була запропонована професором ботаніки Полем Жаккаром у 1901 році і використовується у багатьох галузях для вимірювання схожості між множинами, зокрема у програмуванні.

Розглянемо основні кроки алгоритму в області нашого дослідження:

- розбиття рядків на підмножини (токени): розбиваємо рядок на слова, де кожне слово стає елементом множини;
- обчислення множини токенів для кожного рядка: створюємо множини токенів для кожного рядка;

– обчислення перетину та об’єднання множин: для двох рядків A і B з множинами токенів S(A) і S(B) знаходимо перетин (∩ – множина токенів, які містяться в обох рядках) і об’єднання (∪ – множина токенів, що містить всі елементи, які є в множині A або B, або в обох);
– визначення індексу Жаккара: обчислюємо індекс як відношення розміру перетину до розміру об’єднання.

У результаті отримуємо певне значення індексу Жаккара для кожної пари помилок. Чим вище значення індексу, тим більш схожими будуть вважатися помилки, і їх можна буде згрупувати разом. Алгоритм розрахунку індексу Жаккара є досить простим і ефективним методом для вимірювання схожості між рядками, який широко використовується у процесах тестування та розробки програмного забезпечення.

Існує ще безліч подібних алгоритмів, але вони не сильно відрізняються за витратами та точністю результатів. Для того, щоб обрати найкращий алгоритм подібності рядків для групування помилок у журналі тестування, треба вирішити багатокритеріальну задачу, у результаті якої знайдемо алгоритм, який буде найбільш вигіднім та ефективним для використання.

1.5 Постановка задачі

Основною задачею є аналіз результату групування помилок журналу тестування за допомогою вирішення багатокритеріальної задачі. Тому багатокритеріальною задачею буде – вибрати найкращий метод для групування помилок, враховуючи подані критерії.

Під час дослідження необхідно не тільки враховувати індивідуальні властивості кожного алгоритму, але й їх взаємодію за допомогою багатокритеріальної оптимізації.

Існує безліч алгоритмів подібності рядків, зробимо вибір з декількох найпопулярніших та порівняємо їх відносно таких критерій:

– точність результату: визначає, наскільки точно обраний метод надає точні результати групування. Важливо мати високу точність для

надійного аналізу і виправлення помилок;

- швидкість обробки: оцінює, наскільки швидко метод оброблює наданий обсяг даних. Особливо це важливо під час обробки великого обсягу даних;
- обсяг використаних ресурсів: оцінка обсягу використаних ресурсів є важливою для оцінки ефективності системи. Особливо у великих проектах, де ресурси можуть бути обмеженими;
- стійкість до шуму в даних: оцінює наскільки ефективно обраний метод здатний виявляти та ігнорувати шумові дані у журналі тестування, це необхідно для точного групування;
- вартість: оцінює загальні витрати, враховує фінансові обмеження та обирає метод, який ефективно реалізується в межах бюджету.

Для дослідження було обрано декілька популярних алгоритмів подібності рядків.

Алгоритм подібності рядків на основі косинусної схожості (CosSimilarity): цей алгоритм особливо популярний у області обробки природної мови (NLP – Natural Language Processing) та інших задачах аналізу тексту, має зазвичай високу точність, оскільки враховує семантичну схожість між текстовими рядками. Швидкість обробки – досить висока для великої кількості коротких рядків, але обчислення косинусної схожості може стати витратним завданням для довгих рядків. Зазвичай вимагає мало ресурсів для обробки коротких текстових рядків, але вимоги можуть зростати зі збільшенням обсягу даних. Відносно стійкий до шуму, оскільки використовує векторне представлення, яке може враховувати контексти слів. Вартість впровадження – зазвичай невисока, так як метод має не велику складність щодо реалізації, тому буде залежати від команди розробників, обраного обладнання і тривалості розробки.

Алгоритм подібності рядків за допомогою редакційної відстані (Levenshtein): відомий як алгоритм Левенштейна, використовує поняття редакційної (редагувальної) відстані для вимірювання схожості між двома рядками. Редакційна відстань визначає мінімальну кількість редагувальних

операцій (вставка, вилучення, заміна) для перетворення рядків. Може виявляти помилки з невеликими редакційними змінами, тому має високу точність, високу стійкість до шуму при невеликій кількості даних, а також має потенціал для швидкої обробки. Вартість впровадження відносно невисока, оскільки редакційна відстань вже є добре вивченою та реалізованою концепцією.

Алгоритм розрахунку індексу Жаккара (JaccardIndex): значна перевага у тому, що обчислення індексу не вимагає реалізації складних алгоритмів або значних обчислювальних ресурсів, добре працює з рядками, в яких важливі спільні елементи, але втрачає у точності, якщо даний алгоритм застосовують для множин із великою кількістю унікальних рядків. І хоча індекс Жаккара є досить ефективним, для великих множин даних час обробки та обсяг використаних ресурсів зростає у геометричній прогресії.

Подібність Джаро-Вінклера (JaroWinkler): обчислення коефіцієнта подібності має високу точність, середню швидкість обробки, але досить складний у реалізації порівняно з іншими методами. Добре підходить для обробки текстів, де важливий порядок символів, особливо для текстів, де є значні відмінності на початку рядка.

Коефіцієнт Соренсена-Дайса (SorensenDice): має високу точність обробки текстів, які містять повторювані підрядки, стійкий до шуму, простий у реалізації і не потребує великої кількості ресурсів, підходить для текстів із повторюваними підрядками, де важливий загальний обсяг збігів.

Перед тим як занести дані до таблиці, треба, щоб кожен з критеріїв мав свою власну шкалу та тип. Визначимо основні критерії індивідуально для кожного алгоритму, а саме:

- точність даних – шкала виражена у відсотках, де 0% - це мінімальна точність, 100% - максимальна точність. Значення точності – середньо-арифметичні при використанні середнього обсягу даних;
- швидкість обробки, обсяг використаних ресурсів та стійкість до шуму даних – номінальні шкали, що приймають значення: низька (мінімальне значення), середня, висока, дуже висока (максимальне значення).

Занесемо середні показники вибраних алгоритмів до таблиці (див. табл. 1.1).

Таблиця 1.1 – Критеріальна оцінка алгоритмів (таблиця виконана самостійно)

Назва алгоритму	Точність даних (%)	Швидкість обробки	Обсяг використаних ресурсів	Стійкість до шуму	Вартість
CosSimilarity	77.1	Середня	Низький	Середня	Низька
Levenshtein	81.5	Висока	Середній	Дуже висока	Низька
JaccardIndex	91.6	Низька	Дуже високий	Дуже висока	Висока
JaroWinkler	79.5	Висока	Високий	Низька	Середня
SorensenDice	93.8	Середня	Середній	Дуже висока	Середня

Переробимо таблицю, змінюючи значення якісних шкал під кількісні. Критерії швидкість обробки та стійкість до шуму мають однакову номінальну шкалу, тому будуть мати відповідне однакове оцінювання.

