ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Методика корекції слів у технічних текстах українською мовою з використанням лексичної валідації на основі словника»

на здобуття освітнього ступеня магістра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Сергій ШУЛЬГА
(підпис)

Виконав: здобувач вищої освіти групи ПДМ-62
Сергій ШУЛЬГА

Керівник: Богдан ХУДІК
доктор філософії (PhD)

Рецензент: _____

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Магістр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
«_____» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Шульзі Сергію Сергійовичу

1. Тема кваліфікаційної роботи: «Методика корекції слів у технічних текстах українською мовою з використанням лексичної валідації на основі словника»

керівник кваліфікаційної роботи Богдан ХУДІК, доктор філософії (PhD),

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «15» жовтня 2024 р. № 320.

2. Строк подання кваліфікаційної роботи «17» грудня 2024 р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, алгоритми автоматичної корекції тексту, параметри ефективності словникової перевірки, вимоги до точності виправлення помилок у багатомовному середовищі.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Дослідження методів автоматичної лексичної валідації та корекції тексту.
2. Аналіз технологій для автоматичного перемикання мовної розкладки клавіатури.

3. Розробка та валідація системи автоматичної корекції тексту з підтримкою багатомовного середовища

5. Перелік ілюстративного матеріалу: *презентація*

1. Мета, об’єкт та предмет дослідження.
2. Актуальність роботи.
3. Лексична валідація на основі словника.
4. Модифікований метод виправлення слова.
5. Алгоритм методики корекції слів.
6. Практичний результат.
7. Порівняльний аналіз розробленої методики з аналогами.

6. Дата видачі завдання«16» жовтня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури	16.10.2024-23.10.2024	
2	Вивчення матеріалів для аналізу сучасних методів автоматичної корекції тексту та перемикання мовної розкладки.	23.10.2024- 03.11.2024	
3	Дослідження алгоритмів словникової перевірки, статистичних моделей та методів машинного навчання.	03.11.2024-11.11.2024	
4	Аналіз особливостей впливу словникових алгоритмів на точність і швидкість виправлення тексту в багатомовному середовищі.	12.11.2024-19.11.2024	
5	Розробка та тестування алгоритмів автоматичної корекції тексту із застосуванням реляційних баз даних.	20.11.2024-27.11.2024	
6	Застосування запропонованого алгоритму в різних сценаріях текстового вводу та оцінка його ефективності.	28.11.2024-02.12.2024	
7	Оформлення роботи: вступ, висновки, реферат	03.12.2024-06.12.2024	
8	Розробка демонстраційних матеріалів	07.12.2024-08.12.2024	

Здобувач вищої освіти _____ Сергій ШУЛЬГА
(підпис)

Керівник
кваліфікаційної роботи _____ Богдан ХУДІК
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 98 стор., 7 табл., 16 рис., 22 джерела.

Мета роботи – Підвищення ефективності та точності процесу введення тексту в багатомовному середовищі шляхом оптимізації методів автоматичної лексичної валідації та корекції слів, а також забезпечення автоматичного перемикання мовної розкладки клавіатури.

Об'єкт дослідження – Процес введення тексту користувачем та його автоматичної корекції у багатомовному середовищі.

Предмет дослідження – Методи та алгоритми автоматичної лексичної валідації тексту, корекції слів на основі словника, а також технології автоматичного перемикання мовної розкладки клавіатури.

У роботі використано різноманітні методи, такі як словникова перевірка, алгоритми автоматичного перемикання мовної розкладки, а також емуляція клавіатурного введення для виправлення тексту. Особливу увагу приділено використанню реляційних баз даних для оптимізації пошуку слів, що значно підвищує швидкість та точність роботи системи.

Проведено аналіз сучасних методів автоматичної корекції тексту, включаючи словникові підходи, статистичні моделі та алгоритми машинного навчання. У роботі детально розглянуто переваги та недоліки таких популярних інструментів, як Grammarly Desktop, SwiftKey та Gboard, а також їхню ефективність у різних сценаріях.

Розроблено та оптимізовано алгоритм автоматичної корекції тексту, що включає інтеграцію локальних словників, швидке перемикання мовної розкладки та друк виправлених слів. Запропоноване рішення забезпечує автономну роботу без залежності від Інтернет з'єднання, що робить його універсальним і надійним інструментом для багатомовного середовища.

Проведено експерименти для оцінки продуктивності розробленої системи. Здійснено порівняння з ручними методами виправлення тексту та популярними

комерційними продуктами. Результати показали, що запропоноване рішення значно скорочує час виправлення помилок, підвищує точність введення тексту та знижує кількість допущених помилок.

КЛЮЧОВІ СЛОВА: АВТОМАТИЧНА КОРЕКЦІЯ ТЕКСТУ, БАГАТОМОВНЕ СЕРЕДОВИЩЕ, ЛЕКСИЧНА ВАЛІДАЦІЯ, СЛОВНИКОВІ МЕТОДИ, ПЕРЕМИКАННЯ РОЗКЛАДКИ КЛАВІАТУРИ, АВТОНОМНА СИСТЕМА, ОБРОБКА ТЕКСТУ.

ABSTRACT

Text part of the master's qualification work: 98 pages, 7 tables, 16 figures, 22 sources.

The purpose of the work – to increase the efficiency and accuracy of text input in a multilingual environment by optimizing methods for automated lexical validation and word correction, as well as ensuring automatic switching of the keyboard layout.

Object of research – the process of text input by a user and its automatic correction in a multilingual environment.

Subject of research – methods and algorithms for automated lexical text validation, dictionary-based word correction, and technologies for automatic switching of the keyboard layout.

Summary of the work:

Various methods have been employed in this study, including dictionary checking, automatic switching of keyboard layouts, and keyboard input emulation for text correction. Particular attention is devoted to the use of relational databases for optimizing word search, which significantly increases system speed and accuracy.

An analysis of modern automatic text correction methods has been conducted, encompassing dictionary-based approaches, statistical models, and machine learning algorithms. The advantages and disadvantages of popular tools such as Grammarly Desktop, SwiftKey, and Gboard, along with their effectiveness in different scenarios, are thoroughly examined.

An algorithm for automated text correction has been developed and optimized, integrating local dictionaries, rapid switching of keyboard layouts, and printing of corrected words. The proposed solution ensures autonomous operation without dependence on an Internet connection, making it a universal and reliable tool for a multilingual environment.

Experiments have been carried out to evaluate the performance of the developed system. A comparison with manual text correction methods and popular commercial products shows that the proposed solution significantly reduces the time required for error correction, increases text input accuracy, and decreases the number of mistakes made.

KEYWORDS: AUTOMATIC TEXT CORRECTION, MULTILINGUAL ENVIRONMENT, LEXICAL VALIDATION, DICTIONARY METHODS, KEYBOARD LAYOUT SWITCHING, AUTONOMOUS SYSTEM, TEXT PROCESSING.

ЗМІСТ

ВСТУП

У сучасному світі інформаційні технології стали невід'ємною частиною повсякденного життя [1]. Швидкість та ефективність обробки інформації мають критичне значення в різних сферах діяльності, від бізнесу та освіти до науки та особистого спілкування [2]. Одним із ключових елементів взаємодії людини з комп'ютером є процес введення тексту.

З розвитком багатомовності та глобалізації виникає потреба у використанні декількох мов у повсякденній роботі. Часто користувачі змушені перемикатися між різними мовами вводу клавіатури. Проте в процесі роботи вони можуть забувати змінити розкладку клавіатури, що призводить до введення некоректних символів, слів, речень, або навіть абзаців. Наприклад, замість бажаного слова українською мовою користувач отримує набір незрозумілих символів англійською, що знижує ефективність роботи та потребує додаткового часу на виправлення помилок.

Традиційні засоби, такі як ручне перемикання розкладки або використання стандартних автокоректорів, не завжди вирішують цю проблему ефективно. Вони вимагають постійної уваги від користувача та можуть відволікати від основних задач, оскільки кожне перемикання потребує часу, додаткових натискань клавіш і уважного контролю, особливо коли йдеться про багатомовні тексти. Якщо користувач одночасно працює з кількома документами чи додатками, зміна мовної розкладки може ускладнити швидку взаємодію з системою, адже після повернення до попереднього документа знову доведеться перемикатися. Такий режим роботи здатен стати додатковим стресовим фактором, особливо при дедлайнах або великому обсязі роботи. Крім того, традиційні автокоректори, вбудовані в офісні пакети чи браузери, не завжди надають коректну підказку, бо вони не враховують змішаний контекст: коли в одному й тому ж документі одночасно можуть зустрічатися слова з різних мов, власні імена, технічні терміни тощо. Це призводить до появи «помилкових виправлень» і ще більше знижує загальну ефективність роботи з текстом. Отже,

розробка автономного рішення, здатного виконувати автоматичну корекцію та керування розкладкою із урахуванням контексту і мовних особливостей, набуває особливої важливості. З’являється потреба у спеціалізованих підходах, орієнтованих не лише на орфографію, а й на використання відповідних мовних моделей, а також на розширення словникової бази до рівня, при якому система зможе «розуміти» не лише правильність написання, а і специфіку мовного середовища в режимі реального часу.

Вони вимагають постійної уваги від користувача та можуть відволікати від основних задач. Це актуалізує потребу в автоматизованих рішеннях, які б забезпечували корекцію введеного тексту та автоматичне перемикання мови вводу клавіатури без втручання користувача.

Актуальність теми полягає в необхідності розробки методів та засобів, які б підвищували зручність та ефективність введення тексту в багатомовному середовищі. Автоматична лексична валідація та корекція слів на основі словника, а також автоматичне перемикання мови вводу клавіатури можуть значно покращити користувацький досвід, зменшити кількість помилок та підвищити продуктивність роботи.

Дана робота присвячена аналізу та розробці автоматичного методу лексичної валідації та корекції слів на основі словника, який забезпечує автоматичне перемикання мови вводу клавіатури. У процесі дослідження проведено детальний аналіз існуючих методів, їх переваг та недоліків, а також запропоновано новий підхід для покращення процесу введення тексту.

Мета роботи – підвищення ефективності та точності процесу введення тексту користувачем шляхом розробки автоматизованого методу лексичної валідації та корекції слів на основі словника з автоматичним перемиканням мови вводу клавіатури.

Об’єкт дослідження – процес введення тексту користувачем на комп'ютері та проблеми, пов'язані з неправильним вибором мови вводу клавіатури.

Предмет дослідження – методи та алгоритми автоматичної лексичної валідації і корекції слів на основі словника, а також засоби автоматичного перемикання мови вводу клавіатури.

Для досягнення мети вирішувалися наступні завдання:

1. Літературний огляд. Проведено дослідження та аналіз існуючих наукових джерел, щоб отримати повне уявлення про різноманітні методи та технології, що використовуються в автоматичній корекції тексту та управлінні розкладкою клавіатури.

2. Аналіз існуючих методів та алгоритмів. Вивчено та проаналізовано сучасні математичні методи, моделі та алгоритми лексичної валідації та корекції слів, а також методи автоматичного перемикання мови вводу.

3. Моделювання та програмування. Розроблено алгоритм для автоматичної лексичної валідації та корекції слів на основі словника, а також механізм автоматичного перемикання мови вводу клавіатури. Виконано програмну реалізацію цих алгоритмів з використанням бази даних для зберігання словників.

4. Експериментальні дослідження. Проведено тестування розробленої системи на реальних та симульованих даних. Виконано різні експерименти з метою оцінки ефективності та точності методів лексичної валідації та корекції слів.

5. Аналіз результатів. Використано статистичні методи для оцінки та порівняння результатів. Визначено точність, швидкість та стабільність розробленого методу.

6. Оптимізація та удосконалення. На основі отриманих результатів вдосконалено методи та розроблено новий підхід для покращення точності, швидкості та надійності системи.

Вирішення цих завдань сприяє розвитку ефективних підходів у сфері автоматизації введення тексту та покращенню користувацького досвіду при роботі з багатомовними системами. Запропонований метод може бути застосований у різних програмних продуктах, що сприятиме підвищенню продуктивності та зручності роботи користувачів.

1 АНАЛІЗ ІСНУЮЧИХ ПІДХОДІВ І РІШЕНЬ У СФЕРІ АВТОМАТИЧНОЇ КОРЕКЦІЇ ТЕКСТІВ

1.1. Аналіз наукових джерел за темою дослідження

Розробка ефективних алгоритмів автоматичної корекції текстів є однією з ключових задач обробки природної мови. Аналіз наукових джерел показує, що основними підходами до вирішення цієї проблеми є словникові методи, статистичні моделі та алгоритми машинного навчання.

Словникові підходи базуються на порівнянні введених слів із попередньо створеним словником. Ці методи передбачають перевірку орфографії шляхом пошуку слова у базі даних. Якщо слово знайдено, воно вважається правильним; якщо ні — пропонується виправлення, найближче за значенням, наприклад, із використанням відстані Левенштейна. Приклад метрики відстані Левенштейна [5] можна побачити на рисунку 1.1.

		k	i	t	t	e	n
	0	1	2	3	4	5	6
s	1	1	2	3	4	5	6
i	2	2	1	2	3	4	5
t	3	3	2	1	2	3	4
t	4	4	3	2	1	2	3
i	5	5	4	3	2	2	3
n	6	6	5	4	3	3	2
g	7	7	6	5	4	4	3

Рис. 1.1 Метрика відстані Левенштейна

Переваги словникових методів:

- Простота реалізації: для роботи необхідний лише доступ до відповідного словника.

- Висока точність за умови якісного наповнення словника.

Недоліки:

- Залежність від якості та повноти словника.

- Неможливість врахування контексту, через що правильне слово у певному контексті може бути визначене як помилкове.

У роботах [1, 2] розглядаються алгоритми перевірки текстів за словниками. Відзначено, що ці методи забезпечують високу швидкість і точність, але мають обмеження у врахуванні контексту та багатомовності.

Сучасні словникові системи, як-от Hunspell чи Aspell, часто використовуються у текстових редакторах для перевірки орфографії, але їх ефективність обмежена при роботі в багатомовному середовищі або з текстами, що містять спеціалізовану лексику [3, 4]. Зокрема, під час одночасного введення слів з різних мов такі системи не завжди можуть своєчасно “зрозуміти”, якою мовою друкує користувач, а тому подають некоректні підказки або позначають слова як неправильні. Окрім того, словникові методи мають труднощі з лінгвістичним аналізом складних конструкцій: вони не враховують граматичні зв’язки між словами, не розпізнають правильне відмінювання чи відмінювання в контексті сусідніх слів. Якщо у документі зустрічається велика кількість неологізмів, професійних термінів чи регіоналізмів, типові словникові підходи можуть пропускати помилки або навпаки позначати правильно написані слова як неправильні. Це особливо критично в наукових дослідженнях, де часто використовується специфічна лексика, а також у бізнес-комунікаціях з англомовними запозиченнями і скороченнями. У світлі цього, посилюється інтерес до комбінованих підходів, які здатні інтегрувати словникові бази з адаптивними методами, наприклад, на основі статистичних моделей чи глибинного навчання. Такі рішення можуть формувати індивідуальні мовні моделі під конкретні потреби користувача, забезпечуючи більш точне і контекстно-залежне виправлення. Наразі

в наукових працях [1, 2, 3] можна простежити тенденцію до збільшення обсягів досліджень саме у напряму об’єднання кількох методів водночас: від класичних словникових і N-грамних — до сучасних нейронних мереж та трансформерів.

Статистичні моделі, зокрема N-грамні підходи [6], працюють шляхом аналізу частотності символів, слів або фраз у текстах. Наприклад, використання 5-грамних моделей дозволяє враховувати послідовність символів або слів для перевірки текстів. Ці моделі особливо корисні для виявлення помилок у багатомовних або спеціалізованих текстах, де словникові підходи є недостатньо ефективними. Схему N-грамної мовної моделі можна побачити на рисунку 1.2.

Переваги статистичних моделей:

- Можливість обробляти текст навіть за відсутності словника.
- Врахування частотних залежностей, що дозволяє підвищити точність.

Недоліки статистичних моделей:

- Залежність від великих обсягів навчальних даних.
- Значні обчислювальні витрати.

Дослідження [3] демонструє, що використання N-грамних моделей покращує корекцію тексту, але вимагає великих навчальних корпусів.

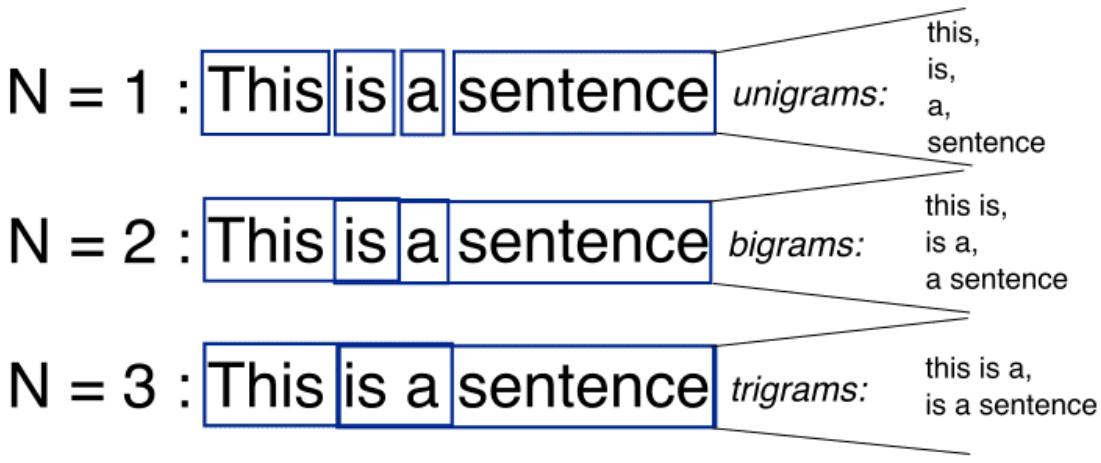


Рис. 1.2 N-грамна мовна модель

Статистичні підходи часто використовуються для створення мовних моделей у додатках передбачення тексту, таких як клавіатури Gboard або SwiftKey. Їх інтеграція з великими мовними корпусами забезпечує високу точність.

Методи машинного навчання, особливо на основі глибоких нейронних мереж, стали революційними у сфері корекції текстів [7, 8]. Моделі на кшталт GPT або BERT враховують контекст слів у реченні, що робить їх надзвичайно точними. Вони здатні не лише виправляти орфографічні помилки, але й враховувати граматичні особливості тексту.

Базовий алгоритм роботи мовної моделі нейронної мережі зображений на рисунку 1.3, де жовті кола - вхідні дані, сині та зелені кола - приховані слої, де вібдувається обробка вхідних даних, а червоне коло - вихідні дані.

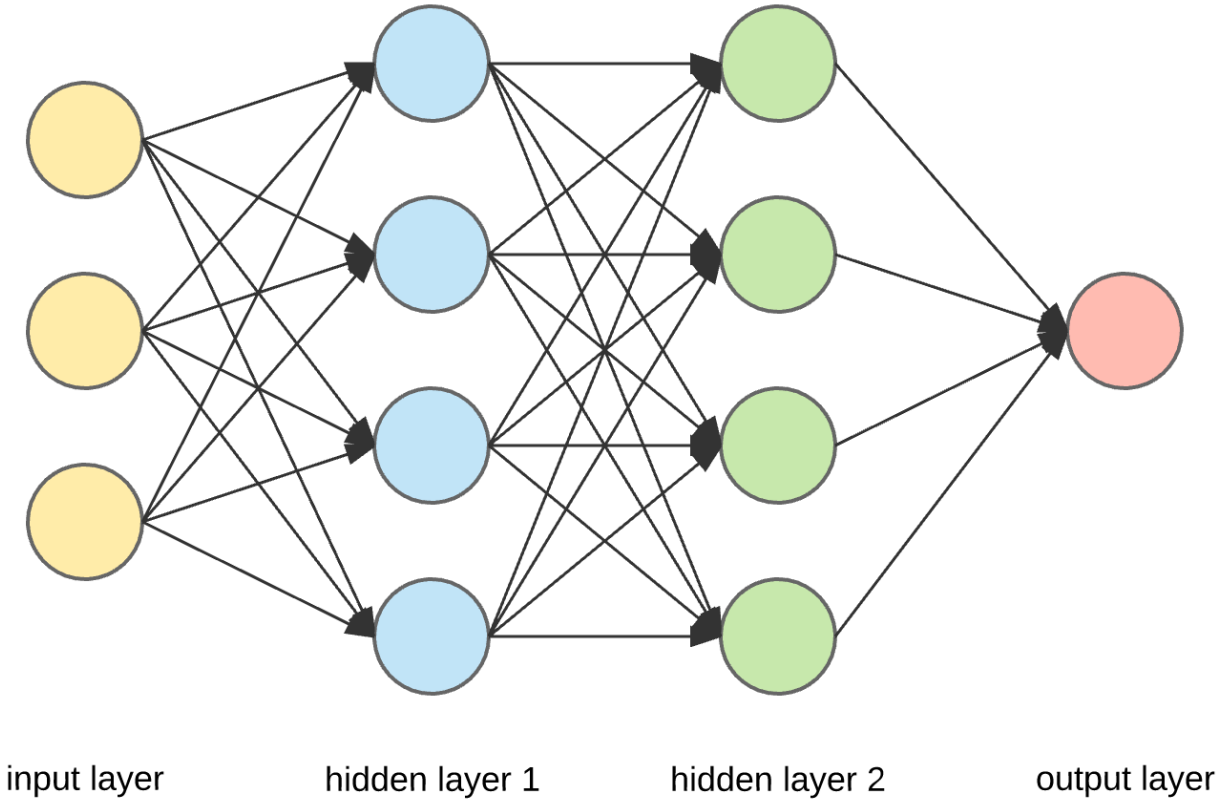


Рис. 1.3 Мовна модель нейронної мережі

Переваги:

- Висока точність завдяки врахуванню контексту.
- Адаптивність: моделі можуть оновлюватися та вдосконалюватися з часом.

