Міністерство освіти і науки України
Криворізький національний університет
Кафедра моделювання та програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти бакалавра

зі спеціальності 121 – Інженерія програмного забезпечення

На тему: Розробка утіліти для автоматичного визначення коректної розкладки клавіатури на основі користувацького введення

Засвідчую, що в цій кваліфікаційній роботі немає запозичень із праць інших авторів без відповідних посилань.
Студент гр. ЗІПЗ–21
_____/ В. В. Пирєу /

Керівник кваліфікаційної роботи _____/ Д. В. Швець /

Завідувач кафедри _____/ А. М. Стрюк /

Кривий Ріг
2025

Криворізький національний університет

Факультет: Інформаційних технологій

Кафедра: Моделювання та програмного забезпечення

Ступінь вищої освіти: бакалавр

Спеціальність: 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедри
_____ А. М. Стрюк
«____» _____ 2025 р.

ЗАВДАННЯ

на кваліфікаційну роботу

студенту групи ЗІПЗ–21 Пирею Владиславу Вікторовичу

1. Тема: «Розробка утіліти для автоматичного визначення коректної розкладки клавіатури на основі користувацького введення» затверджена наказом по КНУ № 203с від «10» квітня 2025 р.

2. Термін подання студентом закінченої роботи: «31» травня 2025 р.

3. Вихідні дані по роботі: розроблений програмний комплекс має забезпечити активацію правильної розкладки клавіатури на основі користувацького введення.

4. Зміст пояснювальної записки (перелік питань, що їх треба розробити): провести аналіз існуючих на ринку аналогічних програмних продуктів, обґрунтувати функціонал розроблюваної системи, створити програмне забезпечення, здійснити тестування розробленого додатку.

5. Перелік ілюстративного матеріалу: функціональна схема, блок–схема алгоритму, зображення екранних форм додатку.

Календарний план:

№	Найменування етапів кваліфікаційної роботи	Термін виконання етапів роботи
1	Розгляд літературних джерел та пошук інтернет-ресурсів з заданої тематики	10.01.25 – 24.01.25
2	Аналіз існуючих методів вирішення проблеми	25.01.25 – 07.02.25
3	Формулювання актуальності роботи і постановка завдань	08.02.25 – 18.02.25
4	Оформлення матеріалів першого розділу роботи	18.02.25 – 02.03.25
5	Створення функціональної системи та алгоритму додатку	03.03.25 – 15.03.25
6	Оформлення матеріалів другого розділу роботи	16.04.25 – 09.04.25
7	Розробка баз даних, інтерфейсу програмного забезпечення, програмних модулів	10.04.25 – 01.05.25
8	Оформлення додатків	02.05.25 – 07.05.25
9	Тестування розробленої програми	08.05.25 – 15.05.25
10	Оформлення пояснювальної записки	16.05.25 – 30.05.25

Дата видачі завдання: «09» січня 2025 р.

Студент: _____ / В. В. Пирею /

Керівник роботи: _____ / Д. В. Швець /

РЕФЕРАТ

РОЗКЛАДКА КЛАВІАТУРИ, ВВЕДЕННЯ, МУЛЬТИМОВНІСТЬ, МУЛЬТИЛІНГВАЛЬНІСТЬ, ПЕРЕКЛАДКА, МАРШАЛІНГ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, УТІЛІТА.

Пояснювальна записка: 66 с., 15 рис., 1 табл., 3 дод., 23 джерела.

Мета кваліфікаційної роботи: розробка утіліти для автоматичного визначення коректної розкладки клавіатури на основі користувацького введення.

Об’єкт проектування: утіліта для автоматичного визначення коректної розкладки клавіатури на основі користувацького введення.

У теоретичній частині роботи виконано аналіз існуючих на сьогодні аналогічних програмних рішень на ринку. Зазначені сильні та слабкі сторони існуючих програмних продуктів. Обґрунтовані актуальність роботи, мета та сформульовані завдання для розробки зазначеної системи.