Для цього призначимо бали для цих шкал від 0 до 4, де 4 – максимальний бал:

- дуже висока – 4 бали;
- висока – 3 бали;
- середня – 2 бали;
- низька – 1 бал.

Обсяг використаних ресурсів та вартість – для цих шкал бали будуть інвертовані відносно шкал швидкості обробки та стійкості до шуму, так як чим легше розробити систему – тим менша кількість ресурсів буде задіяна. Тому для цієї шкали будемо мати наступні бали:

- дуже висока – 1 бал;
- висока – 2 бали;

– середня – 3 бали;

– низька – 4 бали.

Занесемо кількісні значення до таблиці (див. табл. 1.2).

Таблиця 1.2 – Кількісні значення алгоритмів

Назва алгоритму	Точність даних (%)	Швидкість обробки	Обсяг використаних ресурсів	Стійкість до шуму	Вартість
CosSimilarity	77.1	2	4	2	4
Levenshtein	81.5	3	3	4	4
JaccardIndex	91.6	1	1	4	2
JaroWinkler	79.5	3	2	1	3
SorensenDice	93.8	2	3	2	3

Проведемо аналіз таблиці відносно множини Парето. Можна сказати, що кожен з алгоритмів має хоча б один критерій, який є кращим відносно деяких з альтернатив.

Звідси можна вважати, що кожен із описаних алгоритмів є конкурентоспроможним.

Тепер можемо зробити нормування оцінок з урахуванням min та max. Для цього в кожному з критеріїв необхідно знайти мінімальне та максимальне значення та після чого виконати обчислення значення критерію для обраної альтернативи за формулою 1.3:

$$f = \frac{f_{\text{розрахункове}} - f_{min}}{f_{max} - f_{min}} \tag{1.3}$$

де f_{max}, f_{min} – максимальне та мінімальні значення;

$f_{\text{розрахункове}}$ – значення обраної альтернативи.

Після чого маємо наступну таблицю (див. табл. 1.3).

Таблиця 1.3 – Обчислення значення критеріїв

Назва алгоритму	Точність даних (%)	Швидкість обробки	Обсяг використаних ресурсів	Стійкість до шуму	Вартість
CosSimilarity	0	0,5	1	0,33	1
Levenshtein	0,26	1	0,66	1	1
JaccardIndex	0,87	0	0	1	0
JaroWinkler	0,14	1	0,33	0	0,5
SorensenDice	1	0,5	0,66	0,33	0,5
max значення	93,8	3	4	4	4
min значення	77,1	1	1	1	2

Визначимо вагові коефіцієнти для критеріїв пропорційним методом відносно вартості. Звідси можемо визначити ваговий коефіцієнт вартості як z. Швидкість обробки буде відбуватися у реальному часі, і вважається що це у 5 разів важливіше за вартість, тому маємо коефіцієнт 5z. На другому місці по важливості буде точність даних тому коефіцієнт буде 4z. Обсяг використаних ресурсів є більш важливою за стійкість до шуму, так як кількість даних може бути великою і треба розробити ефективну систему уникнення непотрібних даних, тому коефіцієнти будуть: 3z для обсягу використаних ресурсів, 2z для стійкості до шуму.

За властивістю суми вагових коефіцієнтів отримуємо, що 15z = 1, тобто маємо наступні вагові коефіцієнти:

- точність даних: 0.266;
- швидкість обробки: 0.333;
- обсяг використаних ресурсів: 0.2;
- стійкість до шуму: 0.133;
- вартість: 0.066.

Застосуємо лінійну адитивну згортку так як маємо оцінки за різними критеріями та кількісні показники. Використаємо формулу 1.4:

$$Z^* = \max_{i=1,m} \sum_{j=1}^{n} a_{ij}^* \quad (1.4)$$

де a_{ij}^* – значення оцінки за показниками;

$\max_{i=1,m}$ – значення максимального показника.

Для подальшого розрахунку лінійної адитивної згортки з ваговими коефіцієнтами застосуємо формулу 1.5:

$$Z^* = \max_{i=1,m} \sum_{j=1}^{n} \alpha_j \beta_j a_{ij} \quad (1.5)$$

де $\alpha_j \beta_j a_{ij}$ – вагові коефіцієнти;

$\max_{i=1,m}$ – значення максимального показника.

Опираючись на формули лінійної адитивної згортки та отримані вагові коефіцієнти заповнимо наступну таблицю (див. табл. 1.4).

Таблиця 1.4 – Розрахунки адитивної згортки

Назва алгоритму	Точність даних (%)	Швидкість обробки	Обсяг використаних ресурсів	Стійкість до шуму	Вартість	Z*
CosSimilarity	0	0,5	1	0,33	1	0,169
Levenshtein	0,26	1	0,66	1	1	0,263
JaccardIndex	0,87	0	0	1	0	0,152
JaroWinkler	0,14	1	0,33	0	0,5	0,163
SorensenDice	1	0,5	0,66	0,33	0,5	0,250
β	0,266	0,333	0,2	0,133	0.066	
α	0,440	0,333	0,377	0,375	0,333	

Виходячи з отриманих результатів, можна сказати, що алгоритм

Левенштейна, отримавши найкращий результат у розмірі 0,263 після проведення згортки є найкращим вибором серед алгоритмів подібності рядків для групування помилок журналу тестування за результатами вирішення багатокритеріальної задачі. Надалі треба підтвердити або спростувати це за допомогою програмної реалізації та експериментальним методом.

2 ОПИС ПРИЙНЯТИХ ПРОЄКТНИХ РІШЕНЬ

2.1 Вибір технологій

Для побудови інтерфейсу для відображення роботи досліджуваних алгоритмів було обрано бібліотеку React на базі JavaScript, як основної мови програмування [5].

React забезпечує компонентний підхід до розробки інтерфейсів, що дозволяє повторно використовувати вже розроблені компоненти для спрощення процесу розробки. Ця бібліотека дозволяє використовувати сучасні можливості JavaScript, що підвищує ефективність і зручність розробки, а завдяки вбудованим механізмам оновлення сторінки під час її використання React посідає перше місце серед переліку технологій, які можна застосувати для даного магістерського дослідження.

2.2 Бібліотеки та інструменти

Для реалізації роботи алгоритмів подібності рядків та розробки застосунку було використано наступні бібліотеки:

- react (версія 18.3.1): основна бібліотека для створення веб-сторінки;
- @mui/material (версія 5.15.17): бібліотека компонентів для створення гарного і функціонального інтерфейсу користувача. Легко інтегрується з React, надає велику кількість можливостей під час розробки компонентів користувацького інтерфейсу;
- performance-now (версія 2.1.0): бібліотека для вимірювання часу виконання реалізованих алгоритмів, що є дуже важливим для їх аналізу ефективності та оцінки оптимізації;
- string-comparison (версія 1.3.0): бібліотека, що містить реалізації різних алгоритмів подібності рядків, частина з яких була використана під час проведення дослідження. Ця бібліотека користується досить високою популярністю, код знаходиться у відкритому доступі, тому перед інтегруванням і її використанням було порівняно вихідні коди інших

бібліотек. «string-similarity» найкраще серед інших використовує можливості JavaScript і досить точно реалізує алгоритми подібності, що робить її найкращим вибором у даному дослідженні.