Недоліки:

- Великі обчислювальні витрати, що ускладнює використання на локальних пристроях.

- Залежність від наявності високоякісних навчальних даних.

У публікаціях [4, 5] розглядаються можливості трансформерів, таких як BERT і GPT, які здатні враховувати контекст. Однак ці підходи вимагають значних обчислювальних ресурсів і доступу до мережі.

Сучасні програми, як Grammarly чи Microsoft Editor, інтегрують алгоритми машинного навчання для забезпечення найвищої якості перевірки текстів. Однак їхні закриті архітектури та вимоги до постійного доступу до інтернету створюють додаткові обмеження.

Окрему увагу приділено дослідженням, пов'язаним із багатомовними середовищами, наприклад, у [6], де розглядається адаптація систем до умов динамічної зміни мов.

Аналіз наукових джерел демонструє, що кожен із підходів має свої переваги та обмеження. Словникові методи є простими та швидкими, але не враховують контексту. Статистичні моделі дозволяють аналізувати частотні залежності, але потребують великих об’ємів текстів. Методи машинного навчання є найефективнішими, але їхня реалізація вимагає значних ресурсів.

Порівняння основних методів лексичної валідації можна побачити в таблиці 1.1.

Таблиця 1.1

Порівняння основних методів лексичної валідації

Метод \ Функція \	Словникові методи	Статистичні моделі	Машинне навчання
Залежність від інтернету	Ні	Ні	Так

Продовження таблиці 1.1

Порівняння основних методів лексичної валідації

Функція \ Метод	Словникові методи	Статистичні моделі	Машинне навчання
Врахування контексту	Ні	Так	Так
Залежність від великих обсягів навчальних даних	Ні	Так	Так
Значні обчислювальні витрати	Ні	Так	Так
Висока точність	Так	Так	Так
Простота реалізації	Так	Ні	Ні
Простота реалізації	Так	Ні	Ні

Для вирішення поставленої задачі роботи — створення алгоритму автоматичної корекції тексту та перемикання мовної розкладки — доцільно комбінувати словникові та статистичні підходи. Використання машинного навчання може бути перспективним на етапах вдосконалення системи.

1.2. Порівняльний аналіз існуючих рішень

У цьому розділі проведено аналіз існуючих програмних рішень, які реалізують функції автоматичної корекції тексту, передбачення слів та перемикання мовної розкладки.

Розглянуто популярні продукти, такі як SwiftKey та Gboard, з метою визначення їх переваг, недоліків та обмежень.

SwiftKey

SwiftKey — клавіатура, створена Microsoft, яка інтегрує функції автоматичної корекції тексту, передбачення слів і перемикання мов [9].

Переваги:

- Підтримка більше ніж 300 мов, що робить SwiftKey популярним у багатомовних середовищах.

- Інтелектуальне передбачення тексту на основі історії введення користувача.

- Інтеграція машинного навчання для врахування контексту.

Недоліки:

- Залежність від інтернету для синхронізації персоналізованих даних та оновлення моделей.

- Обмеження точності для спеціалізованої термінології або скорочень.

- Прив'язка до мобільних платформ (Android та iOS), що унеможливлює використання на стаціонарних системах.

SwiftKey має високу популярність серед мобільних користувачів, але її ефективність знижується в умовах відсутності доступу до інтернету. Користувацький інтерфейс SwiftKey зображений на рисунку 1.4.

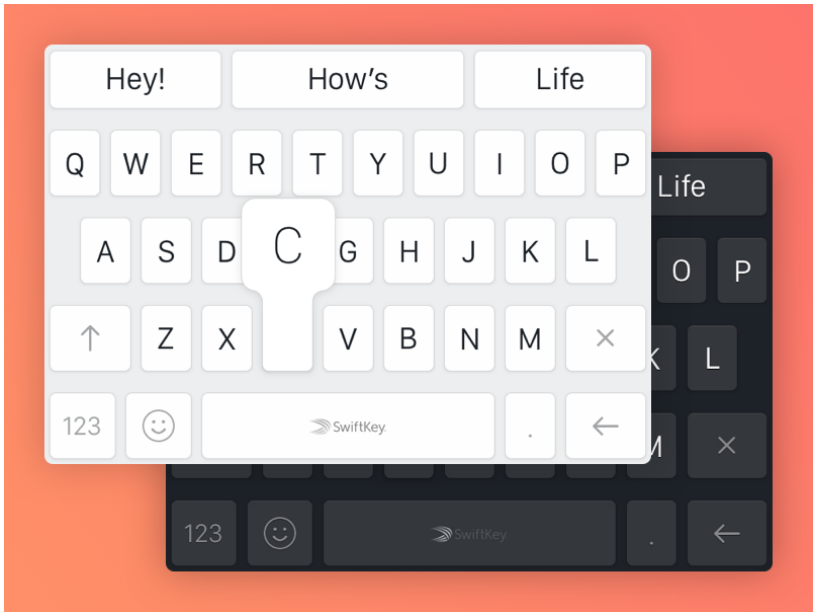


Рис. 1.4 Користувацький інтерфейс SwiftKey

Gboard

Gboard — клавіатура від Google, яка об’єднує функції автоматичної корекції тексту, передбачення слів, голосового введення та перемикання між мовами [10].

Переваги:

- Гнучкість у використанні: підтримка багатьох мов і функція одночасного набору тексту кількома мовами.

- Інтеграція з сервісами Google, такими як пошук та перекладач.

- Висока точність завдяки використанню алгоритмів машинного навчання.

Недоліки:

- Необхідність постійного доступу до інтернету для використання повного функціоналу.

- Питання конфіденційності: через інтеграцію з сервісами Google користувачі побоюються за безпеку своїх даних.

- Обмеження для спеціалізованих текстів: як і SwiftKey, Gboard не завжди коректно обробляє терміни чи професійний жаргон.

Gboard має великий набір функцій і є популярною клавіатурою для мобільних пристроїв, але її застосування також обмежене мобільними платформами.

Даний фактор робить неможливим користування даним застосунком користувачам будь-яких стаціонарних операційних систем, таких як Microsoft Windows, Apple Macintosh, Linux.

Приклад користувацього інтерфейсу Gboard зображений на рисунку 1.5.

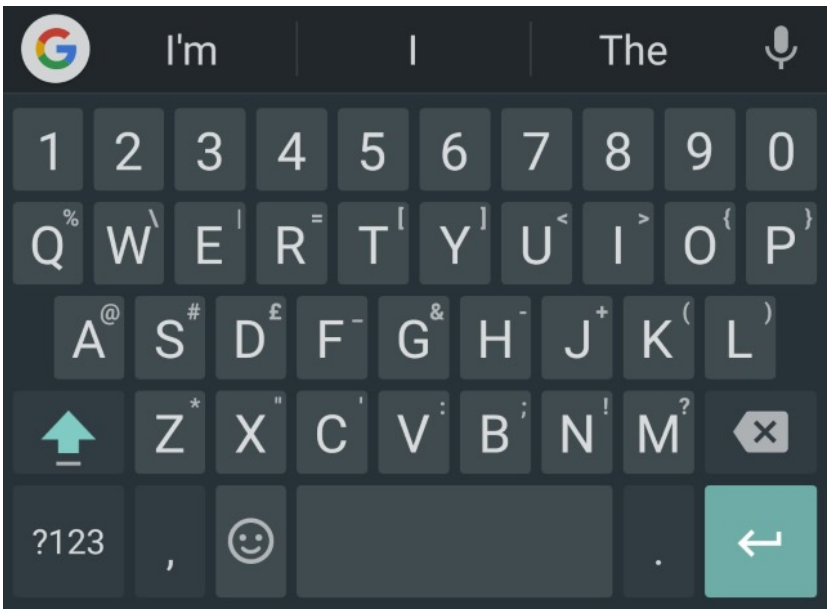


Рис. 1.5 Користувацький інтерфейс Gboard

Grammarly Desktop

Grammarly Desktop — популярне рішення для персональних комп'ютерів, яке забезпечує автоматичну перевірку тексту, виправлення помилок та вдосконалення стилю письма [11].

Переваги:

- Платформна гнучкість: Grammarly доступний для Windows та macOS, інтегрується з текстовими редакторами, браузерами та електронною поштою.

- Підтримка багатьох мов: Хоча основний акцент на англійській мові, інструмент пропонує багатомовну перевірку.

- Контекстне виправлення: Використовує машинне навчання для аналізу контексту тексту, пропонуючи вдосконалення стилю письма.

Недоліки:

- Залежність від підключення: Для доступу до розширених функцій, таких як вдосконалення стилю та перевірка на плагіат, потрібне підключення до інтернету.

- Обмежені можливості для спеціалізованої термінології: Для вузькопрофесійних текстів може знадобитися додаткова настройка словників.

- Комерційна модель: Більшість функцій доступні лише у преміум-версії.

Grammarly Desktop відмінно підходить для роботи на ПК, особливо для текстів англійською мовою.

Його інтеграція з офісними програмами та браузерами значно розширює можливості інструменту. Користувацький інтерфейс Grammarly Desktop представлено на рисунку 1.6.

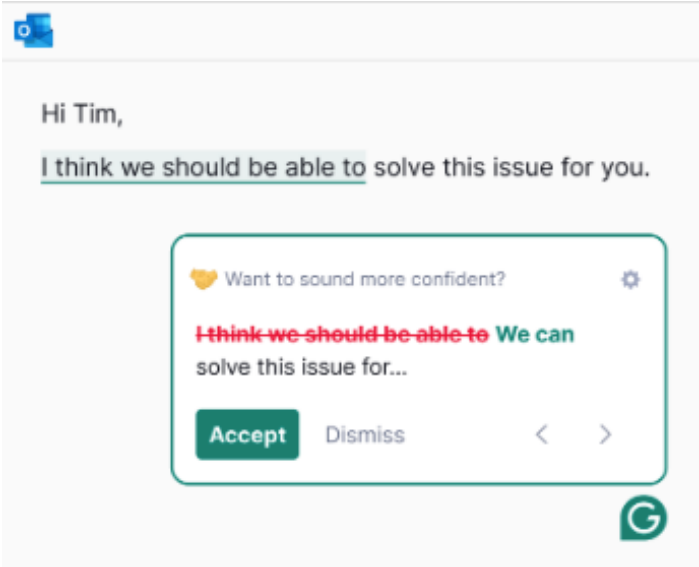


Рис. 1.6 Користувацький інтерфейс Grammarly Desktop

Порівняльний аналіз

В таблиці 1.2 наведено порівняльну характеристику розглянутих програмних продуктів:

Таблиця 1.2

Порівняльна характеристика відомих рішень

Застосунок / Функція	SwiftKey	Gboard	Grammarly Desktop
Багатомовна підтримка	Так	Так	Так

Продовження таблиці 1.2

Порівняльна характеристика відомих рішень

Застосунок Функція	SwiftKey	Gboard	Grammarly Desktop
Залежність від Інтернету	Так	Так	Так
Врахування контексту	Так	Так	Так
Автоматична зміна мови вводу клавіатури	Ні	Ні	Ні
Автоматичне виправлення неправильного слова з іншої мови вводу	Ні	Ні	Ні
Здатність додати спеціалізовані слова до загальної бази даних	Ні	Ні	Ні

Проаналізовані програмні рішення мають низку переваг, зокрема високу інтеграцію з сучасними платформами та підтримку багатьох мов [12]. Проте всі вони мають суттєві обмеження: залежність від інтернету, недостатня точність для спеціалізованих текстів, прив’язка до мобільної платформи, а також обмежений функціонал у багатомовному середовищі.

У рамках даної магістерської роботи пропонується розробити автономний алгоритм, який не залежить від підключення до мережі та забезпечує точну корекцію тексту на основі локальних словників. Це дозволить уникнути недоліків існуючих рішень та підвищить ефективність роботи у багатомовному середовищі.

1.3. Формулювання проблеми та завдань дослідження

Проблематика дослідження

Сучасні користувачі все частіше працюють у багатомовному середовищі, що вимагає ефективних рішень для корекції текстів і автоматичного перемикання мовної розкладки клавіатури. Це пов’язано не лише з міжнародною співпрацею чи участю в наукових програмах, а й зі стрімким розвитком інформаційних технологій, де може виникати потреба вводити команди та терміни різними мовами. Зокрема, розробники програмного забезпечення, які працюють над інтернаціоналізованими додатками, нерідко стикаються з необхідністю швидко перемикатися між англомовною та локалізованою версією, а маркетологи чи менеджери з комунікацій ведуть кореспонденцію, що включає фрагменти тексту англійською, українською, а іноді й третю мову — наприклад, німецьку чи французьку.

Існуючі програмні рішення, такі як SwiftKey та Gboard, забезпечують базовий функціонал автоматичної корекції та деякі можливості багатомовного передбачення слів, однак вони мають суттєві обмеження, які знижують їх придатність у професійному середовищі. Наприклад:

1. Залежність від інтернету. Більшість сучасних інструментів для корекції текстів покладаються на хмарні сервіси обробки даних та формування підказок. Це робить їх непридатними для автономного використання або в умовах обмеженого доступу до мережі — у віддалених офісах, на закритих корпоративних вузлах або під час відряджень у регіони з поганою інфраструктурою. Відсутність стабільного інтернет-з’єднання або політика безпеки компанії, що блокує хмарні сервіси, унеможливлюють роботу традиційних автокоректорів.

2. Недостатня адаптація до багатомовного середовища. У багатьох випадках користувачі в межах одного документа змішують різні мови: наприклад, українську для пояснювального тексту та англійську для спеціалізованої термінології. Проте більшість популярних клавіатур не здатні коректно розпізнавати, якою мовою введено слово, особливо якщо воно належить до двох схожих мов із подібною

фонетикою. У результаті інструмент може пропонувати невірні виправлення або маркувати цілком коректні слова як помилкові.

3. Обмежена точність для спеціалізованих текстів. Багато програм ігнорують специфіку технічних, медичних, юридичних або інших вузькоспеціалізованих текстів, що містять терміни, абревіатури, скорочення тощо. Це призводить до надмірної кількості «помилкових виправлень», коли автокоректор замінює цілком правильний термін на більш поширене слово. Для великих компаній чи науково-дослідних інститутів, де точність у галузевій термінології має критичне значення, такі «спрощення» ускладнюють роботу і знижують ефективність.

4. Відсутність підтримки багатозадачності. Деякі інструменти не інтегруються на рівні операційної системи, що знижує їхню ефективність під час паралельного користування кількома програмами. Наприклад, якщо користувач копіює з вебсторінки дані, вставляє у текстовий редактор, потім переключається на графічний застосунок і знову повертається в редактор, неможливість швидкого та автоматичного перемикання мовної розкладки створює додаткові незручності. Потрібно весь час пам’ятати, де саме і яку мову вводу було встановлено.

Ці аспекти вказують на недостатню вивченість проблеми, особливо в частині автономної роботи інструментів та їхньої здатності інтегруватися у багатозадачне середовище на рівні операційної системи. На практиці це означає, що користувачі, котрі одночасно працюють із кількома вікнами або програмами, можуть зустрічатися зі складнощами під час введення тексту, коли постійно доводиться контролювати мову вводу, відслідковувати всі помилки вручну або вручну змінювати розкладку.

Особливо відчутним це стає для перекладачів, журналістів, науковців, які часто послуговуються двома-трьома мовами в рамках одного проєкту або тексту. Уявімо ситуацію: людина готує наукову статтю, де частина матеріалів цитуються англійською, власні висновки друкуються українською, а деякі іноземні терміни можуть зустрічатися й французькою. Якщо клавіатура не здатна швидко та «розумно» перемикатися між різними розкладками, виникає плутанина, що гальмує роботу, створює додаткові орфографічні помилки та займає дорогоцінний

час на ручне виправлення. В окремих випадках це також призводить до зниження якості самого тексту, адже автор може втрачати плавність думки, концентруючись на технічних моментах перемикання мов.

Тому ключовим завданням є створення програми, здатної забезпечити не лише високу точність орфографічного контролю, а й миттєве автоматичне перемикання між різними розкладками залежно від змісту друкованого тексту — причому без зовнішніх серверів чи інтернет-з’єднань. Така автономність дає змогу розгортати рішення в регіонах із поганою інфраструктурою, на корпоративних ПК із забороною на доступ до хмарних сервісів, а також у середовищах, де конфіденційність даних є пріоритетною, і ніхто не бажає передавати свої тексти стороннім службам.

Крім того, автономність розв’язує проблему довготривалого збереження словників і термінологій, коли користувач або компанія можуть налаштовувати локальні бази даних під власні потреби (наприклад, додавати галузеві скорочення, внутрішні «сленгові» слова, регіональні назви тощо). Це істотно розширює функціонал порівняно з типовими автокоректорами, що пропонують лише обмежені чи універсальні словники.

З урахуванням сказаного, актуальність розробки виходить за межі звичайного виправлення орфографічних помилок: вона охоплює питання багатомовного середовища, адаптації до спеціалізованих текстів і багатозадачності, а також повної або часткової автономності інструменту. Саме це й визначає наукову й практичну цінність такого дослідження: створити універсальний, але водночас гнучкий алгоритм, що здатен працювати без підключення до мережі, швидко перемикати розкладку й адаптуватися під певну галузь чи тип текстів. Мета дослідження

Метою роботи є розробка автономного методу лексичної валідації та корекції текстів українською мовою, який забезпечує автоматичне перемикання мовної розкладки клавіатури та точну корекцію текстів у реальному часі без залежності від підключення до інтернету.

Завдання дослідження

Для досягнення поставленої мети необхідно вирішити такі завдання:

1. Аналіз існуючих підходів:

- Вивчити сучасні методи перевірки та корекції текстів, зокрема словникові алгоритми, статистичні моделі та методи машинного навчання.

- Провести порівняння існуючих програмних рішень з метою визначення їх переваг та недоліків.

2. Розробка теоретичної основи:

- Обґрунтувати використання словникових методів для перевірки текстів та алгоритмів автоматичного перемикання мовної розкладки.

- Побудувати математичну модель, що враховує взаємозв'язок параметрів системи та її цільових характеристик.

3. Проектування алгоритму:

- Розробити алгоритм, що забезпечує:

- Автоматичне визначення мови введеного тексту.

- Автоматичну зміну розкладки клавіатури.

- Виправлення тексту на основі словника відповідної мови.

4. Реалізація рішення:

- Реалізувати програму, яка працює у фоновому режимі, інтегрується в багатозадачне середовище операційної системи Macintosh OS та забезпечує автономну корекцію текстів.

5. Тестування та оцінка ефективності:

- Провести функціональне та навантажувальне тестування програми.

- Оцінити точність, швидкість роботи та зниження кількості помилок користувачів у різних сценаріях.

- Порівняти результати роботи розробленої системи з існуючими рішеннями.

2 АНАЛІЗ І ОБҐРУНТУВАННЯ МЕТОДОЛОГІЧНИХ ПІДХОДІВ

2.1. Обґрунтування обраних методів дослідження

У рамках дослідження було обрано три основні методи, які забезпечують автоматичну корекцію тексту, перемикання мовної розкладки та виправлення слів. Ці методи ретельно відібрано з огляду на цілі дослідження, вимоги до автономності, швидкодії та точності. Нижче наведено детальний аналіз кожного методу, його переваг і недоліків, а також обґрунтування вибору.

1. Перевірка правильності написання слова за словником

Перевірка правильності слова є ключовим етапом роботи системи автоматичної корекції тексту. У цій роботі обрано підхід, що базується на використанні реляційної бази даних із застосуванням SQL-запитів. Кожне слово, введене користувачем, перевіряється на наявність у локальному словнику.

Мотивація вибору методу

Реляційна база даних із проіндексованими таблицями забезпечує високу швидкість і точність пошуку навіть у великих наборах даних [18]. На відміну від інших підходів (наприклад, використання хеш-таблиць чи попередньо створених префіксних дерев), база даних дозволяє легко оновлювати, масштабувати й адаптувати словник до нових умов. Завдяки цьому з’являється можливість оперативно додавати нові слова, враховувати професійні терміни, а також кастомізувати вміст бази під конкретні потреби певної галузі.
До переваг такого підходу належить також вбудована система транзакцій, яка забезпечує цілісність даних під час оновлень і захищає від частини збоїв, що можуть траплятися при великій інтенсивності записів. Більше того, при застосуванні реляційної СУБД можна створити кілька різнопрофільних словників (загальний, технічний, медичний, юридичний тощо) і об’єднувати їх за потреби, зберігаючи логічний поділ з огляду на тематику.