У практичній частині кваліфікаційної роботи реалізовано функціональну схему розроблюваного продукту та алгоритм його роботи. Розроблено інтерфейс програмного продукту, програмну логіку роботи додатку. Проведено тестування розробленого програмного забезпечення.

Розроблена утіліта для автоматичного визначення коректної розкладки клавіатури на основі користувацького введення може використовуватись широким колом користувачів.

ABSTRACT

KEYBOARD LAYOUT, INPUT, MULTILINGUAL, MULTILINGUALISM, MAPPING, MARSHALING, SOFTWARE, UTILITY.

Explanatory note: 66 p., 15 fig., 1 tabl., 3 app., 23 references.

The aim of the qualifying work: to develop a utility for automatically determining the correct keyboard layout based on user input.

Design object: a utility for automatically determining the correct keyboard layout based on user input.

In the theoretical part of the work, an analysis of today's similar software decisions in the market have executed. The strengths and weaknesses of existing software products have listed. The urgency of work, the purpose, and the objectives for developing the specified system have formulated.

The functional scheme of the developed product and its algorithm has realized in the practical part of the qualification work. The interface of the software product, the program logic of the application have developed. The developed software has tested.

The developed utility for automatically determining the correct keyboard layout based on user input can be used by a wide range of users.

ЗМІСТ

ВСТУП

В кваліфікаційній роботі розглядається питання розв’язання завдання автоматичного виправлення користувацького введення в мультимовному середовищі.

Для користувачів з країн, в який англійська мова не є єдиною вживаною, постає актуальність питання використання декількох розкладок клавіатури в операційних системах (наприклад, англійської та української). Цей факт зумовлює появу ситуацій, коли користувач забуває перемкнути розкладку на необхідну, і введення тексту стає некоректним, що зумовлює втрати часу на необхідність видалення вже надрукованого фрагменту та втрат часу на повторний набір.

Це зумовлює необхідність розробки програмних засобів для автоматичного визначення мови користувацького введення та автоматичного виправлення тексту, введеного в некоректній розкладці.

В кваліфікаційній роботі проведено огляд часових витрат на введення тексту в неправильній розкладці клавіатури, розглядається вплив неправильного вибору мовної розкладки на ефективність введення, швидкість друку, когнітивне навантаження і загальну продуктивність користувача. Проведено аналіз літератури з питань помилок введення, витрат на корекцію, та впливу зусиль, що виникають в наслідок зазначеної проблеми, а також розглядаються перспективи створення систем для розв’язання проблеми виникнення помилок при мультимовному введенні текстів.

В ході досліджень було спроектовано утіліту для автоматичного визначення коректної розкладки клавіатури на основі користувацького введення, яка має зберегти час та зусилля користувачів під час виконання текстового набору та виконанні інших повсякденних справ під час роботи на комп’ютері.

1 АНАЛІЗ ПИТАННЯ КОРЕКЦІЇ ВВЕДЕННЯ ТЕКСТУ У МУЛЬТИМОВНИХ СЕРЕДОВИЩАХ

1.1 Актуальність питання автоматичного визначення коректної розкладки клавіатури на основі користувацького введення

Як добре відомо, в операційних системах реалізована можливість перемикатися між кількома мовними розкладками клавіатури, що є необхідним для користувачів, які працюють у багатомовному середовищі. При цьому користувачі часто стикаються з ситуаціями, коли замість очікуваної розкладки (наприклад, української) активною виявляється англійська, що призводить до введення набору символів, які не відповідають їх намірам, і вимагає додаткових витрат часу на виправлення помилок. Такі ситуації впливають на продуктивність роботи завдяки збільшенню часу, витраченого на корекцію, і підвищенню напруження, оскільки користувачеві доводиться постійно контролювати відповідність тексту, що вводиться, очікуваній розкладці.