2.3 Алгоритми подібності рядків

Для дослідження та порівняння алгоритмів подібності рядків було обрано наступні алгоритми:

- алгоритм знаходження косинусної подібності (Cosine similarity): даний алгоритм добре підходить для обчислення схожості між великими текстовими даними, оскільки використовує векторний підхід до представлення текстів;
- алгоритм знаходження відстані Левенштейна (Levenshtein distance): популярний алгоритм подібності рядків, вимірює мінімальну кількість операцій, необхідних для перетворення одного рядка в інший, що можна застосувати для знаходження дрібних відмінностей між текстами помилок;
- алгоритм знаходження індексу Жаккара (Jaccard index): алгоритм, який широко використовується під час тестування, особливо корисний для текстів із повторюваними підрядками;
- алгоритм знаходження подібності Джаро-Вінклера (Jaro-Winkler similarity): популярний алгоритм для групування помилок, особливо ефективний для групування помилок зі схожою символьною множиною;
- алгоритм знаходження коефіцієнта Соренсена-Дайса (Sorensen Dice): алгоритм, який схожий на знаходження індексу Жаккара, але обчислює подібність як відношення подвоєного перетину до суми розмірів множин, що робить даний алгоритм більш чутливим до повторюваних елементів у текстах.

Ці алгоритми мають найбільшу популярність у використані для знаходження подібності між рядками серед JavaScript розробників.

3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

У цьому розділі представлені загальні технічні аспекти, архітектура і структура коду основних компонентів, основні методи та функції, які використовувалися під час дослідження, а також використання описаних бібліотек на практиці.

3.1 Архітектура застосунку

Для проведення дослідження було створено базову архітектуру React-застосунків. Під час побудови архітектури використовувалися команди React CRA (Create React Application) [5-6] команди та середовище розробки Visual Studio Code [7].

Загальна архітектура проєкту представлена на рис.3.1.

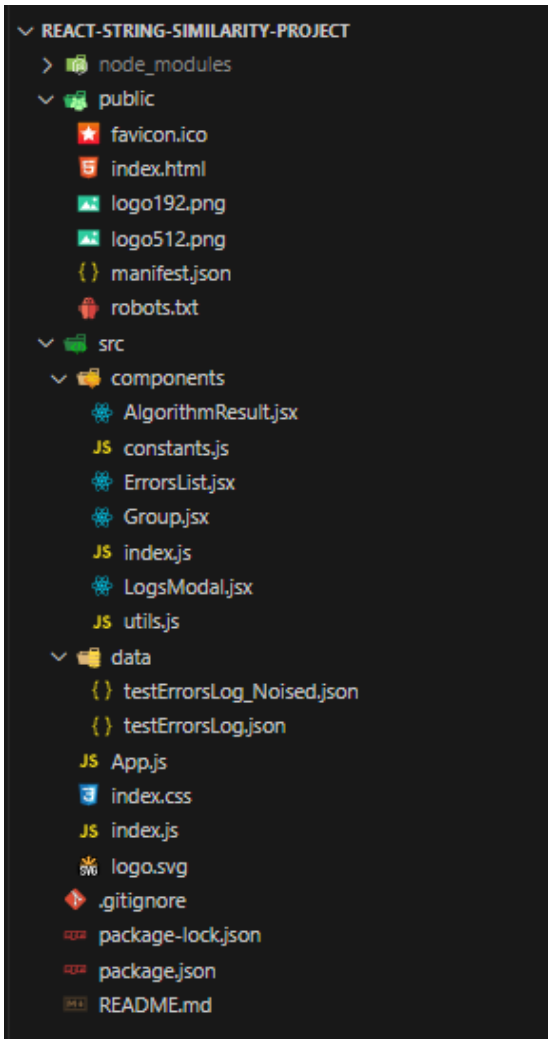


Рисунок 3.1 – Загальна архітектура проєкту (рисунок створено самостійно)

Основна директорія проєкту містить файли бібліотек, які зберігаються у папці «node_modules», скелет веб-застосунку (папка «public»), що містить основний HTML (HyperText Markup Language) файл («index.html») для інтеграції React, а також папка «src», яка зберігає основні React компоненти.

Файли «.gitignore», «package.json», «package-lock.json», «README.md» – це файли конфігурації проєкту, які містять в собі опис використаних бібліотек, загальну інформацію про проєкт та іншу технічну інформацію (див. рис. 3.1).

Основні файли компонентів, які використовуються для побудови веб-сторінки, а також інтеграції даних для дослідження знаходяться у директорії «src». Головними файлами є «index.js» та «App.js», куди за допомогою можливостей JavaScript імпортуються усі компоненти React із директорії «components».

Для проведення дослідження необхідно було зібрати дані, а саме – помилки, які потенційно можуть виникати під час тестування програмного продукту. Дані зберігаються у директорії «data» (файл «testErrorsLog.json») у вигляді множини об’єктів, де кожен об’єкт містить спеціально зареєстрований номер помилки (код) та повідомлення із текстом помилки (див. рис. 3.2).

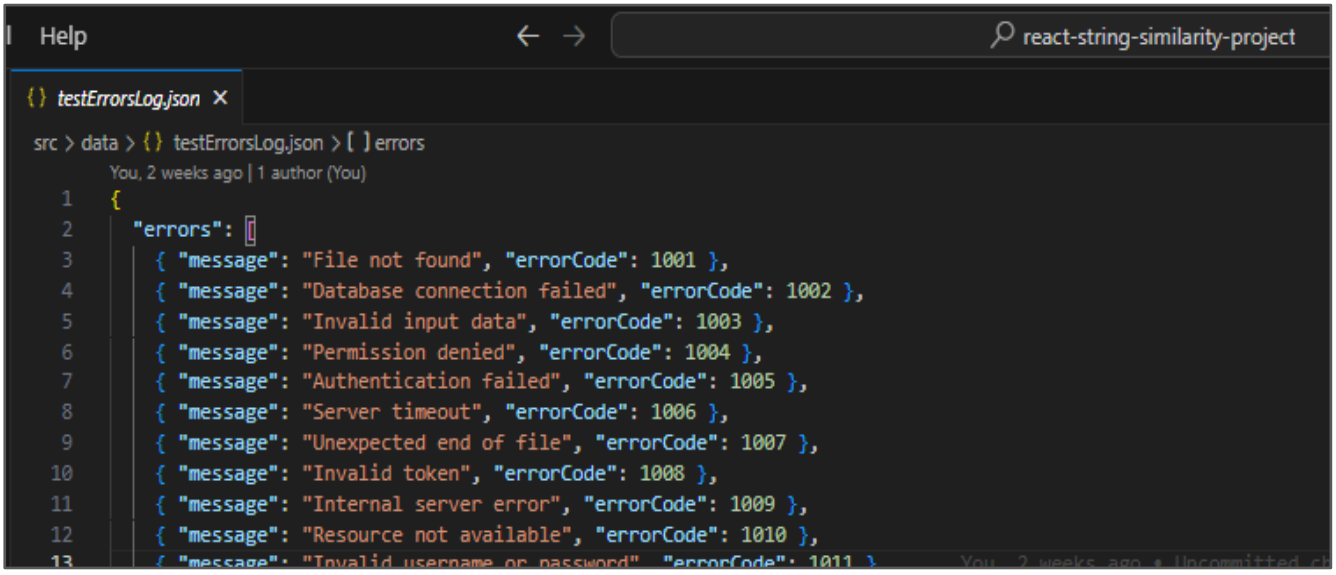


Рисунок 3.2 – Приклад даних для дослідження (рисунок створено самостійно)