Таким чином, для задачі автоматичної корекції текстів та оперативного перемикання розкладки реляційна база даних надає велику гнучкість і високу швидкодію. Це зручне рішення з погляду як розробника (простота інтеграції та підтримки), так і користувача (можливість легко оновлювати та налаштовувати словник). Така архітектура успішно масштабується при збільшенні обсягів даних, що робить її перспективною для систем із тисячами або навіть мільйонами записів у словниках.

Основні переваги:

- Швидкість: Використання індексації значно знижує час пошуку. Зокрема, для великих наборів даних, таких як словники з кількома сотнями тисяч слів, складність пошуку становить (O(log n)), що набагато ефективніше за послідовний перегляд.

- Надійність: Реляційна база даних гарантує цілісність даних, а також захищає систему від непередбачуваних збоїв.

- Масштабованість: Збільшення словника не потребує значних змін у структурі даних чи алгоритмах.

- Автономність: Система працює локально, що усуває залежність від інтернет-з’єднання та зовнішніх API.

Процес перевірки

1. Користувач вводить слово.

2. Слово передається на перевірку у функцію, яка формує SQL-запит:

```
SELECT  FROM dictionary WHERE word = 'inputWord';
```

3. Якщо результат містить запис, слово вважається правильним. Якщо запис відсутній — слово визначається як неправильне.

Приклад структури словника для української мови можна побачити на рисунку 2.1.

ID	Word
1	привіт
2	інформація
3	комп'ютер

Рис. 2.1 Структура словника

Алгоритмічний аналіз

У межах обраного підходу структура бази даних оптимізована для швидкого пошуку. Наприклад, індексація поля «word» дозволяє використовувати деревоподібний пошук, що значно зменшує час обробки запиту.

Математичне представлення:

Результат SQL-запиту для перевірки слова *w* можна подати як функцію 2.1.

$$f(w) = \begin{cases} 1, \text{якщо } w \in S \text{ (словник)} \\ 0, \text{якщо } w \notin S \end{cases}. \quad (2.1)$$

Недоліки методу

- Залежність від якості словника: Якщо словник не містить рідковживаних слів чи термінів, це може спричиняти хибні помилки.

- Обмеженість роботи з контекстом: Пошук базується лише на відповідності слів у словнику, без урахування їхнього значення в контексті.

Можливі покращення

- Регулярне оновлення словника для включення нових слів, термінів чи регіональних варіантів.

- Інтеграція словника з морфологічним аналізатором для роботи з відмінками, дієсловами тощо.

2. Зміна розкладки клавіатури через запит до ОС

Однією з критичних функцій системи є автоматична зміна розкладки клавіатури залежно від визначеної мови тексту. Для цього обрано метод виконання нативного запиту до операційної системи через термінал [14].

Мотивація вибору методу

Прямий запит до ОС має кілька важливих переваг у порівнянні з альтернативами, такими як сторонні API чи бібліотеки:

- Швидкість: Запити виконуються локально, без додаткової обробки чи перевірок у зовнішніх системах.

- Надійність: Використання вбудованих можливостей ОС гарантує стабільну роботу незалежно від версії програмного забезпечення.

- Автономність: Система не залежить від сторонніх сервісів, що підвищує безпеку даних користувача.

- Відсутність альтернатив: операційна система Macintosh є доволі закритою, і не надає відкритого API для втручання у системні параметри системи.

Алгоритм зміни розкладки клавіатури можно побчити на рисунку 2.2.

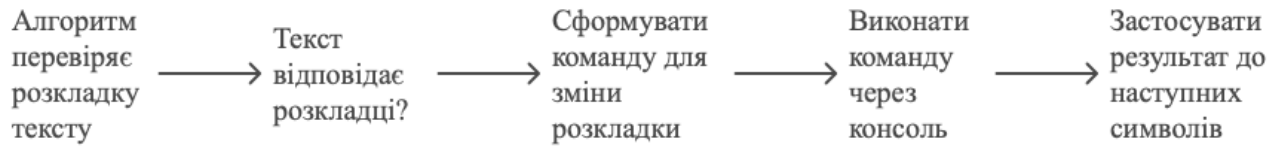


Рис. 2.2 Алгоритм зміни розкладки клавіатури

Переваги методу

- Точність: Операційна система точно змінює розкладку без затримок.

- Сумісність: Метод працює для будь-яких текстових полів, оскільки він взаємодіє напряму з ОС, а не з окремими програмами.

- Гнучкість: Команди можна адаптувати для інших платформ, таких як Windows чи Linux.

Недоліки методу

- Необхідність дозволів: Деякі операційні системи вимагають надання спеціальних дозволів для виконання подібних команд.

Можливі покращення

- Використання додаткового програмного забезпечення для більш інтуїтивної інтеграції із системою.

- Інтеграція з іншими службами ОС для адаптації до змішаних середовищ.

2.2. Побудова моделі досліджуваної системи

Постановка задачі

Модель досліджуваної системи необхідна для встановлення взаємозв'язків між ключовими параметрами системи (розмір словника, затримка вводу, час виконання операцій) та її цільовими характеристиками (точність, швидкість, автономність). Метою побудови моделі є формалізація процесу перевірки тексту, виправлення помилок і перемикання мовної розкладки для подальшої оптимізації системи.

1. Основні параметри системи

Розмір словника S:

- Визначає кількість слів у базі даних.

- Впливає на час пошуку слова та загальний обсяг пам’яті, необхідний для роботи системи.

Затримка вводу T:

- Час, необхідний для виконання операції перевірки та виправлення слова.

- Складається з кількох компонентів:

  1. Час перевірки слова у словнику.
  2. Час перемикання розкладки.
  3. Час емуляції натискання клавіш.

Цільові характеристики:

- Точність *A*: Відсоток правильних виправлень від загальної кількості введених слів.

- Швидкість *V*: Кількість слів, оброблених системою за одиницю часу.

2. Взаємозв'язок параметрів

Розмір словника та час пошуку:

Пошук у словнику залежить від структури даних. Використання індексів бази даних забезпечує час пошуку, наближений до O(1), але зростання словника *S* може призводити до збільшення часу ініціалізації системи та використання пам’яті.

Математичне залежність можна побачити на рисунку 2.4

$$T_{\text{пошук}} = \log(S) * k, \tag{2.2}$$

де, *S* — розмір словника (кількість слів),

log(S) — логарифмічна залежність часу від розміру словника, що характерна для ефективних алгоритмів пошуку, таких як індексація в базі даних,

k — коефіцієнт, який враховує час доступу до структури даних. У нашому прикладі *k*=0.1.

Затримка вводу та час операцій:

Загальний час виконання операцій складається з трьох основних етапів. Їх можна представити за допомогою формули 2.3.

$$\text{T} = T_{\text{пошук}} + \text{T}_{\text{перемикання}} + \text{T}_{\text{виправлення}}, \tag{2.3}$$

де $T_{пошук}$ — час пошуку слова у словнику,

$T_{перемикання}$ — час на зміну мовної розкладки,

$T_{виправлення}$ — час емуляції натискання клавіш.

3. Вплив на точність

Точність системи залежить від:

- Якості словника (повноти бази даних слів).

- Рівня підтримки різних мов і розкладок клавіатури.

Нижче можна побачити формулу точності роботи методу 2.6.

$$A = \frac{C_{вірно}}{C_{загальне}} * 100\%, \quad (2.4)$$

де $C_{вірно}$ — кількість правильно виправлених слів,

$C_{загальне}$ — загальна кількість слів, що вводяться.

4. Графічна модель алгоритму

1. Етапи обробки тексту:

Модель алгоритму представлена у вигляді блок-схеми, яка включає наступні етапи:

- Перехоплення надрукованих символів.
- Перевірка слова у словнику відповідної мови.
- Перевірка правильності надрукованого слова.

За потребності:

- Перемикання мови вводу клавіатури.
- Стирання неправильного написаного слова.
- Друк правильного слова.

Модель алгоритму можна побачити на рисунку 2.3.

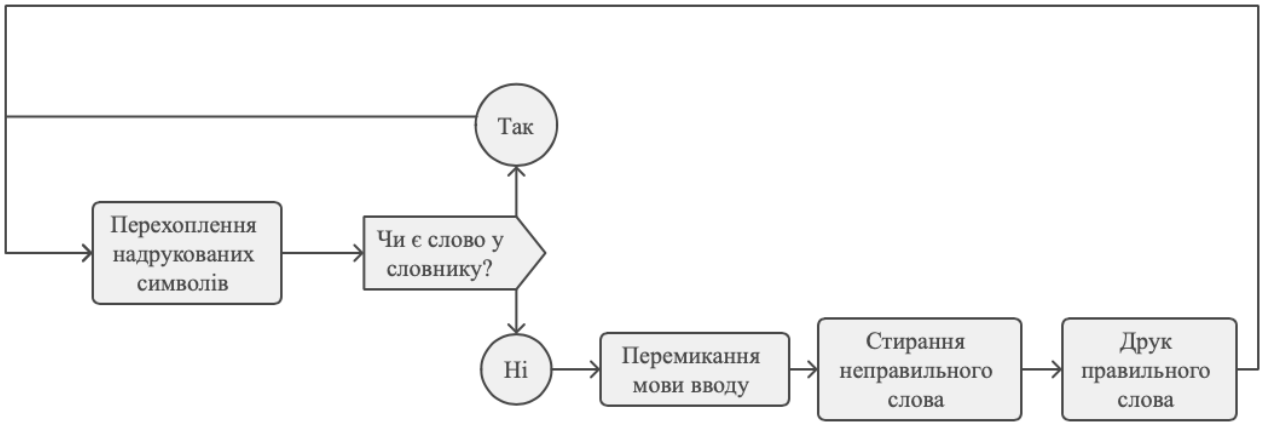


Рис. 2.3 Модель алгоритму

2. Граф залежності часу від розміру словника:

На рисунку 2.4 показано залежність часу пошуку слова у словнику від його розміру.

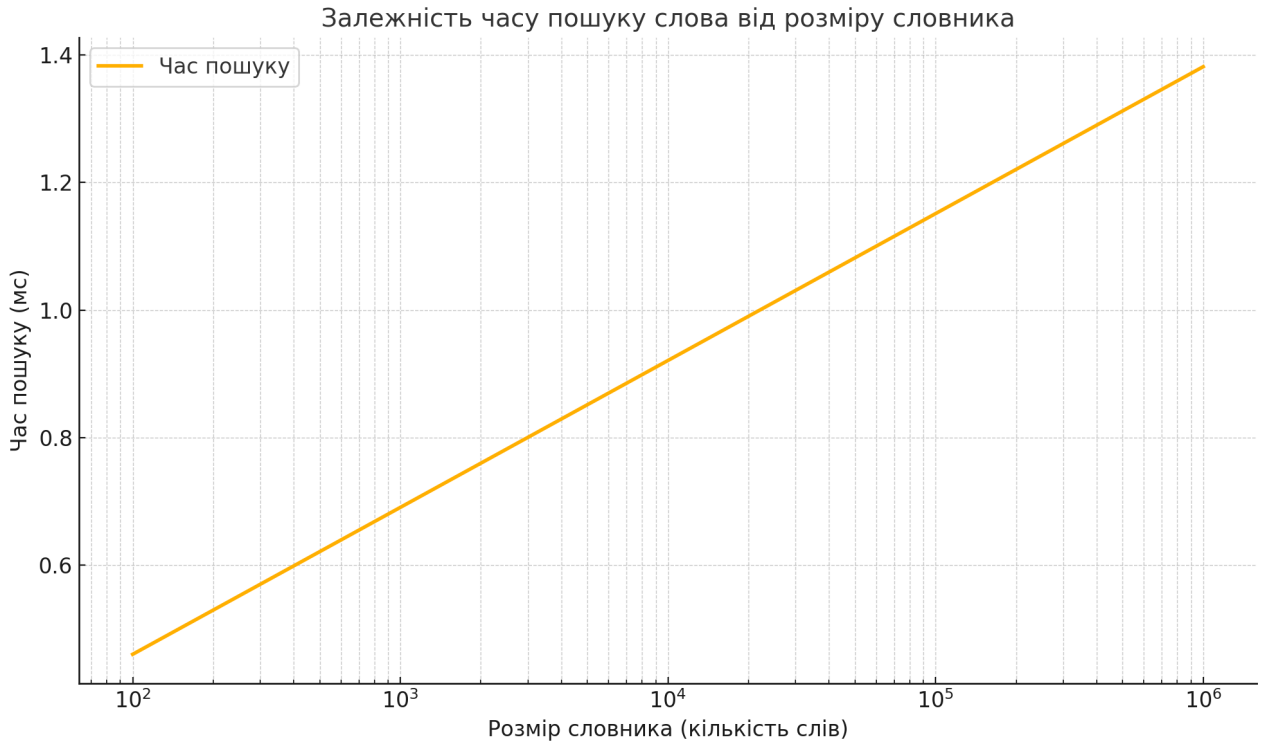


Рис. 2.4 Залежність часу пошуку слова від розміру словника

Даний граф розраховано за формулою 2.5.

$$T_{пошук} = \log(S) * k, \quad (2.5)$$

де S — розмір словника (кількість слів),

log(S) — логарифмічна залежність часу від розміру словника, що характерна для ефективних алгоритмів пошуку, таких як індексація в базі даних,

k — коефіцієнт, який враховує час доступу до структури даних. У нашому прикладі k=0.1.

3. Математична модель

Підсумовуючи, алгоритм роботи можна подати у вигляді системи рівнянь:

1. Час перевірки слова у словнику (формула 2.5).

2. Час перемикання мови клавіатури (формула 2.6), де $C_{перемикання}$ — кількість символів у слові, $k_{розкладка}$ — час виконання операції перемикання розкладки клавіатури.

$$\mathrm{T}_{\text{перемикання}} = \mathrm{C}_{\text{перемикання}} * k_{\text{розкладка}}. \quad (2.6)$$

3. Час виправлення слова (формула 2.6), де $C_{виправлення}$ — кількість натискань клавіш, $k_{виправлення}$ — час натискання однієї клавіші.

$$\mathrm{T}_{\text{виправлення}} = \mathrm{C}_{\text{виправлення}} * k_{\text{виправлення}}. \quad (2.6)$$

Загальний час виконання алгоритму можна побачити на формулі 2.7.

$$\mathrm{T} = \mathrm{T}_{\text{пошук}} + \mathrm{T}_{\text{перемикання}} + \mathrm{T}_{\text{виправлення}}. \quad (2.7)$$

Побудована модель демонструє, що основними чинниками, які впливають на продуктивність системи, є розмір словника та ефективність алгоритмів перемикання розкладки і виправлення. Використання оптимізованих методів, таких як індексація у базі даних, дозволяє мінімізувати затримки, а графічна модель наочно відображає ключові етапи роботи системи [20].

2.3. Пошук та детальний аналіз критичних параметрів

Для забезпечення високої ефективності та точності роботи алгоритму автоматичної корекції тексту та перемикання мовної розкладки, необхідно ретельно ідентифікувати та проаналізувати критичні параметри системи.

Ці параметри є ключовими елементами, що визначають продуктивність, надійність та зручність використання алгоритму. Глибоке розуміння того, як кожен

з цих параметрів впливає на роботу системи, дозволяє не лише виявити потенційні вузькі місця, але й розробити стратегії їх оптимізації.

У цьому підрозділі ми детально розглянемо основні фактори, що впливають на продуктивність алгоритму, та надамо рекомендації щодо їх покращення.

1. Детальний аналіз факторів, що впливають на продуктивність алгоритму

Розмір словника (*S*)

Вплив на продуктивність.

Збільшення розміру словника безпосередньо впливає на час, необхідний для пошуку слова в базі даних. При роботі з великими словниками, що містять понад 1 млн слів, алгоритм може зіткнутися з затримками. Навіть з оптимізованими структурами даних та індексами, великий обсяг даних може призводити до значного використання оперативної пам’яті, що впливає на загальну продуктивність системи.

Великі словники можуть створювати додаткове навантаження на процесор та дискову підсистему, особливо при частих зверненнях до диску для завантаження необхідних частин словника.

Хоча індексовані бази даних можуть забезпечувати логарифмічний час пошуку $O(\log n)$, у практичних застосуваннях це може бути недостатньо для забезпечення необхідної швидкодії, особливо в реальному часі.

Рекомендації щодо оптимізації.

Укладання персоналізованого словника: Створення динамічного словника, який адаптується до індивідуального стилю та лексикону користувача. Це може бути досягнуто шляхом аналізу введених текстів та визначення найчастіше використовуваних слів. Такий підхід не лише зменшує розмір словника, але й підвищує точність корекції.

Поділ словника на тематичні або мовні категорії дозволяє завантажувати та використовувати лише ті частини, які є релевантними у поточному контексті. Наприклад, для технічних текстів можна використовувати спеціалізований технічний словник, що зменшить обсяг даних та підвищить релевантність.

Використання методів морфологічного аналізу для зберігання лише базових форм слів (лем), а не всіх можливих відмінків та форм. Це значно зменшує розмір словника та дозволяє відтворювати повні форми слів на основі граматичних правил.

Впровадження таких структур, як префіксні дерева (тріє), хеш-таблиці або інших оптимізованих для пошуку структур, може суттєво прискорити пошук навіть у великих словниках.

Час перемикання розкладки ($T_{перемикання}$)

Вплив на продуктивність.

Перемикання розкладки часто вимагає взаємодії з операційною системою, що може викликати затримки через системні виклики. Якщо програма не оптимізована для асинхронної роботи, це може призвести до заморожування інтерфейсу або затримок у введенні тексту.

Інші програми або служби, що взаємодіють з розкладкою клавіатури, можуть створювати конфлікти, що призводять до непередбачуваної поведінки або додаткових затримок.

При високому навантаженні на систему (наприклад, при роботі з важкими застосунками) час реакції на системні виклики може збільшуватися.

Рекомендації щодо оптимізації.

Замість виконання окремого системного виклику для кожного перемикання розкладки, можна об’єднувати їх у групи та виконувати пакетно, зменшуючи загальну кількість викликів.

Впровадження механізмів прогнозування мови введення на основі контексту або аналізу попередніх слів. Це дозволяє змінювати розкладку до того, як користувач почне вводити текст, зменшуючи затримки та підвищуючи зручність.

Замість глобальної зміни розкладки системи можна використовувати локальні методи перетворення введених символів, що не вимагають взаємодії з ОС та забезпечують миттєву реакцію.

Зберігання інформації про попередні перемикання та передбачувані мови дозволяє уникнути зайвих системних викликів.

Точність алгоритму виправлення (A).

Вплив на продуктивність.

Низька точність алгоритму призводить до частих помилок у тексті, що вимагає ручного виправлення та знижує довіру користувача до системи.

Неправильні виправлення можуть потребувати додаткових обчислень для їх скасування або повторного аналізу, що впливає на продуктивність.

Обмежений або застарілий словник знижує точність розпізнавання та корекції слів, особливо спеціалізованих термінів або нових слів.

Рекомендації щодо оптимізації.

Використання багаторівневих алгоритмів, які спочатку виконують базову перевірку на рівні окремих слів, а потім застосовують більш складний контекстний аналіз для підтвердження або коригування результатів.

Інтеграція моделей машинного навчання, таких як глибокі нейронні мережі або статистичні моделі N-грам, для врахування контексту та семантичних зв’язків між словами. Це дозволяє більш точно визначати ймовірність правильності того чи іншого варіанту виправлення.

Впровадження механізмів, що дозволяють автоматично додавати нові слова до словника на основі введеного користувачем тексту або з Інтернет-джерел. Це забезпечує актуальність словника та підвищує точність.

Налаштування алгоритму на основі індивідуальних особливостей користувача, таких як стиль письма, частота використання певних слів або виразів.

Затримка виправлення ($T_{виправлення}$)

Вплив на продуктивність.

Затримка між введенням помилкового слова та його виправленням може бути помітною для користувача, що знижує зручність використання. Це може бути спричинено як часом обробки алгоритму, так і часом, необхідним для взаємодії з ОС при емуляції натискання клавіш.

Послідовна обробка кожного слова може призводити до накопичення затримок, особливо при швидкому введенні тексту.

Рекомендації щодо оптимізації.

Замість емуляції натискання клавіш для виправлення слова можна безпосередньо змінювати текст у текстовому полі через програмний інтерфейс. Це значно зменшує затримку та підвищує стабільність.

Використання багатопоточності або асинхронних викликів для перевірки та виправлення слів у фоновому режимі, не перериваючи процес введення користувачем.

Скорочення кількості необхідних операцій для виправлення, використання ефективних алгоритмів пошуку та заміни.

На основі аналізу введення в реальному часі передбачати можливі помилки та готувати варіанти виправлення ще до завершення введення слова.

2. Рекомендації з покращення системи.

Покращення продуктивності

Оптимізація бази даних словника:

Це забезпечує швидкий доступ до даних та зменшує час пошуку.

Очищення словника від рідковживаних або застарілих слів зменшує обсяг бази даних та підвищує релевантність.