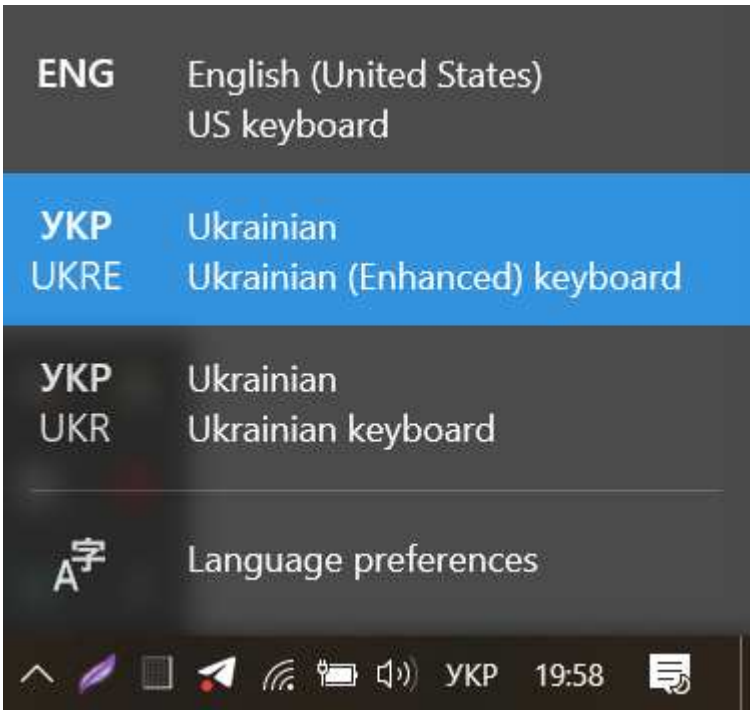

Рисунок 1.1 – Перемикання розкладки клавіатури в Windows 10

Таким чином, у багатомовних інформаційних середовищах користувачі часто змушені перемикатися між декількома розкладками, що додає складності при роботі з клавіатурою. Під час перемикання між розкладками змінюється не тільки розташування символів, а й специфічні особливості автокорекційних алгоритмів, що грунтуються на певних мовних правилах (наприклад, автокорекція у Microsoft Word). Наприклад, система автозаміни, налаштована для англійської мови, буде недостатньо ефективною при спробі коригувати введення, виконане в українській розкладці, і навпаки. Такі невідповідності призводять до збільшення ситуацій, коли неправильна розкладка призводить до введення безглуздого набору символів.

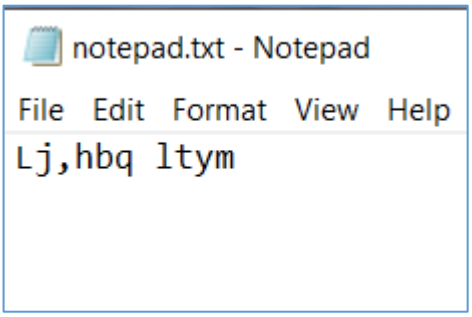

Рисунок 1.2 – Користувач писав «Добрий день», але забув перемикнути розкладку

Крім того, часте перемикання між мовами збільшує ймовірність того, що користувач випадково продовжить набір тексту в неправильній розкладці, особливо якщо перемикання відбувається автоматично або за допомогою гарячих клавіш, які можуть бути натиснуті випадково в умовах поспіху або втоми. Така поведінка спричиняє ланцюжок помилок, оскільки невірно введений текст додатково ускладнює використання інтелектуальних систем і алгоритмів автокорекції, що призводить до ще більшої витрати часу на виправлення помилок.

Таким чином, питання розробки програмного забезпечення для автоматичного визначення коректної розкладки клавіатури на основі

користувацького введення є актуальним та важливим завданням, яке може допомогти зекномити велику кількість часу для всіх користувачів персональних комп’ютерів, які працюють у мультимовному середовищі.