Структура зібраних даних наближена до структури реальних помилок тестування, тому отримані результати використання алгоритмів максимально

наближені до результатів, які отримують розробники під час тестування у реальних проєктах.

3.2 Реалізація інтерфейсу користувача

Директорія «components» містить усі компоненти, які використовуються для побудови інтерфейсу користувача. Головним виконавчим файлом є «index.js», який застосовується для експорту компонентів із даної директорії.

Основним компонентом для створення макету сторінки відображення результатів роботи алгоритмів є «ErrorsList.jsx». Цей компонент поєднує в собі усі елементи і функціонал сторінки, її зовнішній вигляд та поведінку. Завдяки можливостям React можна застосовувати HTML та JavaScript разом.

Даний компонент має декілька важливих елементів:

- основна структура: компонент забезпечує загальну організацію та розташування елементів на сторінці;
- заголовок сторінки: описує призначення сторінки роботи алгоритмів подібності;
- кнопка перегляду вихідних даних: використовується для перегляду даних помилок до їх групування;
- відображення результатів: для результату кожного алгоритму відображається унікальний елемент у структурі сторінки;
- кнопка запуску роботи алгоритмів: запускає основну реалізацію алгоритмів подібності та показує індикатор очікування під час процесу групування;
- модальне вікно: відображає вікно з вихідними даними помилок, яке відкривається шляхом натискання на відповідно кнопку.

Усі елементи даного компонента об’єднані в єдину структуру в межах методу «return», що дозволяє ефективно керувати їх відображенням та взаємодією (див. рис. 3.3).

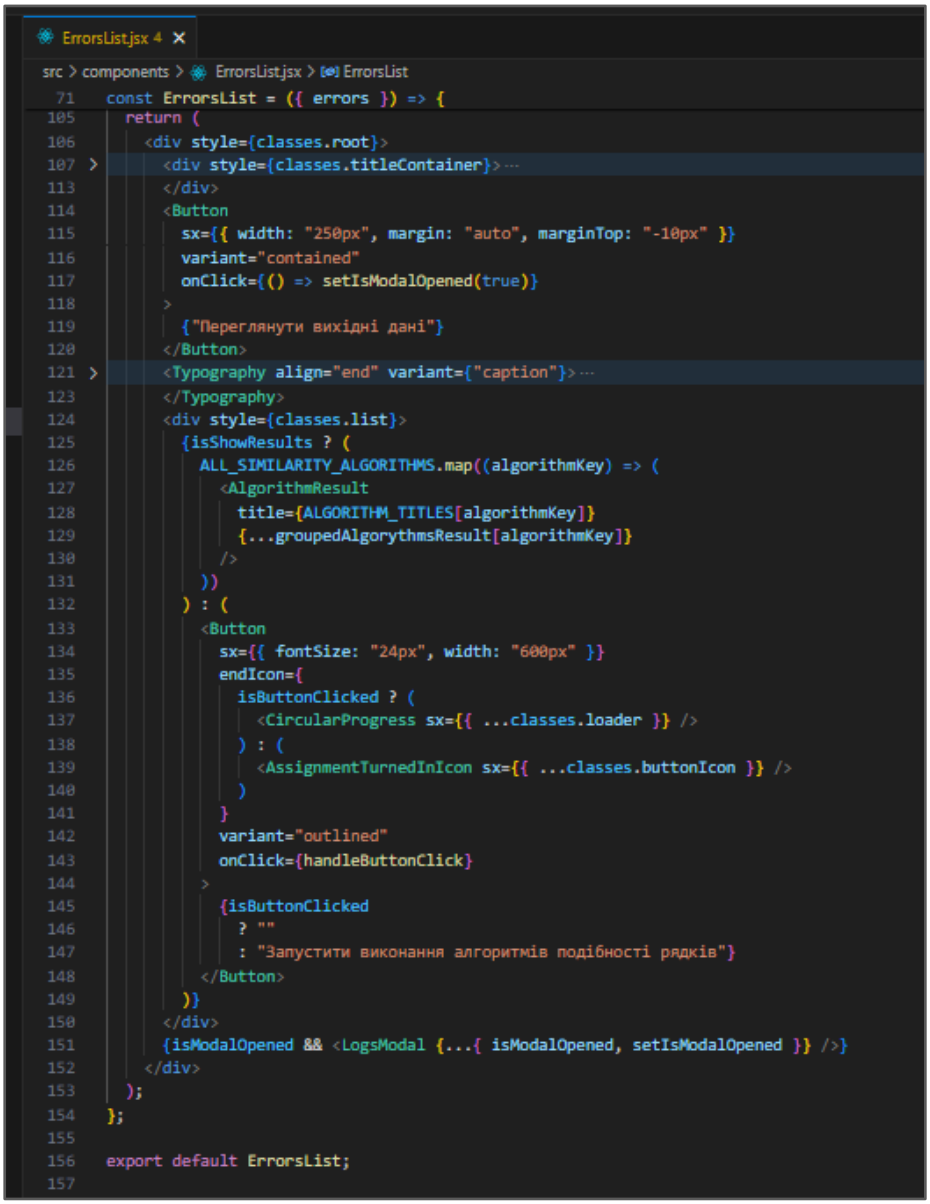


Рисунок 3.3 – Головний макет сторінки (рисунок створено самостійно)

Інші компоненти відповідають за відображення елементів на сторінці. Файл «LogsModal.jsx» відповідає за відображення модального вікна із вихідними даними, файл «AlgorithmResult.jsx» – за відображення результату роботи алгоритмів та інформації про час виконання і обсяг використаних ресурсів, а файл «Group.jsx» відповідає за відображення конкретної групи помилок у вигляді списку.

Для збереження даних, значення яких не змінюється протягом життєвого циклу ПЗ, використовується файл «constants.js», де зберігаються значення похибки («THRESHOLD_VALUE»), необхідної для визначення очікуваної

точності даних при знаходженні подібності, назви алгоритмів та інші константи, які використовуються в основних компонентах (див. рис. 3.4).