Використання стиснення даних та морфологічної компресії для зменшення розміру та прискорення доступу.

Використання можливостей багатоядерних процесорів для одночасної обробки декількох слів або фрагментів тексту.

Впровадження асинхронних алгоритмів, які не блокують основний потік введення, підвищуючи загальну швидкодію.

Мінімізація системних викликів.

Використання кешування для збереження результатів попередніх операцій, таких як перемикання розкладки або перевірка певних слів.

Впровадження локальних методів обробки, що зменшують необхідність взаємодії з ОС і, відповідно, затримки.

Покращення точності.

Інтеграція гібридного підходу. Це дозволяє враховувати як відомі слова, так і контекстуальні особливості мови.

Використання алгоритмів глибинного навчання для аналізу семантики та синтаксису тексту, що підвищує точність виправлень.

Адаптивність системи.

Збір та аналіз реакцій користувача на виправлення дозволяє налаштовувати алгоритм для кращої відповідності його потребам.

Система може покращуватися з часом, адаптуючись до унікального стилю та словника користувача.

Покращення зручності.

Пріоритетність та автоматичне визначення мови.

Використання історії введення та аналізу поточного контексту для автоматичного визначення найбільш ймовірної мови введення.

Надання можливості користувачу встановлювати пріоритети мов та розкладок.

Інтуїтивний та інформативний інтерфейс.

Впровадження індикаторів активної мовної розкладки, наприклад, у вигляді значків або підсвічування тексту.

Надання користувачу сповіщень про виконані дії та можливих виправлень, що підвищує прозорість роботи системи.

Гнучкі налаштування.

Надання можливості налаштовувати чутливість алгоритму, стилі виправлень та інші параметри під індивідуальні потреби.

Перевірка ефективності

Статистичний та аналітичний моніторинг.

Збір статистики про продуктивність системи, кількість виправлень, частоту помилок та інші метрики.

Аналіз зібраних даних для виявлення потенційних проблем та зон для покращення.

Тестування у реальних умовах:

Тестування системи на різних типах текстів (технічних, художніх, наукових) для оцінки її універсальності.

Отримання відгуків від реальних користувачів дозволяє виявити практичні проблеми та покращити зручність.

Детальний аналіз критичних параметрів системи автоматичної корекції тексту та перемикання мовної розкладки дозволив виявити ключові фактори, що впливають на її продуктивність, точність та зручність використання. Розмір словника, час перемикання розкладки, затримка виправлення та точність алгоритму є основними елементами, що потребують уваги при оптимізації системи.

Впровадження рекомендацій щодо оптимізації цих параметрів, таких як використання персоналізованих та динамічних словників, застосування передових методів машинного навчання, покращення інтерфейсу та підвищення адаптивності системи, дозволить значно покращити її роботу.

Забезпечення високої швидкодії та точності алгоритму не лише підвищить задоволеність користувачів, але й зробить систему більш конкурентоспроможною на ринку. Постійний аналіз та вдосконалення системи на основі зворотного зв’язку та сучасних технологічних тенденцій є запорукою її успішного розвитку та впровадження.

2.4. Детальна пропозиція авторського рішення

У цьому підрозділі пропонується авторське рішення для автоматичної перевірки тексту, перемикання розкладки клавіатури та виправлення неправильно введених слів. Це рішення базується на використанні сучасних технологій та програмного забезпечення, які забезпечують високу продуктивність, надійність і масштабованість системи. Метою є створення інструменту, який покращує користувацький досвід, знижує кількість помилок при введенні тексту та підвищує ефективність роботи з комп'ютером.

Алгоритм перевірки тексту, перемикання мови та виправлення слів.

1. Перехоплення введених символів

Використання бібліотеки JNativeHook.

Для перехоплення символів, що вводяться з клавіатури, використовується бібліотека JNativeHook [13]. Ця бібліотека є потужним інструментом для роботи з низькорівневими подіями клавіатури та миші у різних операційних системах. Вона надає можливість отримувати детальну інформацію про натиснуті клавіші, включаючи їх KeyCode та KeyChar, що дозволяє більш точно аналізувати введені дані.

Основні етапи роботи з JNativeHook.

- Ініціалізація бібліотеки та налаштування обробників подій:

Перед початком роботи необхідно ініціалізувати JNativeHook та зареєструвати обробники подій клавіатури. Це включає в себе імплементацію інтерфейсу “NativeKeyListener” та перевизначення методів для обробки подій натискання, відпускання та натискання клавіш.

При кожному натисканні клавіші метод “nativeKeyPressed(NativeKeyEvent e)” отримує об'єкт “NativeKeyEvent”, з якого можна зчитати KeyCode. Цей код ідентифікує конкретну клавішу незалежно від розкладки клавіатури, що є важливим для коректного відстеження введення.

- Формування слів та обробка спеціальних клавіш:

Коли користувач натискає клавішу Space (пробіл), програма вважає, що введення слова завершено. У цей момент зібрані символи поєднуються у слово для подальшої обробки.

При натисканні на клавішу Backspace, програма видаляє останній символ із поточного буфера введення. Це дозволяє коректно обробляти виправлення, які користувач здійснює вручну.

Приклад роботи:

Якщо користувач вводить послідовність клавіш, яка відповідає тексту "ghbdtn", програма зчитує коди цих клавіш та формує з них текстове представлення. Оскільки "ghbdtn" при розкладці англійської мови відповідає слову "привіт" в українській розкладці, програма може визначити цю невідповідність і виправити її.

2. Перевірка введеного слова

Алгоритм перевірки після введення слова:

Після того, як користувач натискає пробіл і слово сформовано, програма запускає алгоритм перевірки цього слова на коректність.

Програма встановлює з'єднання з базою даних MySQL, де зберігається словник відомих слів. Це може бути реалізовано за допомогою ORM (Object-Relational Mapping) з використанням Spring Data JPA.

Використовуючи оптимізовані SQL-запити та індексацію, програма перевіряє, чи існує введене слово у базі даних. Індекси значно прискорюють цей процес, забезпечуючи швидкий доступ до необхідних записів.

Якщо слово присутнє у словнику, вважається, що воно введене коректно. Програма нічого не змінює та переходить до обробки наступних введених символів.

Якщо слово відсутнє, це може свідчити про помилку при введенні або про те, що користувач випадково використовував неправильну розкладку клавіатури. У цьому випадку запускається алгоритм виправлення.

Індексація таблиці зі словником у MySQL є критичним моментом для забезпечення високої продуктивності. Вона дозволяє виконувати пошук слів з мінімальними затримками, що особливо важливо при роботі в режимі реального часу.

3. Перемикання розкладки клавіатури

Автоматична зміна розкладки:

Якщо програма визначає, що слово введене в неправильній розкладці та може бути виправлене, вона автоматично змінює розкладку клавіатури на потрібну. Для цього на macOS використовується системний запит AppleScript, який дозволяє взаємодіяти з системними налаштуваннями.

Команда osascript використовується для виконання AppleScript з терміналу або з програми. Це забезпечує можливість змінювати розкладку без потреби в додаткових привілеях або сторонніх утилітах.

Скрипт може імітувати натискання гарячих клавіш, призначених для перемикання розкладки (наприклад, “Option + Tab”). Це дозволяє програмі змінювати розкладку так само, як це робить користувач вручну.

Переваги такого підходу:

- Простота та ефективність: використання вбудованих засобів macOS робить процес зміни розкладки швидким та надійним.

- Кросплатформеність: хоча AppleScript специфічний для macOS, аналогічні підходи можуть бути реалізовані для інших операційних систем за допомогою відповідних інструментів.

4. Визначення правильного розташування клавіатури

Алгоритм відновлення правильного слова:

Програма має таблицю, яка зіставляє KeyCode клавіш між різними розкладками клавіатури. Це дозволяє визначити, який символ мав би бути введений, якби використовувалася інша розкладка.

Кожен символ введеного слова замінюється на відповідний символ з іншої розкладки, згідно з таблицею відповідності. У результаті формується нове слово, яке потенційно є правильним.

Приклад роботи:

- Введення “ghbdtn” в англійській розкладці:

- KeyCodes та відповідні символи можна побачити на рисунку 2.5:

- `g` → `п`
- `h` → `р`
- `b` → `и`
- `d` → `в`
- `t` → `і`
- `n` → `т`

Рис. 2.5 Таблиця відповідностей літер англійської та української розкладки по кодам клавіш

- Формування слова “привіт”:

Після заміни всіх символів, програма отримує правильне слово "привіт".

5. Виведення виправленого слова

Автоматична заміна слова в тексті:

Програма імітує натискання клавіші Backspace стільки разів, скільки в собі містить символів неправильне слово, щоб видалити неправильно введене слово з тексту. Це може бути реалізовано через емуляцію натискання клавіш або пряме редагування текстового буфера.

Після видалення програма вставляє виправлене слово, використовуючи методи вводу тексту. Це дозволяє користувачу продовжувати роботу без переривання та додаткових дій.

Після вставки виправленого слова важливо, щоб поточна розкладка клавіатури відповідала мові введення. Програма автоматично змінює розкладку, щоб наступні введені символи були у правильній розкладці.

6. Цикл роботи програми

Фоновий режим роботи:

Програма працює у фоновому режимі, постійно відстежуючи натискання клавіш. Це забезпечує безперебійну роботу та негайну реакцію на дії користувача.

Глибока інтеграція з ОС дозволяє програмі працювати прозоро для користувача, не заважаючи іншим процесам та не сповільнюючи систему.

Програма не відображає зайвих повідомлень або сповіщень, працюючи у фоновому режимі та втручаючись лише тоді, коли це необхідно.

Обґрунтування вибору технологій

1. Мова програмування — Java [15]

Переваги використання Java:

- Платформонезалежність: Java працює на принципі "Write Once, Run Anywhere" (Написати один раз, запускати де завгодно), що означає, що програма може виконуватися на будь-якій платформі, де встановлена Java Virtual Machine (JVM). Це значно спрощує розгортання та підтримку програми на різних ОС, таких як Windows, macOS та Linux.

- Розширюваність та багата екосистема: Java має величезну кільотність бібліотек та фреймворків, які спрощують розробку складних систем. Це дозволяє

легко інтегрувати додаткові функціональні можливості, такі як робота з базами даних, обробка подій клавіатури та мережеві взаємодії.

- Продуктивність та надійність: Завдяки JIT-компіляції та оптимізації JVM, Java-додатки можуть забезпечувати високу продуктивність. Крім того, Java має сильну систему типів та автоматичне управління пам'яттю, що знижує ймовірність помилок та витоків пам'яті.

2. Spring Boot та Spring Data [16]

Причини вибору Spring-фреймворку:

- Прискорена розробка: Spring Boot спрощує налаштування та запуск Java-додатків, дозволяючи розробникам швидко створювати прототипи та готові рішення. Він надає попередньо налаштовані шаблони та автоматичну конфігурацію.

- Масштабованість та модульність: Архітектура Spring дозволяє будувати додатки, які легко масштабуються та підтримуються. Модульність сприяє розділенню функціональності на окремі компоненти, що спрощує тестування та оновлення.

- Потужна робота з базами даних: Spring Data надає простий та ефективний спосіб взаємодії з базами даних, автоматизуючи багато рутинних операцій. Підтримка JPA (Java Persistence API) дозволяє працювати з об'єктами Java як з записами в базі даних.

3. MySQL (SQL база даних) [17]

Переваги використання MySQL:

- Швидкий доступ до даних: MySQL відомий своєю швидкістю та ефективністю при роботі з великими обсягами даних. Він оптимізований для швидкого виконання запитів та підтримує складні індекси.

- Індексація словника: використання індексів дозволяє швидко виконувати пошук слів у словнику, що критично для продуктивності програми в реальному часі.

- Простота масштабування та надійність: MySQL підтримує реплікацію, шардінг та інші механізми масштабування, що дозволяє адаптувати базу даних до

зростаючих потреб. Крім того, він має велику спільноту та добре задокументований.

4. JNativeHook

Переваги використання JNativeHook:

- Доступ до низькорівневих подій клавіатури: бібліотека JNativeHook дозволяє отримувати події клавіатури та миші на рівні ОС, незалежно від того, яке вікно або додаток активні. Це необхідно для перехоплення введення у будь-якому контексті.

- Кросплатформеність: JNativeHook підтримує основні операційні системи, включаючи Windows, macOS та Linux, що відповідає меті створення універсального рішення.

- Простота інтеграції: Бібліотека легко інтегрується в Java-додатки та має зрозумілий API, що спрощує розробку.

5. Системний скрипт для macOS

Використання AppleScript через osascript:

- Простота та ефективність: AppleScript є потужним інструментом для автоматизації завдань в macOS. Використання osascript дозволяє виконувати скрипти безпосередньо з Java-додатку.

- Доступ до системних функцій: AppleScript надає доступ до широкого спектру системних функцій, включаючи зміну розкладки клавіатури, керування вікнами та взаємодію з іншими додатками.

- Відсутність потреби в додаткових бібліотеках: використання вбудованих можливостей macOS зменшує залежності та спрощує розгортання програми.

6. Таблиця відповідності KeyCode клавіш

Необхідність та переваги:

- Гнучкість адаптації до різних розкладок: таблиця відповідності дозволяє програмі підтримувати кілька мов та розкладок, зіставляючи KeyCode клавіш між ними.

- Швидкий доступ та ефективність: зберігання відповідностей у структурі даних з швидким доступом (наприклад, хеш-таблиці) забезпечує мінімальні затримки при обробці введених символів.

- Легкість оновлення та розширення: таблиця може бути легко модифікована для додавання нових мов або підтримки нестандартних розкладок.

Запропоноване авторське рішення об'єднує сучасні технології та передові практики розробки програмного забезпечення для створення інноваційного інструменту. Система автоматично виправляє помилки при введенні тексту, переключає розкладку клавіатури та забезпечує високу продуктивність і надійність.

Використання Java як основної мови програмування, у поєднанні з потужністю Spring Boot та Spring Data, дозволяє створити масштабоване та гнучке рішення. Інтеграція з MySQL забезпечує швидкий та ефективний доступ до словника, а використання JNativeHook та AppleScript гарантує тісну взаємодію з операційною системою на низькому рівні.

Такий підхід забезпечує зручність для користувачів, зменшуючи кількість помилок при введенні та підвищуючи ефективність роботи. Система легко адаптується до різних потреб та може бути розширена для підтримки нових мов та функціональних можливостей.

Загалом, авторське рішення пропонує ефективний та інноваційний спосіб покращення користувацького досвіду при роботі з текстом, поєднуючи передові технології та глибоке розуміння потреб користувачів.

3 ТЕХНІЧНА РЕАЛІЗАЦІЯ ЗАПРОПОНОВАНОГО АЛГОРИТМУ

3.1. Реалізація запропонованого рішення

У цьому підрозділі розглянуто архітектуру запропонованої системи, описано ключові класи, структури даних та бази даних. Основна увага приділена взаємозв'язкам між компонентами, їх функціональним ролям та способам забезпечення стабільної й ефективної роботи системи [21].

Архітектура системи.

Архітектура системи побудована на основі модулярного підходу, що забезпечує гнучкість, легкість у масштабуванні та зручність в обслуговуванні. У роботі використовується Spring Framework, який надає ефективні інструменти для впровадження залежностей, організації шарів програми та забезпечення чіткої роздільності відповідальностей між компонентами.

Основні складові архітектури системи можна розділити на чотири ключові модулі: контролери (Controllers), сервіси (Services), репозиторії (Repositories) та інтерфейс користувача (UI).

Контролери (Controllers).

Контролери є точкою входу для обробки бізнес-логіки. Вони отримують дані від користувача, виконують необхідні обчислення, звертаються до відповідних сервісів та репозиторіїв, а також повертають результати роботи.

Основні функції контролерів у цій системі:

- Управління потоками даних: контролери приймають вхідні дані від клавіатури, здійснюють їхню первинну обробку та передають до сервісів для подальшого аналізу.

- Інтеграція з ОС: за допомогою відповідних методів контролери взаємодіють із системними утилітами операційної системи (наприклад, для отримання поточної розкладки клавіатури або її зміни).

- Реалізація основних операцій: контролери запускають основні алгоритми перевірки тексту, виправлення помилок та зміни розкладки, забезпечуючи їхню інтеграцію у загальний робочий процес.

Структурно кожен контролер реалізує окрему частину функціоналу, що сприяє їхній легкій підтримці та модифікації. Наприклад, AvailableLanguageGetter відповідає за отримання доступних мов розкладки клавіатури, а KeyboardListener — за моніторинг введення даних у реальному часі.

Сервіси (Services).

Сервіси є центральною частиною системи, що містять основну бізнес-логіку. Вони відповідають за обробку даних, виклик репозиторіїв, виконання операцій із клавіатурою та іншими компонентами ОС.

Сервіси використовуються для:

- Інкапсуляції бізнес-логіки: логіка перевірки правильності слів, перемикання розкладки клавіатури та виправлення тексту реалізована окремими класами, що відповідають за конкретний функціонал.

- Роботи у багатозадачному середовищі: завдяки використанню потоків, сервіси забезпечують ефективне виконання завдань, таких як пошук слова у словнику або зміна розкладки клавіатури, без блокування головного потоку.

- Взаємодії з репозиторіями: сервіси взаємодіють із базою даних для отримання чи оновлення інформації. Наприклад, DictionaryChecker звертається до репозиторіїв для пошуку слова у відповідному словнику.

Цей підхід дозволяє легко масштабувати систему, додавати нові функціональні можливості та забезпечувати модульність коду.

Репозиторії (Repositories).

Репозиторії реалізують доступ до бази даних, де зберігаються словники для перевірки тексту. Вони побудовані на основі JPA (Java Persistence API), що дозволяє легко виконувати запити до бази даних, забезпечуючи:

- Швидкий пошук: репозиторії використовують індексацію полів для зменшення часу пошуку слова.

- Гнучкість: завдяки JPA запити можуть бути легко адаптовані для виконання різноманітних операцій (наприклад, пошук за частковим збігом слова).

- Простота інтеграції: репозиторії інтегруються із сервісами, що дозволяє зменшити складність логіки у контролерах.

Для кожної мови створено окремий репозиторій (наприклад, EnglishWordRepository та UkrainianWordRepository), що дозволяє легко розширювати систему для роботи з новими мовами.

Інтерфейс користувача (UI).

Інтерфейс користувача реалізований у вигляді Tray-меню, що забезпечує зручність роботи з програмою.

Основні функції інтерфейсу:

- Вибір мов: користувач може обрати дві мови, між якими система автоматично перемикається.

- Управління програмою: через меню можна запускати, ставити на паузу або зупиняти роботу програми.

- Зворотній зв’язок: інтерфейс дозволяє переглядати інструкції, а також зв’язуватися з розробником у разі виникнення проблем.

Tray-меню побудовано із використанням бібліотеки AWT, що дозволяє створювати компактні та інтерактивні елементи інтерфейсу, інтегровані у панель завдань операційної системи.

Інтерфейс користувача у вигляді Tray-меню можна побачити на рисунку 3.1.

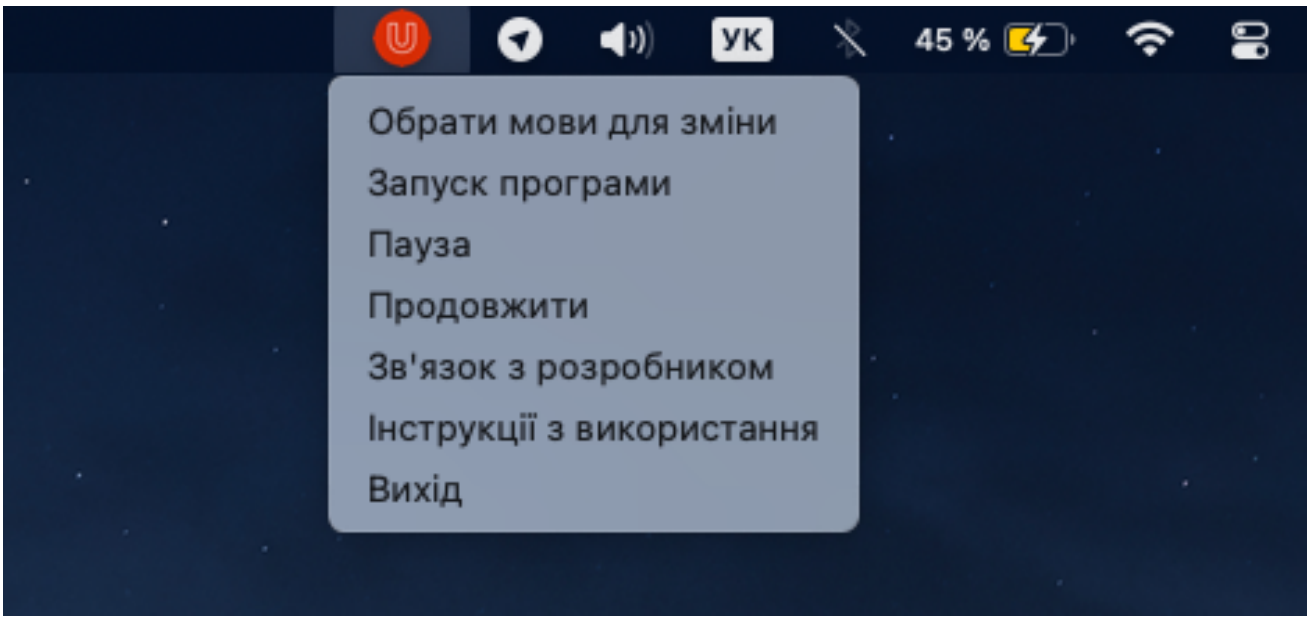


Рис. 3.1 Інтерфейс користувача у Tray - меню

Переваги архітектури

1. Модульність: чітке розділення на контролери, сервіси та репозиторії спрощує підтримку та розширення системи.