1.2 Огляд досліджень з питання проблематики мультимовного введення

Дослідження демонструють, що користувачі, які звикли до конкретної мовної розкладки, мають вищі показники продуктивності введення тексту, тоді як зміна звичної розкладки спричиняє суттєве збільшення помилок і часу на корекцію [1]. У багатомовних сценаріях подібна ситуація особливо критична, тому що різниця між розкладками може бути значною та призвести до значного зниження швидкості введення, особливо в тих користувачів, чий робочий процес ґрунтується на швидкому коригуванні та безперервному потоці інформації [2].

Крім безпосередньо збільшених витрат часу на корекцію, неправильна розкладка клавіатури впливає на загальну якість роботи і задоволеність користувача. Користувачі, які регулярно стикаються з необхідністю коригувати помилки, відчувають підвищений рівень стресу, що призводить до зниження загальної продуктивності. Додаткові витрати часу, що накопичуються протягом робочого дня, можуть істотно знизити продуктивність робочих процесів, які потребують постійного введення текстової інформації [3].

Проблема невірної розкладки також призводить до додаткових когнітивних витрат користувачей, що пов'язані з необхідністю постійного контролю за коректністю введення. Користувачеві потрібно витрачати час на перевірку кожного введеного символу, що призводить до порушення звичного ритму роботи і зниження швидкості набору тексту. Такі завдання, як редагування, коректура або огляд введеного тексту, стають менш ефективними, оскільки увага постійно відволікається на виправлення помилок, спричинених невірною розкладкою [3]. У результаті порушується

безперервність робочого процесу, що має негативний вплив на загальну продуктивність і може призводити до зниження якості виконаної роботи.

Додатковим фактором є вплив неправильної розкладки на використання автоматичних систем введення, таких як автокорекція та передбачення тексту. Коли система автокорекції не може правильно інтерпретувати введення, вона не встигає ефективно скоригувати помилки, а користувачеві доводиться вручну відкочуватися назад і передруковувати текст, що ще більше збільшує витрати часу. Зі свого боку, алгоритми передбачення (наприклад, автодоповнення у Visual Studio чи інших редакторах) часто ґрунтуються на статистиці мови, що призводить до введення невідповідних слів під час набору на іншій розкладці [4].

Крім того, дослідники вказують на те, що помилки введення внаслідок неправильної розкладки можуть негативно впливати на впевненість користувача у власних силах, що призводить до додаткової затримки під час набору тексту та погіршення загального робочого настрою. Такі ефекти особливо помітні серед менш досвідчених користувачів, для яких проблеми з перемиканням розкладок призводять до високих показників помилки та необхідності частого виправлення тексту, що вводиться [5].

Безперервні витрати часу та розумові зусилля, пов'язані з виправленням помилок, спричинених введенням тексту в неправильній розкладці, можуть призводити до значного зниження продуктивності в довгостроковій перспективі. На робочих місцях, де потрібне регулярне введення великих обсягів тексту, навіть невелике збільшення часу на одне виправлення сумарно призводить до втрати значних ресурсів. Це не тільки знижує загальну швидкість виконання завдань, а й впливає на рівень задоволеності користувачів, що може призвести до погіршення якості роботи та підвищеного ризику втоми.

В умовах офісів і коворкінгів, цей ефект суттєво посилюється. Користувач, змушений повторно виправляти помилки, втрачає концентрацію і фокус, що може призводити до систематичних затримок у виконанні завдань,

які потребують швидкого введення інформації. Такі затримки можуть виявитися критичними в динамічних робочих процесах, де час реакції є важливим фактором успіху. Організаційні дослідження показали, що зниження продуктивності введення безпосередньо корелює зі зростанням когнітивного навантаження, зумовленого необхідністю постійної самокорекції [3].

1.3 Розгляд метрик для виміру затрат часу на введення тексту та підрахунку моторних помилок

Одним із способів кількісної оцінки помилок під час введення тексту є використання показника кількості натискань клавіш на символ (Keystrokes Per Character, KSPC) [6,7]. Цей показник є простим відношенням кількості введених символів (включно з пробілами) до кінцевої кількості символів у транскрибованому рядку. Формула для обчислення KSPC має вигляд:

$$\text{KSPC} = \frac{\sum(K_c * F_c)}{\sum(C_c * F_c)} \tag{1.1}$$

де *Kc* - кількість натискань клавіш, необхідних для введення символу;

Cc = 1 («розмір» кожного символу);

Fc - частота появи символу.