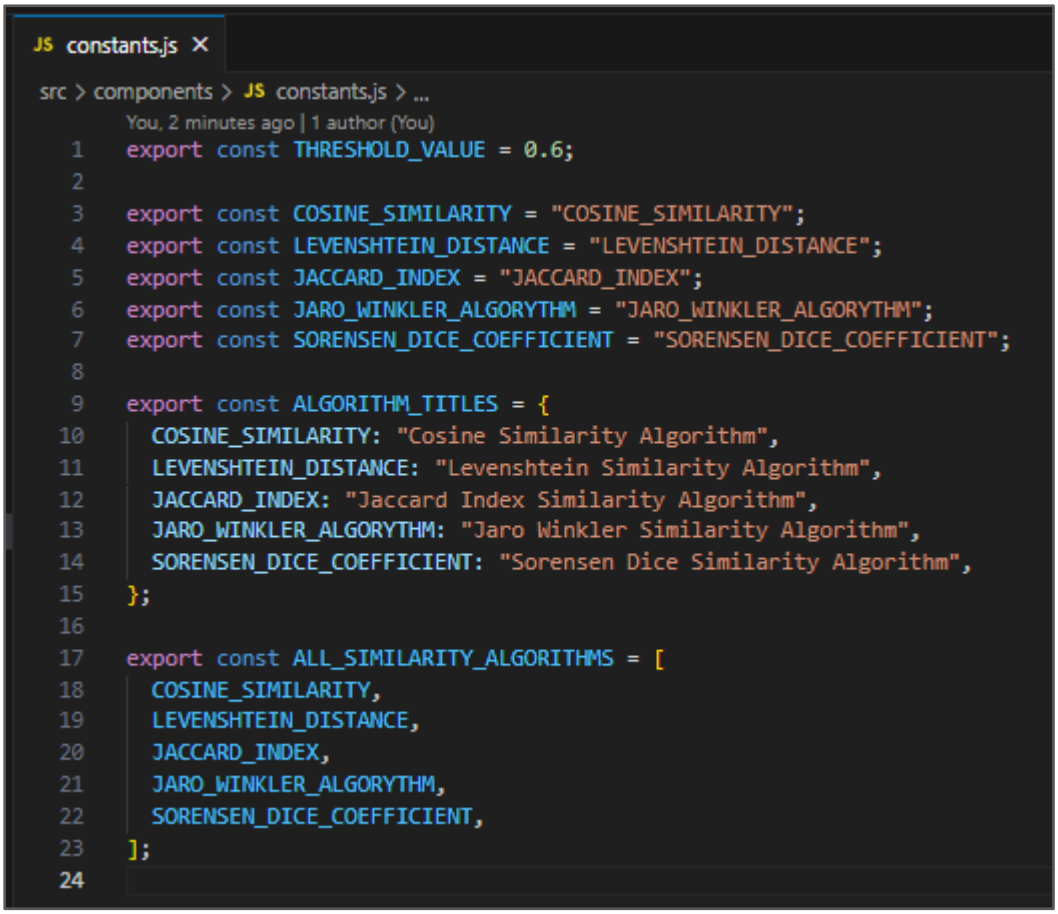
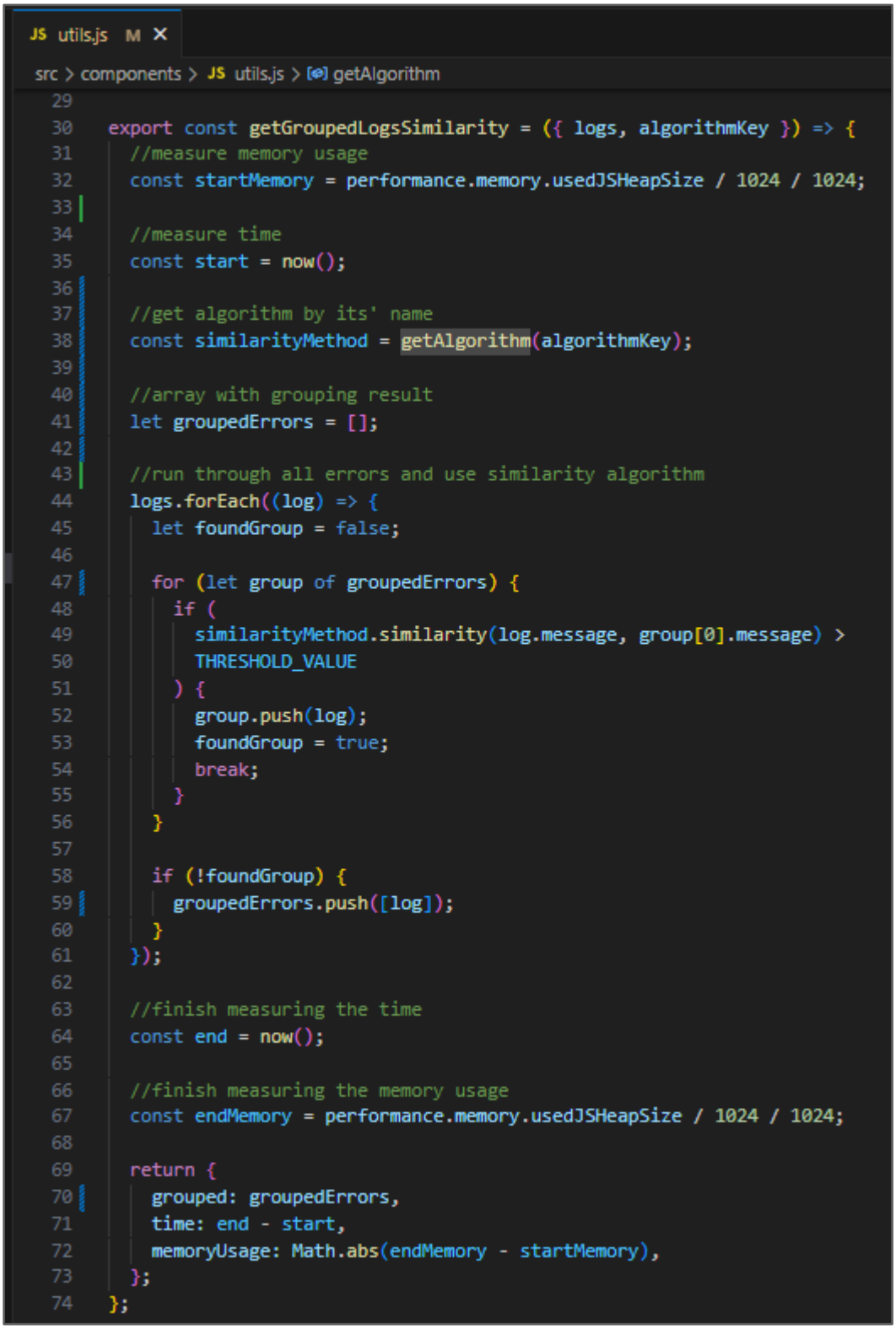


Рисунок 3.4 – Файл незмінних значень (рисунок створено самостійно)

Спеціальні функції та методи, які не мають прямого відношення до конкретного компоненту React, але надають необхідну компонентам інформацію чи дані, розташовані у файлі «utils.js».