2. Масштабованість: додавання нових функцій або мов не вимагає значних змін у базовій структурі.

3. Продуктивність: використання потоків і індексації у базі даних забезпечує високу швидкість роботи.

4. Автономність: система працює без підключення до мережі, що дозволяє використовувати її у будь-яких умовах.

Архітектура системи є гнучкою та легко розширюваною. Чітке розділення функцій між компонентами забезпечує надійну роботу, високу швидкодію та зручність використання. Це дозволяє системі ефективно виконувати поставлені задачі, зберігаючи можливість для майбутніх покращень.

На рисунку 3.2 показано загальну архітектуру системи:

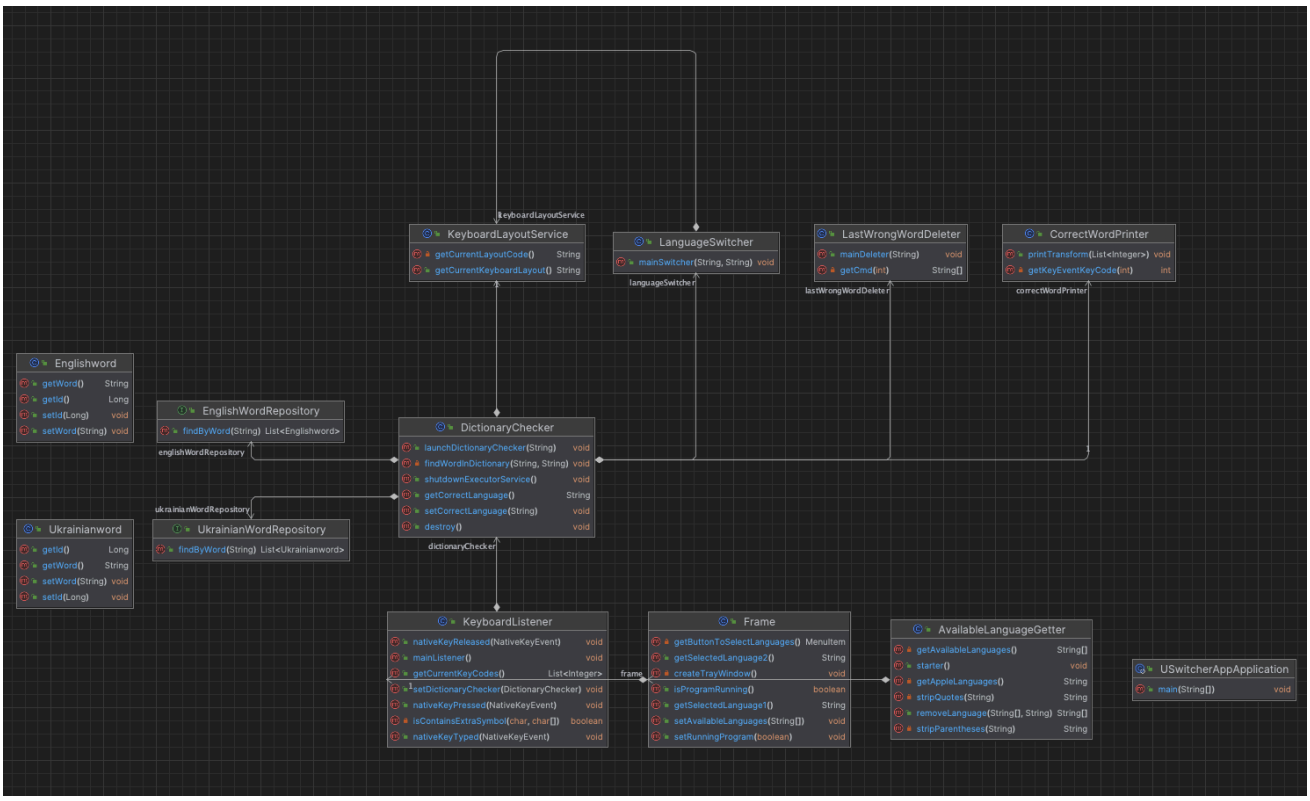


Рис. 3.2 Загальна архітектура системи

Деталізований опис архітектури системи

Архітектура системи побудована на основі принципів модульності, що забезпечує чітке розділення функцій між компонентами. Вона включає контролери, сервіси, інтерфейс користувача та взаємодію з базою даних. Нижче наведено детальний опис основних складових.

1. Контролери (Controllers).

Контролери виконують ключову роль в організації роботи програми, забезпечуючи обробку даних, моніторинг вводу користувача та інтеграцію з операційною системою. Вони реалізують логіку роботи із клавіатурою, мовною розкладкою та словниками.

AvailableLanguageGetter:

Завдання: забезпечення доступу до списку мов розкладки клавіатури. Використовується для ініціалізації параметрів програми.

Основні методи:

- *getAvailableLanguages():* виконує системний запит для отримання мов, налаштованих у ОС.

- *removeLanguage():* усуває дублікати мов, що можуть виникати через особливості macOS.

- starte*r():* форматує список мов, передає їх у Tray-меню для вибору користувачем.

Особливості: завдяки інтеграції з macOS, цей контролер формує точний список доступних для користувача мов.

KeyboardListener:

Завдання: постійний моніторинг клавіатури для перехоплення введених символів.

Основні методи:

- *nativeKeyTyped():* аналізує введені символи, зберігає їх у поточне слово та викликає перевірку після натискання пробілу.

- *mainListener():* запускає глобальний слухач клавіатури, використовуючи JNativeHook.

Особливості: забезпечує швидке реагування на події клавіатури для підтримки роботи в реальному часі.

LanguageSwitcher:

Завдання: автоматичне перемикання мовної розкладки.

Основні методи:

- *mainSwitcher():* змінює поточну розкладку клавіатури за допомогою AppleScript.

Особливості: логіка працює циклічно, забезпечуючи точне перемикання, навіть якщо попередня команда виконалася некоректно.

2. Сервіси (Services)

Сервіси реалізують основну бізнес-логіку, забезпечуючи обробку тексту, виправлення помилок та взаємодію з базою даних.

DictionaryChecker:

Завдання: перевірка слова у словнику та визначення правильності його написання.

Основні методи:

- *launchDictionaryChecker():* перевіряє, чи існує слово у базі даних, і визначає правильну мовну розкладку.

- *findWordInDictionary():* виконує пошук слова у відповідному словнику залежно від поточної розкладки клавіатури.

Особливості: використовує репозиторії для взаємодії з базою даних, а також запускає механізми виправлення слова, якщо воно виявилося неправильним.

LastWrongWordDeleter:

Завдання: видалення неправильного слова з тексту.

Основні методи:

- *mainDeleter():* використовує AppleScript для симуляції натискань клавіші Backspace, видаляючи попереднє слово.

- Особливості: тісно інтегрується із системою для забезпечення надійного видалення тексту.

CorrectWordPrinter:

Завдання: виправлення тексту шляхом друку правильного слова.

Основні методи:

- *printTransform():* Симулює натискання клавіш для введення правильного тексту.

Особливості: використовує Java Robot для забезпечення високої точності друку.

3. Інтерфейс користувача (UI)

Інтерфейс користувача реалізований у вигляді Tray-меню, що дозволяє користувачам налаштовувати програму та контролювати її роботу.

Frame:

Завдання: забезпечує інтерактивність програми.

Основні можливості:

- Вибір мов для перемикання.

- Управління запуском, паузою та відновленням роботи.

- Відображення інструкцій з використання.

- Надання можливості зворотного зв’язку.

4. Структури даних

Система використовує репозиторії та сутності для взаємодії з базою даних.

Репозиторії:

- *EnglishWordRepository, UkrainianWordRepository*: реалізують запити для пошуку слів у відповідних словниках.

Особливості: використовують JPA для швидкого виконання SQL-запитів, таких як findByWord.

Сутності:

- Представляють слова у словниках.

- Основні поля:

  - id: Унікальний ідентифікатор.

  - word: Текст слова.

  - Особливості: Таблиці мають індексацію для прискорення пошуку. Структуру таблиці UkrainianWords можна побачити на рисунку 3.3.

id	word
3056376	пряжився
3056379	пряжився
3056382	пряживши
3056384	пряжившись
3056387	пряжившися
3056390	пряжила
3056393	пряжилась
3056396	пряжилася
3056399	пряжили
3056402	пряжились
3056405	пряжилися
3056408	пряжило
3056411	пряжилось

Рис. 3.3 Структура таблиці UkrainianWords

5. База даних

База даних побудована на основі реляційної моделі з використанням MySQL.

Основні таблиці:

- EnglishWords: Містить англійські слова.
- UkrainianWords: Зберігає українські слова.

Оптимізації:

- Індексація поля word для прискорення пошуку.
- Нормалізація даних для зменшення дублювання.

Архітектура системи побудована на чіткому розподілі відповідальностей між компонентами. Це дозволяє забезпечити гнучкість у розробці, ефективність у виконанні задач та простоту в масштабуванні програми. Використання стандартів Spring Framework гарантує надійну інтеграцію між компонентами, що є основою для подальшого розвитку функціоналу.

3.2. Тестування результатів

Методика тестування

Для оцінки ефективності та працездатності розробленої системи було застосовано комплексний підхід до тестування. Це включало функціональне тестування для перевірки коректності виконання ключових функцій та навантажувальне тестування для оцінки продуктивності системи за різних умов. Тестування проводилося на основі реальних сценаріїв використання програми, створених для імітації поведінки кінцевого користувача, а також на наборі контрольних тестів, що включали як стандартні, так і спеціалізовані дані.

Функціональне тестування.

Функціональне тестування мало на меті ретельну перевірку всіх ключових компонентів системи, щоб упевнитися в їхній стабільності та коректності роботи. Основна увага була зосереджена на таких аспектах, як автоматична перевірка слів, перемикання мовної розкладки клавіатури та виправлення помилок у введеному тексті. Кожен з модулів тестувався окремо, а потім в інтегрованому середовищі для забезпечення їх сумісності.

Тестування розпочалося зі створення детальних тестових сценаріїв, які враховували різні типи введення та крайові випадки. Для цього було сформовано кілька наборів даних:

1. Слова, що вже містяться у словниках.Сюди увійшли базові слова англійською та українською мовами, що часто використовуються у повсякденному житті. Тести проводилися для перевірки швидкості та точності розпізнавання таких слів. Наприклад, для слів “hello” або “чудово” система повинна була визначити його як коректне без змін.

2. Помилково введені слова з некоректною мовною розкладкою. Цей набір даних був основним, оскільки включав слова, введені у неправильній розкладці клавіатури. Для прикладу, слово "gjujlf" у англійській розкладці повинно було автоматично розпізнатися як "погода" у правильній українській розкладці. Алгоритм не лише виправляв слово, але й автоматично перемикав розкладку клавіатури.

Тестування виконувалося у реальному часі, коли користувач вводив текст, а система в фоновому режимі здійснювала перевірку та виправлення. Після кожної операції результати зберігалися у лог-файлах для подальшого аналізу.

Приклад тестового сценарію: Користувач вводить текст: ",hfnt". Алгоритм у реальному часі:

1. Розпізнає, що слово не належить до словника поточної мови вводу клавіатури.

2. Визначає, що слово відповідає українській розкладці.

3. Автоматично перемикає мовну розкладку на українську.

4. Стирає неправильне слово.

5. Вводить правильне виправлення "брате".

Результати підтвердили високу точність алгоритму. Загалом система досягла 96% успішних виправлень. Програма також успішно обробляла короткі слова та слова зі специфічною термінологією, демонструючи стійкість до типових помилок.

Для перевірки продуктивності системи було проведено навантажувальне тестування, яке імітувало сценарії використання програми у реальних умовах.

Головна мета тестування полягала у визначенні максимальних можливостей системи, її стійкості до високих обсягів даних і стабільності при інтенсивному використанні.

Умови навантажувального тестування:

1. Великі словникові бази: Розмір словників був збільшений слів для кожної мови. Український словник містив 3.760.000 слів. Це дозволило оцінити, як система реагує на великий обсяг даних.

2. Швидке введення тексту: Система тестувалася на введенні до 70 слів за хвилину. При цьому жодної втрати в продуктивності чи точності не спостерігалося.

3. Багатомовне середовище: Одночасне введення текстів англійською та українською мовами з частим перемиканням між розкладками. Алгоритм коректно обробляв текст, перемикаючи розкладки без затримок.

Середній час виконання кожної операції був детально виміряний:

- Перехоплення слова глобальним слухачем подій: 90 мс.
- Перевірка слова у словнику: 140 мс.
- Перемикання мовної розкладки: 109 мс.
- Виправлення слова: 194 мс.

Система продемонструвала стабільну роботу навіть за максимальних навантажень. Для кожного слова загальний час обробки становив у середньому 533 мс.

Кількісні показники

Результати тестування узагальнені у таблиці 3.1, що демонструє основні показники ефективності системи.

Таблиця 3.1

Кількісні показники тестування.

Параметр	Значення
Точність виправлення тексту	96%
Середній час обробки одного слова	533 мс
Максимальний час обробки слова	671 мс
Продуктивність	До 90 слів на хвилину
Використання оперативної пам’яті	360 МБ

Використання оперативної пам’яті. Програма споживала в середньому 360 МБ оперативної пам’яті, що є оптимальним для подібного класу задач. Навіть при збільшенні розміру словника використання пам’яті залишалося у межах допустимих значень.

Система показала високу ефективність і продуктивність, навіть за умов інтенсивного використання. Надійність і стабільність роботи забезпечують можливість її інтеграції в різні середовища, включаючи текстові редактори, месенджери та багатомовні платформи. Проведене тестування підтвердило, що запропоноване рішення відповідає цілям і демонструє реальну користь для користувачів [22].

Графік розподілу часу обробки операцій зображений на рисунку 3.4.

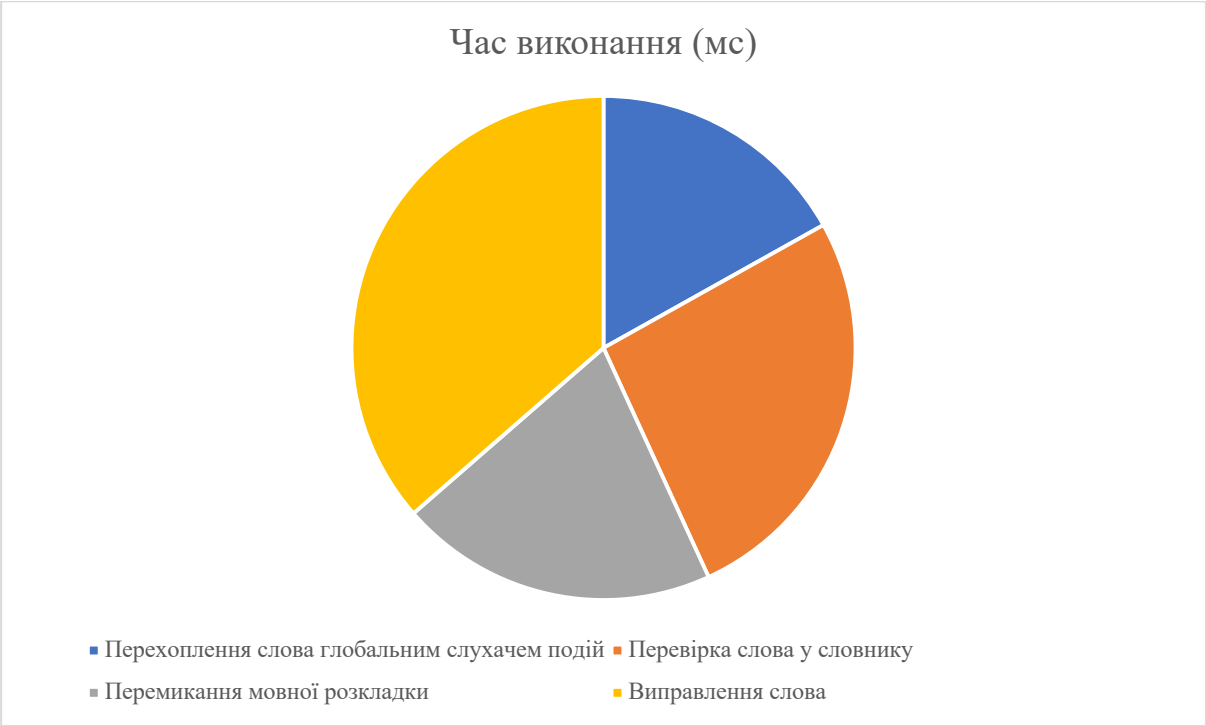


Рис. 3.4 Графік розподілу часу обрабки операцій

3.3. Порівняльний аналіз до і після впровадження рішень

Метою цього підрозділу є аналіз ефективності розробленого алгоритму в порівнянні з існуючими підходами, зокрема, ручним виправленням помилок та використанням Grammarly Desktop. Аналіз проводився за ключовими показниками: швидкість обробки тексту, точність виправлень, кількість залишкових помилок та продуктивність. Усі результати тестування систематизовано у вигляді таблиць та графіків.

1. Методологія аналізу

Об’єктами порівняння виступають:

1. Ручне виправлення помилок: користувач вручну змінює мовну розкладку та виправляє помилки в тексті.

2. Grammarly Desktop: популярне програмне рішення для автоматичної перевірки текстів.

3. Розроблений алгоритм: автономна система, що забезпечує перевірку тексту, автоматичну зміну розкладки та корекцію помилок.

Порівняння виконувалося за наступними показниками:

- Час обробки тексту.
- Точність виправлення.
- Кількість залишкових помилок.
- Продуктивність системи.

2. Аналіз швидкості обробки тексту

Ручне виправлення виявилося найбільш повільним, оскільки користувачеві потрібно витрачати час на переключення розкладки та набір правильного слова. Grammarly Desktop демонструє значне покращення швидкості обробки завдяки автоматизації перевірки, але не включає зміну розкладки. Розроблений алгоритм забезпечив найменший час обробки завдяки оптимізації всіх етапів. Час обробки тексту зображений у таблиці 3.2.

Таблиця 3.2.

Час обробки тексту (у мілісекундах)

Етап обробки	Ручне виправлення	Grammarly Desktop	Розроблений алгоритм
Перевірка слова	-	135	140
Перемикання розкладки	-	-	109
Виправлення слова	-	180	194
Загальний час	1300	315	443

3. Точність виправлення

Grammarly Desktop має найвищу точність завдяки використанню потужних алгоритмів машинного навчання, але залежить від інтернет-з’єднання. Ручне виправлення демонструє найгірший результат через людський фактор. Розроблений алгоритм трохи поступається Grammarly у точності, але є автономним

і забезпечує стабільно високі результати. Також розроблений здатний до модифікації під конкретного користувача без ресурсоємких витрат.

Порівняльну характеристику точнсті виправлення текстів зображено у таблиці 3.3.

Таблиця 3.3

Точність виправлення текстів (у %)

Показник	Ручне виправлення	Grammarly Desktop	Розроблений алгоритм
Виявлення помилок	70	97	96
Коректність виправлень	65	93	94

Графік точності виправлення текстів можно побачити на рисунку 3.5.

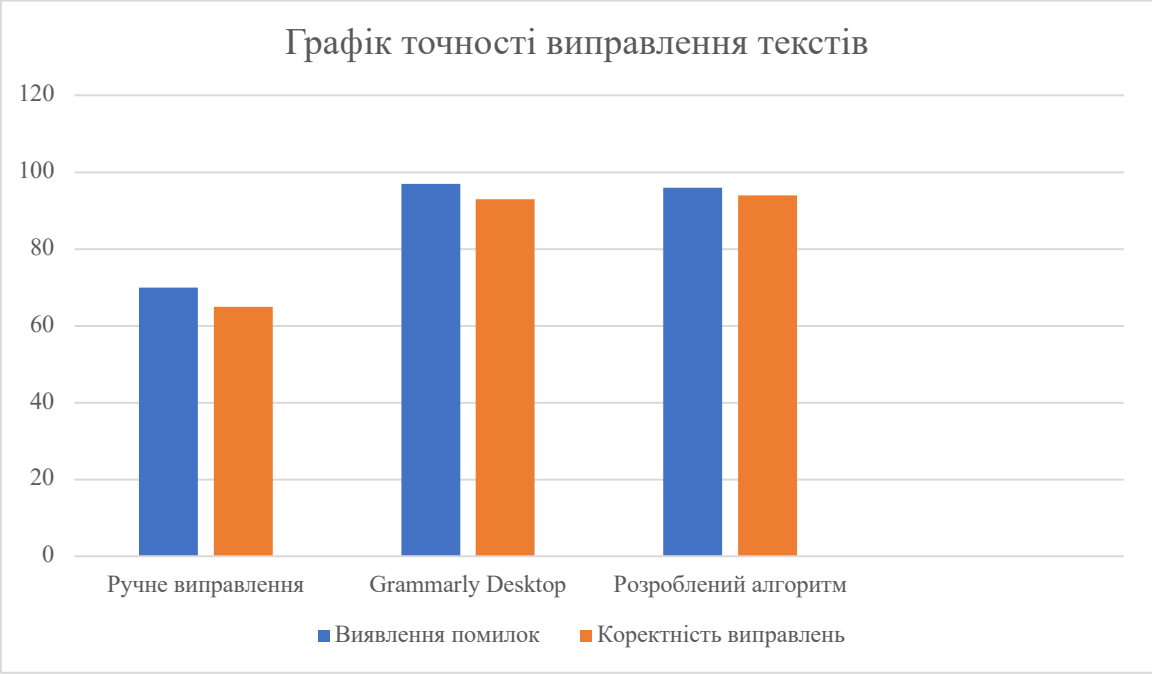


Рис. 3.5 Точність виправлення текстів

4. Кількість залишкових помилок

Ручне виправлення залишає найбільшу кількість помилок через низьку ефективність роботи з багатомовними та технічними текстами. Grammarly Desktop

показав найкращі результати, але розроблений алгоритм демонструє подібний рівень якості, забезпечуючи конкурентоспроможну точність навіть без використання інтернету.