Ще одним показником є Total Error Rate (TER), що об’єднує усі види помилок (як виправлені, так і невиправлені) [8].

$$TER = \frac{V + N}{C} \tag{1.2}$$

де *V* – виправлені помилки;

N - невиправлені помилки:

C – загальна кількість помилок.

Неправильний вибір розкладки призводить до систематичних помилок введення, таких як заміна символів, пропуски або вставлення зайвих знаків, що відображається в підвищених значеннях метрик KSPC і TER.

Кількість слів за хвилину (WPM) є одним з найбільш поширених емпіричних показників швидкості введення тексту. З початку двадцятого сторіччя прийнято вважати середню довжину слова за 5 символів, включаючи пробіли (принайні, англійського) [9]. Важливо, що показник WPM не враховує кількість натискань клавіш або жестів, зроблених під час введення, а лише довжину отриманого транскрибованого рядка і час, необхідний для його створення. Таким чином, формула для обчислення WPM має вигляд:

$$WPM = \frac{|T-1|}{S} * 60 * \frac{1}{5} \quad (1.3)$$

де T – кінцевий транскрибований рядок;

$|T|$ - довжина цього рядка;

60 – кількість секунд в хвилині;

1/5 – відношення довжини слова до символу.

Зазначається, що T може містити літери, цифри, розділові знаки, пробіли та інші друковані символи, але не пробіли. Таким чином, T фіксує не процес введення тексту, а лише його результат.

Під час перемикання між розкладками клавіатури користувач змушений адаптувати свої моторні навички та переглядати ментальні карти розташування символів. Цей процес потребує значних когнітивних ресурсів, що призводить до збільшення часу на введення та помилкової корекції. Збільшення когнітивного навантаження безпосередньо пов'язане зі збільшенням часу, що витрачається на виправлення помилок, а також із підвищеним рівнем розчарування в користувача [10]. Ба більше, необхідність постійного контролю символів, що вводяться, і увага до корекції помилок

порушують природний ритм друку, знижуючи показники WPM і збільшуючи KSPS за рахунок додаткових натискань клавіш для виправлення [11].

Зазначені формули розглядали точність під час введення тексту, але як щодо точності отриманого транскрибованого рядка? Для цього можна використати статистику мінімальної відстані між рядками [12,13]. Статистика MSD дає «відстань» між двома рядками з точки зору найменшої кількості операцій виправлення помилок, необхідних для перетворення одного рядка в інший. Доступні операції виправлення помилок - це вставка символу, видалення (або пропуск) символу та заміна символу [14-16]. Обчислити частоту помилок мінімальної редакційної відстані (Minimum String Distance – MSD) можна за наступним рівнянням [6]:

$$MSD\ error\ rate\ = \frac{MSD(P,T)}{\max(|P|,|T|)} \tag{1.4}$$

де P – подані рядки;

T – транскрибовані рядки;

Метрики продуктивності введення тексту, такі як WPM, KSPC, MSD і TER, дають змогу кількісно оцінити вплив вибору неправильної розкладки клавіатури на роботу користувача. Аналіз даних показує, що за використанням неправильної розкладки спостерігається значне збільшення кількості натискань клавіш для отримання правильного результату, що свідчить про збільшення витрат часу на введення тексту [3]. Підвищений показник KSPC означає, що користувач змушений виконувати додаткові дії порівняно з оптимальним введенням, що вказує на неефективність поточної системи введення.

Крім того, збільшення TER і кількості невірних символів свідчить про те, що помилки введення потребують тривалішого часу на коригування, оскільки користувач витрачає додаткові секунди на виправлення кожного

окремого символу або слова. Такі витрати часу особливо відчутні під час набору великих обсягів тексту, де накопичення дрібних затримок може призвести до значних втрат загальної продуктивності. Високі показники помилок також можуть спричинити додаткову моральну втому в користувача, що знижує загальну задоволеність процесом введення та знижує працездатність під час виконання завдань.