3.3 Реалізація алгоритмів подібності рядків

Основна функція яка реалізує виконання алгоритму подібності, знаходиться у файлі «utils.js» під назвою «getGroupedLogsSimilarity». Вона приймає два параметри: «logs» – множина помилок, які необхідно згрупувати, та «algorithmKey» – визначає, який саме алгоритм необхідно застосувати (див. рис. 3.5).



```
JS utils.js M ×
src > components > JS utils.js > [e] getAlgorithm
29
30  export const getGroupedLogsSimilarity = ({ logs, algorithmKey }) => {
31    //measure memory usage
32    const startMemory = performance.memory.usedJSHeapSize / 1024 / 1024;
33
34    //measure time
35    const start = now();
36
37    //get algorithm by its' name
38    const similarityMethod = getAlgorithm(algorithmKey);
39
40    //array with grouping result
41    let groupedErrors = [];
42
43    //run through all errors and use similarity algorithm
44    logs.forEach((log) => {
45      let foundGroup = false;
46
47      for (let group of groupedErrors) {
48        if (
49          similarityMethod.similarity(log.message, group[0].message) >
50          THRESHOLD_VALUE
51        ) {
52          group.push(log);
53          foundGroup = true;
54          break;
55        }
56      }
57
58      if (!foundGroup) {
59        groupedErrors.push([log]);
60      }
61    });
62
63    //finish measuring the time
64    const end = now();
65
66    //finish measuring the memory usage
67    const endMemory = performance.memory.usedJSHeapSize / 1024 / 1024;
68
69    return {
70      grouped: groupedErrors,
71      time: end - start,
72      memoryUsage: Math.abs(endMemory - startMemory),
73    };
74  };
```

Рисунок 3.5 – Функція групування помилок (рисунок створено самостійно)

До кожної операції додано пояснення у вигляді коментарів зеленого кольору, яке коротко описує ціль кожної операції.

Основні кроки виконання даної функції:

- фіксація моментів початку вимірювання часу виконання та обсягу використаних ресурсів: змінна «startMemory» зберігає значення обсягу використаної пам’яті перед застосуванням алгоритму подібності за допомогою інтерфейсу JS – «performance». Подібним чином фіксуємо позначку початку вимірювання часу у змінній «start» за допомогою бібліотеки «performance-now» та метода «now»;
- отримання обраного алгоритму: завдяки параметру «algorithmKey» та функції «getAlgorithm» отримаємо функцію знаходження подібності. Функція «getAlgorithm» розроблена для збереження чистоти коду. Вона досить проста у реалізації – отримує відповідний до ключа метод алгоритму;
- створення множини: оголошуємо змінну «groupedErrors» для збереження множин згрупованих помилок;
- обчислення подібності рядків: для проходження по всім помилкам у множині використовується цикл «forEach», де на кожній ітерації виконується функція алгоритму для знаходження подібності між поточною помилкою множини із вже згрупованими. Значення подібності отримане у результаті виконання функції порівнюється із значенням похибки («THRESHOLD_VALUE»), і якщо дане значення перевищує похибку – це означає, що знайдена група для даної помилки, якщо ні – то поточна помилка додається до результату як нова група;
- фіксація моментів кінця вимірювання часу виконання та обсягу використаних ресурсів: аналогічно фіксації початку маємо змінну «endMemory» для збереження значення обсягу використаної пам’яті після застосування алгоритму подібності, а також змінну «end», яка фіксує позначку кінця вимірювання часу;
- повернення результатів: після групування помилок, функція «getGroupedLogsSimilarity» повертає результат групування до основного компоненту «ErrorList», де отримані дані використовуються для відображення результатів знаходження подібності на веб-сторінці. Слід

зазначити, що показники часу та обсягу використаних ресурсів дорівнюють різницям їхніх початкових моментів від їхніх кінцевих.

Таким чином у результаті виконання функції «getGroupedLogsSimilarity» отримуємо згруповані обраним алгоритмом помилки, а також значення часу виконання та обсягу використаних ресурсів [8-9].

4 ОПИС ЕКСПЕРИМЕНТАЛЬНИХ ДОСЛІДЖЕНЬ

4.1 Підготовка даних та налаштування

Метою експериментального дослідження є доведення або спростування результату вирішення багатокритеріальної задачі, а саме, що алгоритм Левенштейна є дійсно кращим вибором серед найпопулярніших алгоритмів подібності рядків.

Для проведення експерименту було згенеровано 209 помилок, які треба згрупувати п’ятьма різними алгоритмами подібності рядків та оцінити за наступними критеріями: час виконання, обсяг використаних ресурсів, точність результату, стійкість до шуму в даних та вартість.

Переглянути вихідні дані можна за допомогою кнопки «Переглянути вихідні дані», що знаходиться під титульною назвою сторінки (див. рис. 4.1).

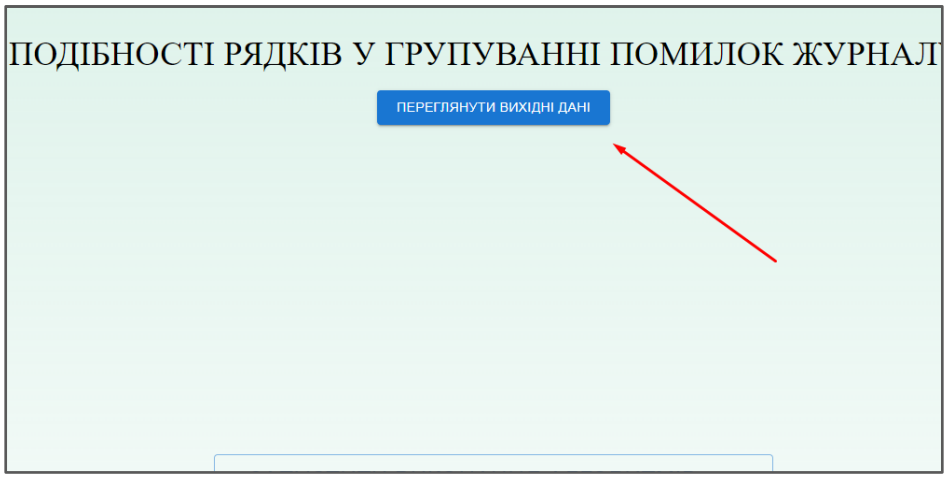


Рисунок 4.1 – Кнопка отримання вихідних даних (рисунок створено самостійно)

Після натискання відобразиться модальне вікно, в якому дані представлені у вигляді списку. Код помилки відображений червоним кольором, а повідомлення помилки – чорним (див. рис. 4.2).