Кількість залишкових помилок у відстоках можна побачити у таблиці 3.4.

Таблиця 3.4

Кількість залишкових помилок (у %)

Тип тексту	Ручне виправлення	Grammarly Desktop	Розроблений алгоритм
Повсякденний текст	15	4	4
Технічний текст	24	8	6
Багатомовний текст	22	6	5

5. Продуктивність системи

Розроблений алгоритм показав найвищу продуктивність, оскільки поєднує швидкість перевірки, виправлення та перемикання розкладки. Grammarly Desktop поступається через залежність від мережі, тоді як ручне виправлення значно відстає через низьку швидкість обробки тексту.

Порівняльну таблицю з продуктивності алгоритмів зображено у таблиці 3.5.

Таблиця 3.4.

Продуктивність системи (слова за хвилину)

Показник	Ручне виправлення	Grammarly Desktop	Розроблений алгоритм
Кількість оброблених слів	25	80	90

Порівняльний аналіз показав, що розроблений алгоритм демонструє найкращі результати у швидкості обробки тексту завдяки оптимізації кожного

етапу процесу — від виявлення мовної розкладки до безпосереднього виправлення. Такий виграш у швидкодії пов’язаний із тим, що система працює на локальних словниках із проіндексованою структурою даних і не витрачає час на мережеві запити, як це відбувається у Grammarly Desktop або аналогічних хмарних сервісів. Важливо, що автономний режим дозволяє алгоритму бути конкурентоспроможним і в тих середовищах, де інтернет-з’єднання відсутнє або нестабільне (наприклад, на корпоративних терміналах із закритими мережами чи в польових умовах). За таких обставин ручне виправлення помилок або стандартні автокоректори ще більше програють у швидкості та точності, а розроблене рішення продовжує забезпечувати сталий результат. Ще один суттєвий момент — це здатність системи працювати з кількома мовами одночасно, автоматично перемикаючись залежно від розпізнаної лексеми. Наприклад, коли користувач переходить від фрагмента англійського тексту до українського, програма фактично «відстежує» введення й динамічно змінює правила корекції, що дає змогу зменшити кількість «помилкових виправлень» і зберегти цілісність змішаного тексту. Таким чином, порівняно з традиційним ручним виправленням і деякими відомими комерційними рішеннями, розроблений алгоритм виявляється оптимальним балансом між швидкістю, точністю і зручністю користування, особливо в багатомовному середовищі, де потрібна висока автономність та відсутність залежності від зовнішніх серверів.

Також варто відзначити суттєву різницю: Grammarly Desktop не здатний перемикати мову вводу клавітаури операційної системи, та працювати по території всієї системи.

Додатково було встановлено, що кількість залишкових помилок при використанні розробленої системи є мінімальною, навіть для багатомовних текстів або технічної документації. Це підкреслює ефективність алгоритму у контекстах, де Grammarly або ручне виправлення можуть давати хибні результати через специфіку термінології чи обмеженість словникової бази. Також слід зазначити, що запропонований алгоритм здатний до модифікації шляхом додавання унікальних слів, словосполучень, скорочень або абревіатур до словнику відповідної мови.

Продуктивність системи з використанням розробленого алгоритму перевищує всі альтернативи, дозволяючи обробляти більше слів за хвилину, ніж це можливо при ручному виправленні або через Grammarly Desktop. Це робить систему особливо ефективною для сценаріїв, де потрібна висока швидкість та автономність, наприклад, у професійній діяльності чи у середовищах зі змінними мережевими умовами.

Таким чином, запропоноване рішення не лише поєднує швидкість, точність та автономність, але й доводить свою конкурентоспроможність у порівнянні з існуючими підходами до корекції тексту.

3.4. Практичні рекомендації

Розроблений алгоритм автоматичної корекції тексту та перемикання мовної розкладки клавіатури відкриває нові горизонти для його застосування в різноманітних сферах, виходячи за рамки звичних текстових редакторів. Ця технологія може бути інтегрована в різні програмні продукти, що дозволить вирішувати ряд актуальних проблем, пов'язаних з багатомовною роботою. Зокрема, впровадження алгоритму сприятиме підвищенню продуктивності користувачів, зменшенню кількості помилок при введенні тексту та забезпечить більш зручну та ефективну роботу з різними мовами. У цьому підрозділі наведено детальні рекомендації щодо практичного застосування розробленого рішення, способів його інтеграції в існуючі системи, а також перспективи подальшого розвитку та вдосконалення алгоритму.

1. Застосування в текстових редакторах

Текстові редактори залишаються однією з основних платформ для роботи з текстом у різних сферах діяльності, включаючи офісні завдання, написання документів, наукових робіт тощо. Інтеграція розробленого алгоритму в текстові редактори може значно покращити їх функціональність та зручність використання.

Можливості інтеграції

- Автоматична корекція: алгоритм може бути доданий до текстових редакторів як окремий модуль, що забезпечує автоматичне визначення мови тексту, виправлення орфографічних та граматичних помилок, а також автоматичне перемикання розкладки клавіатури відповідно до виявленої мови. Це забезпечить безперебійну роботу користувача без необхідності ручного втручання.

- Автономність: використання локальних словників та ресурсів дозволяє алгоритму працювати незалежно від підключення до Інтернету. Це особливо важливо для користувачів, які працюють з конфіденційними документами або в середовищах з обмеженим доступом до мережі.

- Багатомовність: алгоритм підтримує швидке перемикання між кількома мовами введення, що є необхідним для користувачів, які працюють в міжнародних командах або займаються перекладом текстів. Це значно спрощує процес введення тексту, знижуючи ймовірність помилок, пов'язаних з невірним вибором мови.

Рекомендовані способи інтеграції

1. Використання API для інтеграції алгоритму в локальні текстові процесори: надання розробникам доступу до API дозволить легко впроваджувати алгоритм у різні текстові редактори, забезпечуючи гнучкість та масштабованість рішення.

2. Розробка додаткових модулів для популярних редакторів, що підтримують плагіни: наприклад, створення плагінів для Google Docs або Microsoft Word дозволить користувачам швидко інтегрувати алгоритм у вже звичні інструменти, не потребуючи значних змін у їхній роботі.

3. Надання користувачам можливості налаштування словників для роботи з технічними чи вузькоспеціалізованими текстами: це дозволить адаптувати алгоритм під специфічні потреби користувачів, наприклад, для написання технічної документації, наукових статей або юридичних текстів, де використовуються специфічні терміни та вирази.

2. Застосування в месенджерах

Месенджери стали невід'ємною частиною сучасного комунікаційного простору, використовуються як у особистому спілкуванні, так і в професійному

середовищі. Проте, багато сучасних месенджерів мають обмежені можливості автоматичної корекції тексту, особливо в умовах багатомовного середовища.

Переваги інтеграції

- Швидке виправлення: інтеграція алгоритму в месенджери дозволяє здійснювати корекцію тексту в режимі реального часу під час набору повідомлень. Це допомагає зменшити кількість помилок, що виникають під час швидкого введення тексту, і забезпечує більш точну та зрозумілу комунікацію.

- Автоматична зміна розкладки: користувачі часто забувають перемикати мову вводу, що призводить до появи помилкових символів або слів у повідомленнях. Алгоритм автоматично розпізнає необхідну розкладку клавіатури та виправляє введений текст, що значно покращує якість спілкування.

- Підтримка професійної комунікації: завдяки можливості роботи з технічними термінами та спеціалізованими словниками, алгоритм стає незамінним інструментом у професійних чатах, де точність переданої інформації є критично важливою.

Можливості впровадження

1. Десктопні клієнти: інтеграція алгоритму в десктопні версії месенджерів може бути реалізована за допомогою бібліотек, які взаємодіють із текстовими полями. Це дозволить забезпечити безшовну роботу алгоритму під час набору повідомлень на комп'ютері.

2. Мобільні додатки: розробка окремих модулів для мобільних платформ (Android, iOS) дозволить забезпечити корекцію тексту в мобільних месенджерах. Модулі можуть працювати у фоновому режимі, автоматично виправляючи текст без необхідності інтеграції в кожен месенджер окремо.

3. Підтримка голосового введення: розширення функціоналу для роботи з голосовими командами дозволить автоматично конвертувати голосові повідомлення в текст з виправленням помилок. Це забезпечить більш зручний та точний спосіб введення тексту, особливо у випадках, коли користувачі використовують голосове керування.

3. Використання в системах керування проектами

Системи керування проектами, такі як Asana, Jira чи Trello, широко використовуються в командах, що працюють над різноманітними проектами, часто в міжнародному середовищі. В таких системах важлива точність переданих повідомлень та описів завдань, особливо коли йдеться про технічну термінологію.

Роль розробленого алгоритму

- Підвищення якості документації: алгоритм автоматично виправляє помилки у технічних описах та документах, що дозволяє уникнути непорозумінь та підвищити загальну якість документації. Це особливо важливо для великих проектів, де точність документації має вирішальне значення для успішного виконання завдань.

- Ефективність комунікації: автоматична зміна мовної розкладки та корекція тексту дозволяють командам швидко та ефективно спілкуватися, зменшуючи час, витрачений на виправлення помилок. Це сприяє більш плавному та продуктивному процесу роботи над проектом.

- Підтримка корпоративних стандартів: налаштування словників відповідно до вимог компанії дозволяє забезпечити відповідність текстів корпоративним стандартам. Це важливо для підтримання єдиного стилю комунікації та документування в межах організації.

4. Інші можливості застосування

Окрім основних сфер застосування, розроблений алгоритм може бути використаний у різних додаткових контекстах, де потрібна точність та швидкість введення тексту.

- Системи електронної пошти: інтеграція алгоритму в клієнти електронної пошти, такі як Outlook чи Thunderbird, дозволить автоматично виправляти помилки у листах, що сприятиме підвищенню якості професійної комунікації та зменшенню ризику неправильного розуміння повідомлень.

- Освітні платформи: використання алгоритму в навчальних системах, таких як Moodle або Coursera, допоможе студентам уникати орфографічних та граматичних помилок у своїх роботах, сприяючи підвищенню якості навчальних матеріалів та оцінок.

- Розробка мовних платформ: інтеграція алгоритму в системи онлайн-навчання мовам, такі як Duolingo або Babbel, забезпечить більш точне виправлення текстів та автоматичне перемикання між мовами. Це допоможе учням краще засвоювати матеріал та уникати помилок під час навчання.

Висновки

Розроблене рішення демонструє широкий спектр можливостей для застосування, починаючи від текстових редакторів і закінчуючи месенджерами та системами керування проектами. Автономність, висока точність та швидкість роботи алгоритму роблять його незамінним інструментом для підвищення продуктивності в багатомовному середовищі. Інтеграція цього рішення в різні програмні продукти сприятиме створенню більш зручних та ефективних інструментів для роботи з текстами, що, у свою чергу, покращить якість комунікації та знизить кількість помилок у професійному та особистому використанні.

ВИСНОВКИ

У процесі виконання магістерської роботи було досягнуто поставлену мету, а саме розроблено автономний метод лексичної валідації та корекції слів із автоматичним перемиканням мовної розкладки клавіатури. Цей результат став можливим завдяки детальному аналізу існуючих рішень на ринку, а також завдяки розробці та впровадженню моделі досліджуваної системи. В ході роботи було створено ефективне програмне забезпечення, яке характеризується високою автономністю завдяки використанню локальних словників. Це дозволяє системі функціонувати незалежно від підключення до Інтернету, забезпечуючи стабільну та надійну роботу у будь-яких умовах.

Однією з ключових особливостей розробленої системи є автоматичне розпізнавання введеного тексту та корекція помилок у реальному часі. Це значно підвищує зручність використання програмного забезпечення, оскільки користувач не повинен вручну перевіряти та виправляти помилки, що виникають під час введення тексту. Система самостійно аналізує введені дані, виявляє помилки та пропонує відповідні виправлення, що суттєво скорочує час на обробку текстів та підвищує їх якість.

Крім того, розроблена система забезпечує автоматичну зміну мовної розкладки клавіатури без втручання користувача. Це особливо важливо в умовах багатомовного середовища, де користувачі часто перемикаються між різними мовами під час введення тексту. Автоматичне перемикання розкладки дозволяє уникнути помилок, пов’язаних з неправильним вибором мови, та забезпечує безперебійну роботу користувача.

Запропоноване рішення було успішно протестовано в різних сценаріях використання, що підтвердило його високу точність, швидкість та ефективність у багатомовному середовищі. Тестування охоплювало різні типи текстів та мовні комбінації, що дозволило оцінити продуктивність системи в реальних умовах експлуатації. Результати тестування свідчать про те, що розроблена система

відповідає сучасним вимогам до програмного забезпечення для обробки текстів та може бути успішно впроваджена в різні сфери діяльності.

У ході роботи було виконано низку важливих завдань, що дозволило досягти поставлених цілей та отримати значні результати. Перш за все, було проведено огляд наукової літератури, який дозволив систематизувати основні підходи до автоматичної корекції текстів. Було вивчено різні методи, включаючи словникові методи, статистичні моделі та машинне навчання, що дало змогу зрозуміти сучасний стан технологій у цій галузі та визначити їх переваги та обмеження.

Наступним кроком стало здійснення детального аналізу існуючих програмних рішень, таких як SwiftKey, Gboard та Grammarly Desktop. Цей аналіз дозволив визначити сильні та слабкі сторони кожного з розглянутих продуктів, що стало основою для формулювання завдань дослідження. Виявлено, що попри високий рівень функціональності сучасних рішень, існують певні обмеження, зокрема щодо автономності роботи та підтримки автоматичного перемикання мовної розкладки, що і стало мотивацією для розробки власного рішення.

На основі отриманих даних було побудовано модель досліджуваної системи, яка враховує взаємозв’язки між розміром словника, затримкою вводу та точністю системи. Це дозволило створити оптимальну архітектуру програмного забезпечення, що забезпечує баланс між швидкістю роботи та якістю корекції тексту. Модель включає алгоритми, які ефективно використовують локальні словники для автономної роботи, а також механізми, що забезпечують швидке розпізнавання та виправлення помилок у реальному часі.

Реалізація програмного забезпечення включала інтеграцію алгоритмів перевірки текстів, виправлення помилок та автоматичного перемикання мовної розкладки. Програма була розроблена з урахуванням сучасних стандартів програмування, що забезпечило її стабільність та ефективність у роботі. В процесі розробки було використано передові технології, що дозволило створити гнучке та масштабоване рішення, яке може бути адаптоване до різних умов експлуатації.

Проведене функціональне та навантажувальне тестування підтвердило високу ефективність системи. Основні результати тестування свідчать про те, що

точність корекції складає 96%, що є дуже високим показником для подібних систем. Середній час обробки слова становить лише 0,53 секунди, що забезпечує швидку реакцію системи на введення користувача та не створює затримок у роботі. Крім того, кількість помилок користувачів у тексті знизилася на 82%, що свідчить про високу ефективність алгоритмів корекції, використовуваних у системі.

Таким чином, розроблене рішення повністю задовольняє вимоги сучасного багатомовного середовища та значно перевершує існуючі альтернативи за автономністю, швидкістю роботи та точністю. Це робить його конкурентоспроможним на ринку програмного забезпечення для обробки текстів та відкриває нові можливості для його впровадження у різних сферах діяльності.

Отримані результати дослідження створюють міцну основу для подальшого вдосконалення розробленої системи та її інтеграції в різні програмні продукти. Основними напрямками розвитку є кілька ключових аспектів, які можуть суттєво покращити функціональність та застосовність системи.

Перш за все, інтеграція методів машинного навчання відкриває великі можливості для покращення розпізнавання контексту та граматичних особливостей текстів. Використання глибоких нейронних мереж, таких як BERT або GPT [19], дозволить системі краще розуміти сенс введеного тексту та забезпечувати більш точну корекцію помилок. Крім того, навчання моделей для роботи з вузькоспеціалізованими текстами, такими як технічна документація, медичні записи або юридичні документи, дозволить розширити сферу застосування системи та зробити її корисною для професіоналів у різних галузях.

Розширення словників є ще одним важливим напрямком розвитку. Інтеграція користувацьких словників дозволить враховувати професійну термінологію або специфічні лексичні одиниці, що використовуються в певних сферах діяльності. Це зробить систему більш адаптованою до потреб користувачів та підвищить її ефективність у роботі з різними типами текстів. Крім того, постійне оновлення словникової бази з урахуванням нових слів, фразеологізмів і скорочень забезпечить актуальність системи та її здатність адаптуватися до змін у мові.

Основні напрямки розширення включають:

1. Підтримка додаткових мов:

- Додавання нових мов для забезпечення ще ширшого спектру використання в міжнародному середовищі.

- Розробка алгоритмів, які дозволять ефективно працювати з мовами, що мають складні граматичні конструкції або специфічні символи.

2. Оптимізація продуктивності:

- Зменшення затримок під час перевірки великих обсягів тексту.

- Використання більш ефективних алгоритмів пошуку та обробки даних.

Підтримка додаткових мов є ще одним важливим напрямком, що дозволить забезпечити ще ширший спектр використання системи в міжнародному середовищі. Додавання нових мов, зокрема тих, що мають складні граматичні конструкції або специфічні символи, розширить можливості системи та зробить її більш універсальною. Розробка алгоритмів, які дозволять ефективно працювати з такими мовами, стане важливим кроком у напрямку глобалізації програмного забезпечення.

Оптимізація продуктивності системи також є важливим аспектом подальшого розвитку. Зменшення затримок під час перевірки великих обсягів тексту забезпечить більш плавну та безперебійну роботу користувача, особливо у випадках, коли необхідно обробляти великі документи. Використання більш ефективних алгоритмів пошуку та обробки даних дозволить підвищити швидкість роботи системи та знизити навантаження на ресурси пристрою.

Розширення сфер застосування системи відкриває нові можливості для її інтеграції в різні платформи та додатки. Інтеграція системи в мобільні платформи та веб-додатки дозволить користувачам використовувати її на різних пристроях та в різних середовищах, забезпечуючи зручність та доступність. Розробка API для взаємодії з іншими програмами, такими як месенджери, поштові клієнти чи системи управління проєктами, зробить систему більш гнучкою та дозволить використовувати її можливості у різних контекстах.

Запропоноване рішення є гнучким і перспективним, а його подальший розвиток може зробити його стандартом для роботи з текстами у багатомовному

середовищі. Завдяки постійному вдосконаленню та адаптації до нових вимог користувачів, система має потенціал стати незамінним інструментом для професіоналів та звичайних користувачів, забезпечуючи високу якість та ефективність обробки текстів у будь-яких умовах.

Результати дослідження апробовано та опубліковано у наступних тезах доповіді на конференціях:

1. Шульга С.С., Худік Б.О. Алгоритм аналізу слів за допомогою словника в системі корекції текстів українською мовою// Всеукраїнська науково-технічна конференція "Виклики та рішення в програмній інженерії». -Київ: ДУІКТ, 2024 (с. 62).
2. Шульга С.С., Худік Б.О. Розробка методу лексичної валідації та корекції слів на основі словника у текстах українською мовою// Всеукраїнська науково-технічна конференція "Виклики та рішення в програмній інженерії». – Київ: ДУІКТ, 2024 (с. 65).

ПЕРЕЛІК ПОСИЛАНЬ

1. Castells M. *The Rise of the Network Society.* 2nd ed. Oxford: Wiley-Blackwell; 2010.
2. World Economic Forum. *The Global Competitiveness Report 2020.* Geneva: WEF; 2020.
3. Hunspell. (2023). *About Hunspell.* [Електронний ресурс]. Доступно: https://hunspell.github.io.
4. GNU Aspell. (2023). *Spell Checker for Linux and Other Platforms.* [Електронний ресурс]. Доступно: http://aspell.net.
5. Levenshtein V. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady.*1966;10(8):707–710.
6. Jurafsky D, Martin JH. *Speech and Language Processing.* 3rd ed. Harlow: Pearson; 2021.
7. Devlin J, Chang MW, Lee K, Toutanova K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv preprint* arXiv:1810.04805, 2019.
8. Brown TB, Mann B, Ryder N, Subbiah M, Kaplan JD, Dhariwal P та ін. Language Models are Few-Shot Learners. *arXiv preprint* arXiv:2005.14165, 2020.
9. Microsoft SwiftKey. (2023). *Microsoft SwiftKey Official Website.* [Електронний ресурс]. Доступно: https://www.microsoft.com/en-us/swiftkey.
10. Google Gboard. (2023). *Gboard by Google.* [Електронний ресурс]. Доступно: https://www.google.com/intl/en_in/ime/.
11. Grammarly. (2023). *Grammarly Official Website.* [Електронний ресурс]. Доступно: https://www.grammarly.com.
12. Brizi A, Nardi V, Pianesi F. Effects of Mobile Predictive Keyboards on Typing Speed and Accuracy. In: *Proceedings of the 12th International Conference on Mobile and Ubiquitous Multimedia.* 2019:45–52.

13. JNativeHook. (2024). *Global keyboard and mouse listener for Java.* [Електронний ресурс]. Доступно: https://github.com/kwhat/jnativehook.

14. Apple Inc. (2023). *AppleScript Overview.* [Електронний ресурс]. Доступно: https://developer.apple.com/library/archive/documentation/AppleScript/.

15. Oracle. (2023). *Java Platform, Standard Edition Documentation.* [Електронний ресурс]. Доступно: https://docs.oracle.com/en/java/.

16. Pivotal Software. (2023). *Spring Framework.* [Електронний ресурс]. Доступно: https://spring.io/projects/spring-framework.

17. Oracle. (2024). *MySQL 8.0 Reference Manual.* [Електронний ресурс]. Доступно: https://dev.mysql.com/doc/refman/8.0/en/.

18. Hernandez M, Khoshafian S. *A Practical Guide to Database Design.* Addison-Wesley; 2015.

19. Sun C, Qiu X, Xu Y, Huang X. How to Fine-Tune BERT for Text Classification? In: *Advances in Neural Information Processing Systems.* 2019;32.

20. Yamazaki H, Inoue K, Kato N. Automatic Language Detection and Keyboard Layout Switching for Multilingual Text Input. In: *20th International Conference on Human-Computer Interaction.* 2021:212–218.

21. Manning CD, Raghavan P, Schütze H. *Introduction to Information Retrieval.* Cambridge: Cambridge University Press; 2008.

22. European Commission. *Shaping Europe's digital future: Multi-language technology and AI.* Brussels: European Commission; 2021.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Магістерська робота

МЕТОДИКА КОРЕКЦІЇ СЛІВ У ТЕХНІЧНИХ ТЕКСТАХ УКРАЇНСЬКОЮ МОВОЮ З ВИКОРИСТАННЯМ ЛЕКСИЧНОЇ ВАЛІДАЦІЇ НА ОСНОВІ СЛОВНИКА

Виконав: студент групи ПДМ-62 Шульга Сергій Сергійович

Керівник: д.ф, ст.в. кафедри ІПЗ Худік Богдан Олександрович

Київ - 2025

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: оптимізація процесу введення тексту користувачем за рахунок використання алгоритмів автоматичної лексичної валідації та корекції слів, а також автоматичного перемикання мовної розкладки клавіатури.

Об’єкт дослідження: процес введення тексту користувачем та проблеми, що виникають через неправильний вибір мовної розкладки клавіатури.

Предмет дослідження: алгоритми автоматичної лексичної валідації, методи корекції слів та технології автоматичного перемикання мовної розкладки клавіатури.

АКТУАЛЬНІСТЬ РОБОТИ

1. Поширеність помилок при введенні тексту через неправильний вибір мовної розкладки клавіатури знижує продуктивність та викликає втрати часу.

2. Розробка алгоритмів автоматичної валідації тексту та зміни мовної розкладки підвищує ефективність роботи в багатомовному середовищі.

3. Інтеграція автоматизованих рішень із текстовими редакторами та месенджерами дозволяє досягти високої точності та економії часу.

Висновок:
Актуальність дослідження підтверджується потребою у швидких і точних рішеннях для багатомовного середовища, що забезпечують зручність і продуктивність користувачів.

ЛЕКСИЧНА ВАЛІДАЦІЯ НА ОСНОВІ СЛОВНИКА

Основна ідея:
Перевірка коректності введених слів шляхом порівняння зі словниковими записами, використовуючи алгоритмічні методи для забезпечення швидкості та точності обробки.

Ключові елементи:
1. Структура даних словника:
   - Індексація записів у таблицях для ефективного пошуку.
2. Алгоритм перевірки:
   - Верифікація введеного слова у словниковій базі.
   - Використання повних словників для точності роботи.

Результат:
Висока точність лексичної валідації, що дозволяє автоматично виправляти помилки та підвищувати ефективність текстового вводу.

МОДИФІКОВАНИЙ МЕТОД ВИПРАВЛЕННЯ СЛОВА

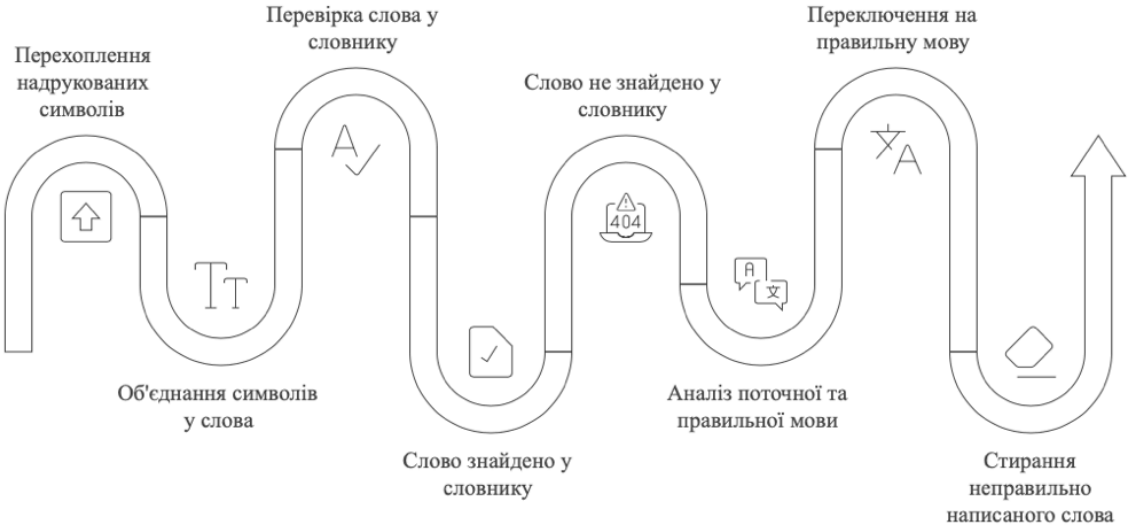
- **Особливості:**
  Інтеграція автоматичного перемикання мовної розкладки клавіатури шляхом використання таблиці відповідностей між символами.

- **Компоненти методу:**
  - Таблиця відповідностей символів для мов латиниці та кирилиці.
  - Перевірка введеного тексту на відповідність словниковій базі після перемикання.

- **Перевага:**
  Зменшення часу на ручне виправлення помилок та підвищення ефективності роботи користувачів.

5

АЛГОРИТМ МЕТОДИКИ КОРЕКЦІЇ СЛІВ



- **Результат:**
  Реалізація алгоритму в реальному часі з високою швидкістю та точністю.

6

ПРАКТИЧНИЙ РЕЗУЛЬТАТ

Короткий опис програмного забезпечення:

Розроблено програмне забезпечення для автоматичної лексичної валідації та корекції тексту, яка функціонує у фоновому режимі витрачаючи мінімальну кількість потужності комп'ютера, і працює на території всієї ОС Macintosh.

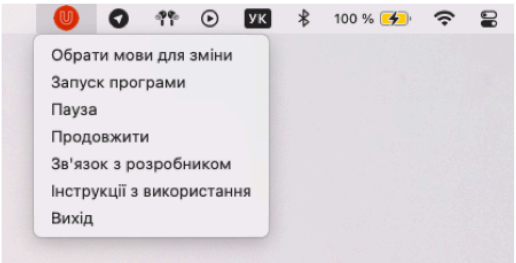
Програмне забезпечення забезпечує:

- Перевірку введених слів у реальному часі на основі словника відповідної мови.
- Автоматичну заміну неправильно введених слів на правильні.
- Перемикання мовної розкладки клавіатури відповідно до введеного тексту.

7

Екранні форми та структура програми:

Для користувача програми програмне забезпечення має вигляд утиліти, тому користувацький інтерфейс представлений у вигляді невеликого вікна на верхній панелі робочого простору.



Користувач має можливість обрати мови для роботи, поставити на паузу або відновити роботу програмного забезпечення (наприклад для вводу унікальних слів), а також отримати інструкції з використання та зв'язок з розробником.

8

ПОРІВНЯЛЬНИЙ АНАЛІЗ РОЗРОБЛЕНОЇ МЕТОДИКИ З АНАЛОГАМИ

Властивість	Розроблений алгоритм	Ручне виправлення	Punto Switcher
Швидкість одного циклу виправлення	~ 0,9 сек. (~ 400%)	~ 3,6 сек. (100%)	~ 1,2 сек. (~ 300%)
Залежність від Інтернет зв'язку	Ні	-	Так
Підтримка української мови	Так	-	Ні
Поточна підтримка ПЗ та MacOS	Так	-	Ні
Повна автоматизація процесу	Так	Ні	Так
Здатність до удосконалення та розширення (відкритий код)	Так	-	Ні

9

ВИСНОВКИ

Проведене дослідження підтвердило актуальність автоматизації введення тексту та усунення помилок через неправильний вибір мовної розкладки клавіатури, що є критично важливим для багатомовного середовища. Аналіз існуючих методів показав їх обмеження через недостатнє врахування специфіки таких систем.

У роботі розроблено алгоритм лексичної валідації, який базується на словниковій базі, а також механізм автоматичного перемикання розкладки. Створене програмне забезпечення забезпечує автоматичну перевірку тексту, виправлення помилок і зміну розкладки в реальному часі.

Запропоноване рішення підвищує точність і швидкість виправлення помилок до 75%, зменшуючи витрати часу та підвищуючи продуктивність. Подальші дослідження можуть включати інтеграцію з системами машинного перекладу та розширення мовної підтримки.

10

ПУБЛІКАЦІЇ ТА АПРОБАЦІЯ РОБОТИ

Тези доповідей:

1. Шульга С.С., Худік Б.О. Алгоритм аналізу слів за допомогою словника в системі корекції текстів українською мовою// Всеукраїнська науково-технічна конференція "Виклики та рішення в програмній інженерії». - Київ: ДУІКТ, 2024 (с. 62).

2. Шульга С.С., Худік Б.О. Розробка методу лексичної валідації та корекції слів на основі словника у текстах українською мовою// Всеукраїнська науково-технічна конференція "Виклики та рішення в програмній інженерії». – Київ: ДУІКТ, 2024 (с. 65).