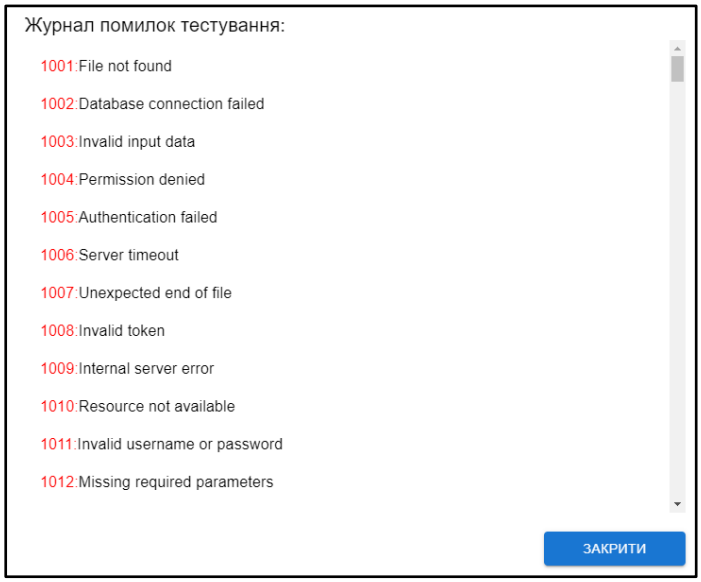


Рисунок 4.2 – Модальне вікно вихідних даних (рисунок створено самостійно)

Усього у вікні вихідних даних знаходиться 209 помилок, які використовуються алгоритмами для групування.

4.2 Виконання алгоритмів

Для того, щоб отримати результати роботи алгоритмів, необхідно натиснути на кнопку «Запустити виконання алгоритмів подібності рядків», яка розташована в центральній частині сторінки (див. рис. 4.3).

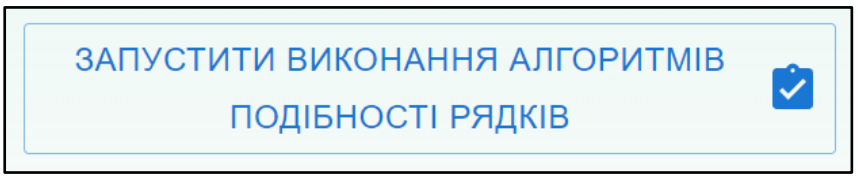


Рисунок 4.3 – Кнопка запуску алгоритмів (рисунок створено самостійно)

Після натискання кнопки, процес обробки вихідних даних запускається окремо кожним алгоритмом. Після завершення роботи всіх алгоритмів, сторінка відображає результати у вигляді п’яти списків, де кожен елемент списку представляє собою утворену групу помилок. Кожний список підписаний назвою застосованого алгоритму. Результати групування можна подивитися окремо для кожного алгоритму у відповідній області (див. рис. 4.4).

Рисунок 4.4 – Результати виконання алгоритмів подібності рядків у групуванні помилок журналу тестування (рисунок створено самостійно)

Розглянемо результати виконання алгоритмів. У кожній групі результатів відображений час, витрачений на роботу алгоритму у форматі мілісекунд (ms), кількість утворених груп помилок, а також кількість витрачених ресурсів пам’яті у форматі мегабайтів (MB). Слід зазначити, що якщо кількість витрачених ресурсів пам’яті дорівнює 0.00 MB, це означає, що обсяг ресурсів витрачений на виконання операцій групування, є незначним. Наприклад у результаті виконання алгоритму знаходження косинусної подібності утворилося шість груп помилок за 1.7 мілісекунди, а кількість витрачених ресурсів пам’яті є незначною, тому маємо відповідний результат (див. рис. 4.5).

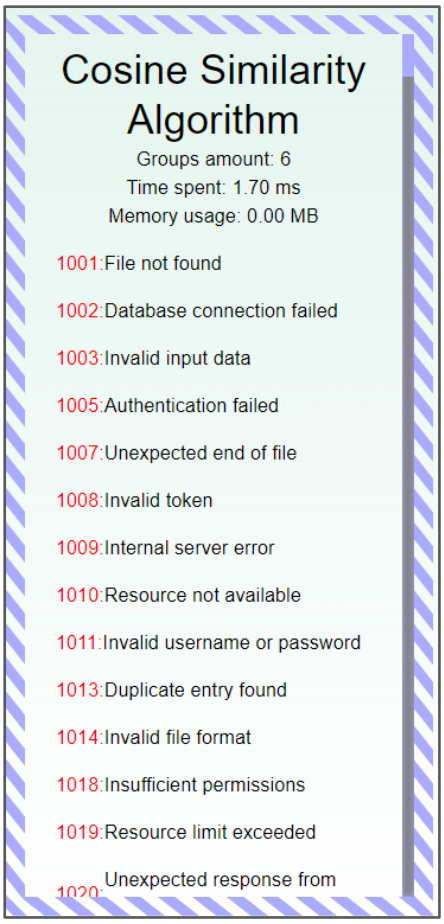


Рисунок 4.5 – Результат виконання алгоритму знаходження косинусної подібності (рисунок створено самостійно)

Незначний обсяг використаних ресурсів мають наступні алгоритми: косинусної подібності, Джаро-Вінклера, Соренсена-Дайса, що зменшує вартість впровадження даних алгоритмів.

4.3 Перевірка стійкості до шуму

Протестуємо виконання алгоритмів на стійкість до шуму. Для цього необхідно додати «шум» до вихідних даних помилок. Створимо новий файл «testErrorsLog_Noised.json», у якому оригінальні дані помилок перетворимо, додавши зайві символи та зробимо заміну деяких символів на числа (див. рис. 4.6).

Рисунок 4.6 – Оригінальні дані з доданням шуму (рисунок створено самостійно)

Використаємо створені дані для перевірки на стійкість до шуму. Після повторного запуску виконання алгоритмів отримали наступні результати (див. рис. 4.7).

Рисунок 4.7 – Результат групування помилок з доданням шуму (рисунок створено самостійно)

Перевіривши отримані результати, можемо зазначити, що кількість груп, створених алгоритмом знаходження косинусної подібності, зросла до восьми, що свідчить про нестійкість алгоритму до шуму. Однак утворені групи є кращими, ніж в оригінальних даних, щодо точності. Тому стійкість до шуму у даного алгоритму можна оцінити як «середня».

Кількість утворених груп при застосуванні алгоритму знаходження індексу Жаккара збільшилася вдвічі. Також можемо спостерігати, що час виконання даного алгоритму збільшився, а точність – зменшилася.

Алгоритм знаходження подібності Джаро-Вінклера показує досить непогані результати, які мають незначні відмінності від оригінальних, що свідчить про високу стійкість до шуму.

У результаті виконання алгоритму Левенштейна для даних із шумом утворилося додатково 20 груп, але відмінність від оригінальних груп мінімальна, що свідчить про те, що алгоритм Левенштейна дуже стійкий до шуму. Дуже схожий результат показує алгоритм знаходження коефіцієнта Соренсена-Дайса. На відмінну від обробки оригінальних даних, поточний обсяг використаних ресурсів збільшився.

4.4 Аналіз отриманих результатів

Для оцінки точності виконання групування оригінальних даних використаємо формулу 4.1:

$$\text{точність} = \frac{\sum_{\text{правильних результатів}} * \ 100\%}{\sum_{\text{помилок}}} \tag{4.1}$$

де $\sum_{правильних\ результатів}$ – сума результатів, які знаходяться у групі, сусідні результати якої є семантично споріднені або схожі;

$\sum_{помилок}$ – сумарна кількість помилок у вихідних даних, яка становить 209 помилок.

Після обчислення точності та загального аналізу отриманих результатів

створимо таблицю оцінки алгоритмів за метриками (див. табл. 4.1).

Таблиця 4.1 – Результат оцінки алгоритмів

Назва алгоритму	Точність даних (%)	Швидкість обробки	Обсяг використаних ресурсів	Стійкість до шуму	Вартість
CosSimilarity	62,2	Дуже Висока	Низький	Середня	Низька
Levenshtein	96,17	Низька	Високий	Дуже висока	Висока
JaccardIndex	79,1	Середня	Середній	Низька	Середня
JaroWinkler	95,8	Дуже Висока	Низький	Висока	Низька
SorensenDice	94,1	Висока	Низький	Дуже висока	Низька

Найвищу точність даних демонструє алгоритм Левенштейна (96,17%), що робить його найбільш точним серед досліджуваних алгоритмів. Однак, цей алгоритм має найнижчу швидкість обробки та використовує значну кількість ресурсів, що може бути неприйнятним у ресурсозатратних середовищах або у випадках, де швидкість обробки є критично важливою.

Алгоритм Джаро-Вінклера та Соренсена-Дайса також демонструють високу точність (95,8% та 94,1% відповідно) при значно вищій швидкості обробки та нижчому споживанню ресурсів, що робить їх привабливими для багатьох застосувань.

Алгоритм знаходження косинусної подібності показав найгіршу точність (62,2), що значно обмежує його застосування у задачах, де точність є критичним показником.

На основі отриманих результатів можна сказати, що алгоритм знаходження подібності Джаро-Вінклера – це універсальний алгоритм, який при низькій вартості реалізації забезпечує кращий баланс між точністю та ефективністю. Тому даний алгоритм може бути рекомендований для знаходження подібностей між помилками у журналі тестування.

ВИСНОВКИ

В роботі проведено детальний аналіз предметної галузі, описано найпопулярніші алгоритми подібності рядків та визначено основні цілі дослідження. За допомогою вирішення багатокритеріальної задачі та програмної реалізації алгоритмів було визначено оптимальний алгоритм для групування помилок у журналі тестування.

Виходячи з отриманих результатів, теоретичний аналіз дещо відрізняється від практичного результату у ході експерименту. Алгоритм Левенштейна досить повільно виконує групування вихідних даних, а при зростанні кількості помилок – час на виконання та обсяг використаних ресурсів буде збільшуватися у геометричній прогресії, що збільшує вартість реалізації даного алгоритму. Експериментально було доведено, що точність алгоритму Левенштейна більша ніж було визначено теоретично, навіть сягає найбільшого показника серед інших досліджуваних алгоритмів, а стійкість до шуму залишається дуже високою. Як висновок щодо даного алгоритму можна сказати, що для випадків, коли для тестування важлива точність отриманих результатів, то алгоритм Левенштейна найкраще підходить на дану роль, але як було зазначено, він поступається швидкістю та обсягом використаних ресурсів, тому можемо виключати можливість вважати даний метод найкращим для групування помилок журналу тестування.

Друге місце за точністю, але не останнє за загальною оцінкою займає алгоритм знаходження подібності Джаро-Вінклера. Значення точності згрупованих помилок сягає 95,8%, що є дуже високим показником для простого у реалізації алгоритму. Даний алгоритм дуже швидкий, має високу стійкість до шуму та низький обсяг використаних ресурсів, що робить його найдешевшим і в той же час найпростішим у реалізації.

Як результат, можна сказати, що алгоритм знаходження подібності Джаро-Вінклера – є найкращим серед найпопулярніших алгоритмів подібності рядків, які використовують розробники JavaScript.

Слід зазначити, що практичні результати можуть відрізнятися в залежності

від багатьох факторів: середовища тестування, мови програмування, потужності обчислювальної машини, а також способу реалізації. Тому результат даного дослідження може відрізнятися в залежності від зміни даних факторів.

Експериментальним шляхом було доведено, що алгоритм Джаро-Вінклера є найефективнішим методом для групування помилок у журналі тестування. Даний результат має значний вплив на розробку програмних застосунків та комерційну діяльність. Завдяки впровадженню цього алгоритму у широкому спектрі проєктів, можна очікувати підвищення точності та ефективності обробки помилок, що призведе до скорочення часу на тестування і їх виправлення. У свою чергу, це забезпечить швидший вихід продуктів на ринок, підвищить їх якість та знизить витрати на розробку.

Отже, впровадження алгоритму Джаро-Вінклера для групування помилок у журналі тестування стане вагомим кроком до оптимізації процесу розробки програмних продуктів та підвищення ефективності комерційної діяльності компаній.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Healthy relationship between Tester and Developer. URL: https://www.linkedin.com/pulse/healthy-relationship-between-qa-developer-mohd-shariq-lyp9c (дата звернення: 05.04.2024).

2. Sergiy Zagorodnyuk, Bohdan Sus, Ilona Revenchuk, Oleksandr Bauzha Information Security of Users Rights Assignment via the Software Solutions Based on LDAP // Problem of Infocommunications. Science and Technolpgy (PIC S&T′2020), Kharkiv, Ukraine- 6-9 October 2020.

3. Testing in Software Engineering. What is it and why do we do it? | by Jhonny Chamoun | Medium. URL: https://jhonny-chamoun.medium.com/testing-in-software-engineering-d32e36ceb30 (дата звернення: 07.04.2024).

4. Dynamic Programming vs Divide-and-Conquer - DEV Community. URL: https://dev.to/trekhleb/dynamic-programming-vs-divide-and-conquer-218i (дата звернення: 11.04.2024).

5. Quick Start – React. URL: https://react.dev/learn (дата звернення: 30.04.2024).

6. Getting Started | Create React App. URL: https://create-react-app.dev/docs/getting-started (дата звернення: 30.04.2024).

7. React JavaScript Tutorial in Visual Studio Code. URL: https://code.visualstudio.com/docs/nodejs/reactjs-tutorial (дата звернення: 20.05.2024).

8. TIOBE Index - TIOBE. URL: https://www.tiobe.com/tiobe-index/ (дата звернення: 22.05.2024).

9. Гриб Р.В., Ревенчук І.А. Дослідження методів та інструментів для групування помилок журналу тестування за допомогою алгоритмів подібності рядків: матер VII Міжнар наук.-практ конф. "Scientific Research: Theoretical Foundations and Practical Applications". Відень 24-26.01.2024. P.102-105 URL: https://isu-conference.com/wp-content/uploads/2024/01/Scientific_research_theoretical_foundations_and_practical_applications.pdf (дата звернення: 25.05.2024).