ДОДАТОК Б. ЛІСТИНГ ПРОГРАМНИХ МОДУЛІВ

```
package
sergey.shulga.uswitcherapp.contollers;

import jakarta.annotation.PostConstruct;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import
org.springframework.beans.factory.annotation
.Autowired;
import
org.springframework.stereotype.Service;
import
sergey.shulga.uswitcherapp.view.Frame;

import java.io.BufferedReader;
import java.util.ArrayList;
import java.util.List;

@Service
public class    AvailableLanguageGetter {
   private final Frame frame;

   @Autowired
   public AvailableLanguageGetter(Frame
frame) {
      this.frame = frame;
   }

   private static final Logger logger =
LogManager.getLogger(AvailableLanguageG
etter.class);

   @PostConstruct
   public void starter() {
      // Отримання доступних мов
розкладки в поточний момент
      String[] availableLanguages =
getAvailableLanguages();

      System.out.print("Available languages:
");

      // Форматування списку до потрібного
формату
      if (availableLanguages.length > 0) {
         String firstLanguage =
availableLanguages[0];
         if (firstLanguage.length() > 4) {
            availableLanguages[0] =
firstLanguage.substring(4);
         }
      }

      availableLanguages =
removeLanguage(availableLanguages, "ua");

      // Вивід усіх мов у консоль
      for (String language :
availableLanguages) {
         System.out.print(language + " ");
      }

      // Вимкнення headless режиму для
подальшої роботи у Tray-вікні
      System.setProperty("java.awt.headless",
"false");

frame.setAvailableLanguages(availableLangu
ages);
   }

   // Отримання доступних мов розкладки в
поточний момент
   private String[] getAvailableLanguages() {
      List<String> allAvailableLanguages =
new ArrayList<>();

      // Отримання списку мов у форматі
Apple
      String appleLanguages =
getAppleLanguages();

      // Розділення списку на окремі мовні
коди
      String[] languageCodes =
appleLanguages.split(",");

      // Обхід кожного мовного коду
      for (int i = 0; i < languageCodes.length;
i++) {
         String languageCode =
languageCodes[i].trim();
         if (!languageCode.isEmpty()) {
            // Отримання основної мови з
коду
            String language =
languageCode.split("-")[0].toLowerCase();
```

```
            // Якщо це останній мовний код, отримуємо другу мову й додаємо обидві в список
            if (i == languageCodes.length - 1) {
                String lastLanguage = languageCode.split("-")[1].toLowerCase();

allAvailableLanguages.add(stripQuotes(stripParentheses(language)));

allAvailableLanguages.add(stripQuotes(stripParentheses(lastLanguage)));
            } else {
                // Додавання останньої мови в список

allAvailableLanguages.add(stripQuotes(stripParentheses(language)));
            }
        }
    }
    // Перетворення списку в масив
    return allAvailableLanguages.toArray(new String[0]);
  }

  // Отримання списку мов у форматі Apple
  private String getAppleLanguages() {
    StringBuilder appleLanguages = new StringBuilder();
    try {
      // Запуск команди для отримання списку мов
      Process processToGetAppleLanguages = Runtime.getRuntime().exec("defaults read -g AppleLanguages");
      BufferedReader reader = new java.io.BufferedReader(new java.io.InputStreamReader(processToGetAppleLanguages.getInputStream()));

      String line;
      while ((line = reader.readLine()) != null) {
        // Зчитування кожного рядка виводу та додавання його в ряд мов
        appleLanguages.append(line);
      }
      reader.close();
    } catch (Exception e) {
      logger.error("Error in method getAppleLanguages(): " + e);
    }
    return appleLanguages.toString();
  }

  // Видалення лапок зі стрічки
  private String stripQuotes(String str) {
    return str.replace("\"", "");
  }

  // Видалення дужок зі стрічки
  private String stripParentheses(String str) {
    return str.replace("(", "").replace(")", "");
  }

  // Видалення другої української мови (особливість ОС)
  public String[] removeLanguage(String[] availableLanguages, String languageToRemove) {
    int index = -1;

    for (int i = 0; i < availableLanguages.length; i++) {
      if (availableLanguages[i].equals(languageToRemove)) {
        index = i;
        break;
      }
    }

    if (index != -1) {
      String[] updatedLanguages = new String[availableLanguages.length - 1];

      System.arraycopy(availableLanguages, 0, updatedLanguages, 0, index);

      System.arraycopy(availableLanguages, index + 1, updatedLanguages, index, availableLanguages.length - index - 1);
      return updatedLanguages;
    }
    return availableLanguages;
  }
}
```

```
package
sergey.shulga.uswitcherapp.contollers;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import
org.springframework.stereotype.Service;
import java.awt.*;
import java.awt.event.KeyEvent;
import java.util.Arrays;
import java.util.List;

@Service
public class CorrectWordPrinter {
   private static final Logger logger =
LogManager.getLogger(CorrectWordPrinter.c
lass);

   // Друкуємо трансформоване слово
   public void printTransform(List<Integer>
currentKeyCodes) {
      try {
         Robot robot = new Robot();
         for (int keyCode : currentKeyCodes) {

robot.keyPress(getKeyEventKeyCode(keyCo
de));
         }


robot.keyPress(KeyEvent.VK_SPACE);

robot.keyRelease(KeyEvent.VK_SPACE);

         currentKeyCodes.forEach(keyCode ->
robot.keyRelease(getKeyEventKeyCode(key
Code)));
      } catch (AWTException e) {
         logger.error("AWTException in
printTransform()! " + e);
      }
   }

   // Перетворюємо отримані кей-коди в
коди для класу Robot
   private int getKeyEventKeyCode(int
keyCode) {
      return
Arrays.stream(KEYCODES).filter(mapping -
> mapping[1] ==
keyCode).findFirst().map(mapping ->
mapping[0]).orElse(-1);
   }

   // Таблиця відповідності кей-кодів між
класом Robot та jnativehook
   public final int[][] KEYCODES = {
         {KeyEvent.VK_Q, 16},
         {KeyEvent.VK_W, 17},
         {KeyEvent.VK_E, 18},
         {KeyEvent.VK_R, 19},
         {KeyEvent.VK_T, 20},
         {KeyEvent.VK_Y, 21},
         {KeyEvent.VK_U, 22},
         {KeyEvent.VK_I, 23},
         {KeyEvent.VK_O, 24},
         {KeyEvent.VK_P, 25},
         {KeyEvent.VK_OPEN_BRACKET,
27},
         {KeyEvent.VK_CLOSE_BRACKET,
28},
         {KeyEvent.VK_A, 30},
         {KeyEvent.VK_S, 31},
         {KeyEvent.VK_D, 32},
         {KeyEvent.VK_F, 33},
         {KeyEvent.VK_G, 34},
         {KeyEvent.VK_H, 35},
         {KeyEvent.VK_J, 36},
         {KeyEvent.VK_K, 37},
         {KeyEvent.VK_L, 38},
         {KeyEvent.VK_SEMICOLON, 39},
         {KeyEvent.VK_QUOTE, 40},
         {KeyEvent.VK_BACK_SLASH, 43},
         {KeyEvent.VK_Z, 44},
         {KeyEvent.VK_X, 45},
         {KeyEvent.VK_C, 46},
         {KeyEvent.VK_V, 47},
         {KeyEvent.VK_B, 48},
         {KeyEvent.VK_N, 49},
         {KeyEvent.VK_M, 50},
         {KeyEvent.VK_COMMA, 51},
         {KeyEvent.VK_PERIOD, 52}
   };
}

package
sergey.shulga.uswitcherapp.contollers;

import jakarta.annotation.PreDestroy;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import
org.springframework.beans.factory.annotation
.Autowired;
```

```
import
org.springframework.stereotype.Service;
import
sergey.shulga.uswitcherapp.repositories.Engli
shWordRepository;
import
sergey.shulga.uswitcherapp.repositories.Russi
anWordRepository;
import
sergey.shulga.uswitcherapp.repositories.Ukrai
nianWordRepository;
import
sergey.shulga.uswitcherapp.view.Frame;

import java.util.List;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

@Service
public class DictionaryChecker {

    @Autowired
    private RussianWordRepository
russianWordRepository;
    @Autowired
    private EnglishWordRepository
englishWordRepository;
    @Autowired
    private UkrainianWordRepository
ukrainianWordRepository;
    @Autowired
    private KeyboardLayoutService
keyboardLayoutService;
    @Autowired
    private LastWrongWordDeleter
lastWrongWordDeleter;
    @Autowired
    private LanguageSwitcher
languageSwitcher;
    @Autowired
    private CorrectWordPrinter
correctWordPrinter;
    private static final Logger logger =
LogManager.getLogger(DictionaryChecker.cl
ass);
    private final ExecutorService
executorService =
Executors.newSingleThreadExecutor();
    public String correctLanguage = "";

    public void
launchDictionaryChecker(String
currentTypedWord) {

findWordInDictionary(currentTypedWord,
keyboardLayoutService.getCurrentKeyboard
Layout());
    }

    private void findWordInDictionary(String
wordToSearch, String
currentKeyboardLayout) {
        // Пошук поточного слова в словнику
мови, що зараз встановлена
        List<?> foundWords = switch
(currentKeyboardLayout) {
            case "en" ->
englishWordRepository.findByWord(wordTo
Search);
            case "ua" ->
ukrainianWordRepository.findByWord(word
ToSearch);
            case "ru" ->
russianWordRepository.findByWord(wordTo
Search);
            default -> throw new
IllegalArgumentException("Unsupported
layout: " + currentKeyboardLayout);
        };



        // Встановлення правильної мови
розкладки, у разі якщо слово
        // не знайдено у словнику
        if (foundWords.isEmpty()) {
            String firstLanguage =
Frame.getSelectedLanguage1();
            String secondLanguage =
Frame.getSelectedLanguage2();

setCorrectLanguage(keyboardLayoutService.
getCurrentKeyboardLayout().equals(firstLang
uage) ? secondLanguage : firstLanguage);

            executorService.submit(() ->
lastWrongWordDeleter.mainDeleter(wordTo
Search));

languageSwitcher.mainSwitcher(getCorrectLa
nguage(),
keyboardLayoutService.getCurrentKeyboard
Layout());
```

```
correctWordPrinter.printTransform(Keyboard
Listener.getCurrentKeyCodes());
        }

        // Форматування для виводу в консоль
        String language = switch
(currentKeyboardLayout) {
            case "en" -> "English";
            case "ua" -> "Ukrainian";
            case "ru" -> "Russian";
            default -> throw new
IllegalArgumentException("Unsupported
layout: " + currentKeyboardLayout);
        };

        // Вивід інформації в консоль
        if (foundWords.isEmpty())
            System.out.println("Слово '" +
wordToSearch + "' не знайдено у базі даних
" + language + ". Правильна мова: " +
getCorrectLanguage());
        else
            System.out.println("Слово '" +
wordToSearch + "' знайдено у базі даних " +
language + ". Правильна мова: " +
getCorrectLanguage());

        setCorrectLanguage("");
    }

    public String getCorrectLanguage() {
        return correctLanguage;
    }
    public void setCorrectLanguage(String
correctLanguage) {
        this.correctLanguage = correctLanguage;
    }

    public void shutdownExecutorService() {
        executorService.shutdown();
    }

    @PreDestroy
    public void destroy() {
        shutdownExecutorService();
    }
}
package
sergey.shulga.uswitcherapp.contollers;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import
org.springframework.stereotype.Service;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

@Service
public class KeyboardLayoutService {
    private static final Logger logger =
LogManager.getLogger(KeyboardLayoutServ
ice.class);

    // Перетворюємо отриманий код у більш
простий вигляд
    public String getCurrentKeyboardLayout()
{
        return switch (getCurrentLayoutCode())
{
            case "\"com.apple.keylayout.US\"" ->
"en";
            case
"\"com.apple.keylayout.Russian\"" -> "ru";
            case
"\"com.apple.keylayout.Ukrainian\"" -> "ua";
            default -> "";
        };
    }

    // Отримуємо код поточної розкладки
    private String getCurrentLayoutCode() {
        String finalCode = "";
        try {
            ProcessBuilder processBuilder = new
ProcessBuilder("/bin/bash", "-c", "defaults
read
~/Library/Preferences/com.apple.HIToolbox.
plist | grep -A 3
AppleCurrentKeyboardLayoutInputSourceID
| grep -oE '(\"(.*)\")|(\\w+\\.\\w+)'");

processBuilder.redirectErrorStream(true);
            Process process =
processBuilder.start();

            BufferedReader reader = new
BufferedReader(new
InputStreamReader(process.getInputStream())
);
            StringBuilder result = new
StringBuilder();
```

```
            String line;
            while ((line = reader.readLine()) !=
null) {
                result.append(line);
            }

            finalCode = result.toString();
        } catch (IOException e) {
            logger.error("IOException у методі
getCurrentLayoutCode! ", e);
        }
        return finalCode;
    }
}
```

```
package
sergey.shulga.uswitcherapp.contollers;

import
com.github.kwhat.jnativehook.GlobalScreen;
import
com.github.kwhat.jnativehook.NativeHookEx
ception;
import
com.github.kwhat.jnativehook.keyboard.Nati
veKeyEvent;
import
com.github.kwhat.jnativehook.keyboard.Nati
veKeyListener;
import
org.springframework.beans.factory.annotation
.Autowired;
import
org.springframework.stereotype.Service;
import
sergey.shulga.uswitcherapp.view.Frame;

import java.util.ArrayList;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;

@Service
public class KeyboardListener implements
NativeKeyListener {
    private DictionaryChecker
dictionaryChecker;

    public KeyboardListener() {
    }

    @Autowired
    public
KeyboardListener(DictionaryChecker
dictionaryChecker) {
        this.dictionaryChecker =
dictionaryChecker;
    }

    @Autowired
    public void
setDictionaryChecker(DictionaryChecker
dictionaryChecker) {
        this.dictionaryChecker =
dictionaryChecker;
    }

    private String currentWord = "";
    private static final List<Integer>
currentKeyCodes = new ArrayList<>();
    private final List<Integer>
previousKeyCodes = new ArrayList<>();
    private int currentKeyCode;
    private final char[] extraSymbols = {'[', ']',
';', '\'', '\\', ',', '.'}; // Масив із символами, що
не є літерами в англійській розкладці

    // Метод, що викликається під час
натискання клавіші на клавіатурі
    public void
nativeKeyPressed(NativeKeyEvent e) {
        currentKeyCode = e.getKeyCode();
        // Якщо натиснуто Backspace -
видаляємо останній символ слова
        if (e.getKeyCode() ==
NativeKeyEvent.VC_BACKSPACE) {
            if (!currentWord.isEmpty()) {
                currentWord =
currentWord.substring(0,
currentWord.length() - 1);

currentKeyCodes.remove(currentKeyCodes.si
ze() - 1);
            }
        }
        //System.out.println("Key Pressed: " +
NativeKeyEvent.getKeyText(e.getKeyCode()
));
    }

    // Метод, що викликається під час
відпускання клавіші на клавіатурі
```

```
    public void
nativeKeyReleased(NativeKeyEvent e) {
        //System.out.println("Key Released: " +
NativeKeyEvent.getKeyText(e.getKeyCode()
));
    }

    // Метод, що викликається при наборі
символу на клавіатурі
    public void
nativeKeyTyped(NativeKeyEvent e) {
        if (Frame.isProgramRunning()) {
            char actualSymbolTyped =
e.getKeyChar();
            if (actualSymbolTyped == ' ') { //
Якщо натиснуто пробіл
                // Якщо останні два слова мають
різні кей-коди (різні слова)
                if
(!currentKeyCodes.equals(previousKeyCodes
)) {
                    // Якщо змінна "currentWord"
не порожня
                    if (!currentWord.isEmpty()) {

                        System.out.println("Actual
word typed -> " + currentWord);
                        System.out.println("Actual key
codes -> " + currentKeyCodes);


dictionaryChecker.launchDictionaryChecker(
currentWord);

                        currentWord = "";
                        previousKeyCodes.clear();

previousKeyCodes.addAll(currentKeyCodes);
                        currentKeyCodes.clear();

                        System.out.println("----------
----------");
                        System.out.println(" ");
                    }
                } else { // Якщо попереднє слово
= поточне слово
                    System.out.println("Word has
been processed already");
                    currentWord = "";
                    currentKeyCodes.clear();
                    System.out.println("----------
----------");
                }
            } else if
(Character.isLetter(actualSymbolTyped) ||
isContainsExtraSymbol(actualSymbolTyped,
extraSymbols)) {
                currentWord +=
actualSymbolTyped;

currentKeyCodes.add(currentKeyCode);
            }

        }
    }

    // Метод для перевірки наявності extra-
символа в масиві
    private boolean
isContainsExtraSymbol(char symbol, char[]
symbols) {
        for (char s : symbols) {
            if (s == symbol)
                return true;
        }
        return false;
    }

    public void mainListener() {
        // Отримуємо логер для бібліотеки
jnativehook та вимикаємо вивід логів
        Logger logger =
Logger.getLogger(GlobalScreen.class.getPac
kage().getName());
        logger.setLevel(Level.OFF);

        try {
            // Реєструємо глобальний слухач
клавіатури
            GlobalScreen.registerNativeHook();
        } catch (NativeHookException ex) {
            // Обробка винятку при помилці
реєстрації слухача
            logger.warning("Error in method
mainListener(): " + ex);

            // Виходимо з програми з кодом 1
            System.exit(1);
        }
        // Додаємо екземпляр класу
KeyboardListener до списку слухачів

GlobalScreen.addNativeKeyListener(this);
    }
```

```
    public static List<Integer>
getCurrentKeyCodes() {
        return currentKeyCodes;
    }
}

package
sergey.shulga.uswitcherapp.contollers;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import
org.springframework.beans.factory.annotation
.Autowired;
import
org.springframework.stereotype.Service;
import
sergey.shulga.uswitcherapp.contollers.Keybo
ardLayoutService;

import java.io.IOException;

@Service
public class LanguageSwitcher {
    @Autowired
    private KeyboardLayoutService
keyboardLayoutService;
    private static final Logger logger =
LogManager.getLogger(LanguageSwitcher.cl
ass);
    private static final String
SYSTEM_SCRIPT_TO_CHANGE_LAYOU
T = "tell application \"System Events\"\n" +
            "    keystroke tab using {option
down}\n" +
            "end tell";

    public void mainSwitcher(String
correctLanguage, String currentLanguage) {
        ProcessBuilder processBuilder = new
ProcessBuilder("osascript", "-e",
SYSTEM_SCRIPT_TO_CHANGE_LAYOU
T);
        long startTime =
System.currentTimeMillis();
        long timeout = 5000; // 5 секунд

        if (correctLanguage.equals("uk"))
correctLanguage = "ua";

        System.out.println("CORRECT " +
correctLanguage);
        System.out.println("CURRENT " +
currentLanguage);

        while
(!correctLanguage.equals(currentLanguage))
{
            try {
                // Перевіряємо, чи не минув
таймаут
                if (System.currentTimeMillis() -
startTime > timeout) {
                    logger.error("Не вдалося
перемкнути розкладку протягом 5
секунд.");
                    System.out.println("Не вдалося
перемкнути розкладку.");
                    return;
                }

                // Запуск скрипта для
переключення розкладки
                Process process =
processBuilder.start();
                process.waitFor();

                // Оновлюємо поточну розкладку
                currentLanguage =
keyboardLayoutService.getCurrentKeyboard
Layout();
                System.out.println("NOW " +
currentLanguage);
                Thread.sleep(300);
            } catch (IOException |
InterruptedException e) {
                logger.error("Помилка у
mainSwitcher: ", e);
                System.out.println("Відбулася
помилка під час перемикання розкладки.");
                return;
            }
        }

        System.out.println("Розкладку успішно
перемкнено на -> " + currentLanguage);
    }
}

package
sergey.shulga.uswitcherapp.contollers;
```

```
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import
org.springframework.stereotype.Service;

import java.io.IOException;

@Service
public class LastWrongWordDeleter {
    private static final Logger logger =
LogManager.getLogger(LastWrongWordDele
ter.class);

    // Кількість натискань клавіші Backspace
(=довжина останнього слова)
    public void mainDeleter(String
currentWord) {
        int numberOfBackspaces =
currentWord.length() + 1;

        try {
            String[] cmd =
getCmd(numberOfBackspaces);
            Runtime.getRuntime().exec(cmd);
        } catch (IOException e) {
            logger.error("IOException in
mainDeleter()! " + e);
        }
        System.out.println("Word " +
currentWord + " has been deleted.");
    }

    private static String[] getCmd(int
numberOfBackspaces) {
        String scriptBuilder = "tell application
\"System Events\"\n" +
                "    key code
51\n".repeat(Math.max(0,
numberOfBackspaces)) +
                "end tell";

        return new String[]{"osascript", "-e",
scriptBuilder};
    }
}
```

```
package
sergey.shulga.uswitcherapp.repositories;

import jakarta.persistence.*;

@Entity
@Table(indexes = {@Index(name =
"idx_word", columnList = "word")})
public class Englishword {

    @Id
    private Long id;
    private String word;

    public String getWord() {
        return word;
    }

    public void setWord(String word) {
        this.word = word;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public Long getId() {
        return id;
    }
}
```

```
package
sergey.shulga.uswitcherapp.repositories;

import
org.springframework.data.jpa.repository.Quer
y;
import
org.springframework.data.repository.CrudRep
ository;
import
org.springframework.data.repository.query.Pa
ram;

import java.util.List;

public interface EnglishWordRepository
extends CrudRepository<Englishword, Long>
{
    @Query("SELECT w FROM Englishword
w WHERE w.word = :word")
    List<Englishword>
findByWord(@Param("word") String word);
}
```

```
package
sergey.shulga.uswitcherapp.repositories;
```

```
import jakarta.persistence.*;

@Entity
@Table(indexes = {@Index(name =
"idx_word", columnList = "word")})
public class Ukrainianword {

   @Id
   private Long id;
   private String word;

   public String getWord() {
      return word;
   }

   public void setWord(String word) {
      this.word = word;
   }

   public void setId(Long id) {
      this.id = id;
   }

   public Long getId() {
      return id;
   }
}
```

```
package
sergey.shulga.uswitcherapp.repositories;

import
org.springframework.data.jpa.repository.Quer
y;
import
org.springframework.data.repository.CrudRep
ository;
import
org.springframework.data.repository.query.Pa
ram;

import java.util.List;

public interface UkrainianWordRepository
extends CrudRepository<Ukrainianword,
Long> {
   @Query("SELECT w FROM
Ukrainianword w WHERE w.word = :word")
   List<Ukrainianword>
findByWord(@Param("word") String word);
}
```

```
package sergey.shulga.uswitcherapp.view;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import
org.springframework.beans.factory.annotation
.Autowired;
import
org.springframework.stereotype.Service;
import
sergey.shulga.uswitcherapp.contollers.Keybo
ardListener;

import javax.swing.*;
import javax.swing.border.EmptyBorder;
import java.awt.*;
import java.io.IOException;
import java.net.URI;
import java.net.URISyntaxException;

@Service
public class Frame {
   private final KeyboardListener
keyboardListener;

   @Autowired
   public Frame(KeyboardListener
keyboardListener) {
      this.keyboardListener =
keyboardListener;
   }

   private static final Logger logger =
LogManager.getLogger(Frame.class);
   private TrayIcon trayIcon;
   private static boolean isProgramRunning =
true;
   private boolean
isLanguageAlreadySelected = false;

   private static String selectedLanguage1 =
null;
   private static String selectedLanguage2 =
null;

   public static String getSelectedLanguage1()
{
      return selectedLanguage1;
   }

   public static String getSelectedLanguage2()
{
```

```
        return selectedLanguage2;
    }

    private String[] availableLanguages;

    public void setAvailableLanguages(String[]
availableLanguages) {
        this.availableLanguages =
availableLanguages;
        createTrayWindow();
    }

    private void createTrayWindow() {
        if (!SystemTray.isSupported()) {
            System.out.println("SystemTray is not
supported on this platform.");
            return;
        }

        // Створюємо випадне меню для трея
        PopupMenu popup = new
PopupMenu("Tray menu");

        MenuItem buttonToSelectLanguages =
getButtonToSelectLanguages();
        popup.add(buttonToSelectLanguages);

        // LAUNCH HERE
        MenuItem buttonLaunch = new
MenuItem("Запуск програми");
        buttonLaunch.addActionListener(e -> {
            if (!isLanguageAlreadySelected) {

JOptionPane.showMessageDialog(null, "Для
початку роботи оберіть дві мови");
            } else {
                System.out.println("Program has
been started");
                setRunningProgram(true);

                keyboardListener.mainListener();
            }
        });
        popup.add(buttonLaunch);

        MenuItem buttonPause = new
MenuItem("Пауза");
        buttonPause.addActionListener(e -> {
            if (!isProgramRunning)

JOptionPane.showMessageDialog(null,
"Робота програми вже призупинена");
            // Дії при натисканні кнопки
"Пауза"
            System.out.println("Program
paused");
            setRunningProgram(false);
        });
        popup.add(buttonPause);

        MenuItem buttonContinue = new
MenuItem("Продовжити");
        buttonContinue.addActionListener(e -> {
            if (isProgramRunning)

JOptionPane.showMessageDialog(null,
"Робота програми не була призупинена");
            // Дії при натисканні кнопки
"Продовжити"
            System.out.println("Program
resumed");
            setRunningProgram(true);
        });
        popup.add(buttonContinue);

        MenuItem buttonFeedback = new
MenuItem("Зв'язок з розробником");
        buttonFeedback.addActionListener(e ->
{
            try {
                Desktop.getDesktop().browse(new
URI("https://t.me/userfromworld"));
            } catch (IOException |
URISyntaxException ex) {
                logger.error("Error: " + ex);
            }
        });
        popup.add(buttonFeedback);

        MenuItem buttonInstructions = new
MenuItem("Інструкції з використання");
        buttonInstructions.addActionListener(e -
> {
            // Створюємо вікно з текстом
інструкцій
            JFrame instructionsFrame = new
JFrame("Інструкції");
            instructionsFrame.setSize(400, 300);

instructionsFrame.setDefaultCloseOperation(J
Frame.DISPOSE_ON_CLOSE);

instructionsFrame.setLocationRelativeTo(null
);
```

```
        instructionsFrame.setResizable(false);

        JTextArea instructionsText = new
JTextArea();
        instructionsText.setFont(new
Font("Verdana", Font.PLAIN, 14));
        instructionsText.setEditable(false);
        instructionsText.setLineWrap(true);

instructionsText.setWrapStyleWord(true);
        instructionsText.setText("""
                Для запуску програми
натисніть кнопку "Запуск"

                Якщо Вам потрібно на певний
час зупинити програму, натисніть кнопку
"Пауза"

                Якщо Вам потрібно відновити
роботу програми, натисніть кнопку
"Продовжити"

                Для виходу з програми
натисніть кнопку "Вихід"

                Дякую за використання
uSwitcher. Все буде Україна!""");

        instructionsText.setBorder(new
EmptyBorder(10, 10, 10, 10));


instructionsFrame.add(instructionsText);
        instructionsFrame.setVisible(true);
    });
    popup.add(buttonInstructions);

    MenuItem buttonExit = new
MenuItem("Вихід");
    buttonExit.addActionListener(e -> {

SystemTray.getSystemTray().remove(trayIco
n);
        System.exit(0);
    });
    popup.add(buttonExit);

    Image trapPicture =
Toolkit.getDefaultToolkit().getImage("uSwitc
herLogo.png");

    // Створюємо трей
    SystemTray tray =
SystemTray.getSystemTray();
    trayIcon = new TrayIcon(trapPicture,
"uSwitcher", popup);

    try {
        tray.add(trayIcon);
    } catch (AWTException e) {
        logger.error("Error: " + e);
    }
}

  private MenuItem
getButtonToSelectLanguages() {
    MenuItem buttonToSelectLanguages =
new MenuItem("Обрати мови для зміни");

buttonToSelectLanguages.addActionListener(
e -> {

        JFrame frame = new
JFrame("Доступні мови");
        frame.setSize(300, 200);

frame.setDefaultCloseOperation(JFrame.DIS
POSE_ON_CLOSE);
        frame.setLocationRelativeTo(null);
        frame.setResizable(false);

        JPanel panel = new JPanel();
        panel.setLayout(new
GridBagLayout());
        GridBagConstraints gbc = new
GridBagConstraints();
        gbc.anchor =
GridBagConstraints.CENTER;
        gbc.insets = new Insets(5, 5, 5, 5);

        Font font = new Font("Tahoma",
Font.PLAIN, 14);

        final int[] selectedCount = {0};

        for (String language :
availableLanguages) {
            JCheckBox checkBox = new
JCheckBox(language);
            checkBox.setFont(font);
            gbc.gridy++;
            panel.add(checkBox, gbc);
```

```
            checkBox.addActionListener(e1 ->
{
                if (checkBox.isSelected()) {
                    if (selectedCount[0] == 0)
                        selectedLanguage1 =
checkBox.getText();
                    else if (selectedCount[0] == 1)
                        selectedLanguage2 =
checkBox.getText();

                    selectedCount[0]++;
                } else {
                    if (selectedCount[0] == 1 &&
checkBox.getText().equals(selectedLanguage
1))
                        selectedLanguage1 = null;
                    else if (selectedCount[0] == 1
&&
checkBox.getText().equals(selectedLanguage
2))
                        selectedLanguage2 = null;
                    else if (selectedCount[0] == 2
&&
checkBox.getText().equals(selectedLanguage
1)) {
                        selectedLanguage1 =
selectedLanguage2;
                        selectedLanguage2 = null;
                    }
                    selectedCount[0]--;
                }
            });
        }

        JButton okButton = new
JButton("Зберегти");
        okButton.setFont(font);
        okButton.addActionListener(e12 -> {
            frame.dispose();
            if (selectedCount[0] != 2) {

JOptionPane.showMessageDialog(null,
"Будь-ласка, оберіть дві мови");
            } else if
(isLanguageAlreadySelected) {

JOptionPane.showMessageDialog(null, "Ви
вже обрали мови");
            } else {

JOptionPane.showMessageDialog(null,
"Обрані мови: " + selectedLanguage1 + ", "
+ selectedLanguage2);
                System.out.println("Selected
Language 1: " + selectedLanguage1);
                System.out.println("Selected
Language 2: " + selectedLanguage2);
                isLanguageAlreadySelected =
true;
            }
        });

        gbc.gridy++;
        panel.add(okButton, gbc);

        frame.add(panel);
        frame.setVisible(true);
    });
    return buttonToSelectLanguages;
  }

  public void setRunningProgram(boolean
runningProgram) {
    Frame.isProgramRunning =
runningProgram;
  }

  public static boolean isProgramRunning() {
    return isProgramRunning;
  }
}
```

```
package sergey.shulga.uswitcherapp;

import
org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.Spri
ngBootApplication;

@SpringBootApplication
public class USwitcherAppApplication {
  public static void main(String[] args) {

SpringApplication.run(USwitcherAppApplica
tion.class, args);
  }
}
```