Для аналізу впливу неправильної розкладки клавіатури на витрати часу дослідники застосовують різні експериментальні методики, що включають вимірювання WPM, KSPC та інших показників помилкового введення. Експериментальні дослідження, проведені за участю досвідчених і не дуже користувачів, підтвердили, що набір тексту в неправильній розкладці характеризується збільшенням загальної кількості натискань клавіш для досягнення необхідної точності, а також збільшенням часу на корекцію помилок [17]. Крім того, проведення експериментів у контрольованих умовах дало змогу об'єктивно оцінити відмінності в продуктивності під час використання різних розкладок, а також виявити залежність між розумовим навантаженням користувача і числом допущених помилок [11].

В одному з експериментів, проведених на стандартній QWERTY-клавіатурі, користувачам було запропоновано вводити англійські фрази за умов, коли розкладка відповідала їхньому досвіду, і порівняти результати із введенням тексту за неправильної розкладки, що дало змогу порівняти показники швидкості введення та кількість необхідних корекційних натискань. Результати показали, що при неправильній розкладці спостерігається значне зниження швидкості введення, а також збільшення показників, таких як KSPC, що вказує на необхідність додаткового набору і коригування тексту. Додатково, дослідження в галузі вивчення характерів помилок показують, що неправильна розкладка спричиняє систематичні заміщення символів, що вимагає використання різних методів корекції, які збільшують загальну витрату часу [18].

Експериментальні моделі, що враховують взаємодію моторних і когнітивних аспектів під час набору тексту, показують, що звичне розташування клавіш дає змогу мінімізувати міжклавішні інтервали й оптимізувати моторику рук користувача, тоді як перемикання на нову, незнайому розкладку вимагає перерозподілу моторних патернів і візуального зворотного зв'язку, що додатково збільшує когнітивне навантаження і час виконання завдань [19, 20].

Помилки, що виникають під час використання неправильної розкладки, часто не просто випадкові, а пов'язані з тим, що користувач, який звик до однієї мовної системи, продовжує використовувати усталені моторні патерни, не адаптувавши їх до чужого набору символів. Така ситуація призводить до розбіжності між наміром користувача і фактичним результатом уведення, що вимагає подальшої корекції та додаткових часових витрат.

1.4 Огляд результатів аналізу існуючого програмного забезпечення та висновків по огляду наукових джерел

Як показав проведений аналіз програм аналогів для автоматичного визначення коректної розкладки клавіатури на основі користувацького введення, такі рішення не є розповсюдженими. В першу чергу це стосується того, що велика кількість користувачі великої кількості англомовних країн просто не мають потреби перемикатися з англійської на інші клавіатурні розкладки. Тому в західних країнах ця ідея розповсюдження не отримала. Існують аналоги від розробників з підсанкційних країн колишнього СНД, але, в першу чергу, їх використання викликає безпекові питання, а то й взагалі заборонено законодавством. Таким чином можна зробити виновок, що реалізованих програмних аналогів для запропонованої ідеї на сьогодні у відкритому доступі не існує.

Проблема витрат часу на введення тексту в неправильній розкладці клавіатури є багатоаспектною і впливає як на кількісні показники продуктивності (WPM, KSPC, TER, MSD), так і на суб'єктивне сприйняття

користувача. Невірний вибір розкладки клавіатури призводить до збільшення кількості помилок, додаткових корекційних операцій, підвищеного когнітивного навантаження і, як наслідок, зниження загальної ефективності робочого процесу. В умовах багатомовних середовищ, де користувачі регулярно перемикаються між англійською та українською розкладками, проблема загострюється через додатковий дисонанс між звичками введення та автоматичними алгоритмами автокорекції.

В той же час, неправильна розкладка призводить не тільки до зниження швидкості набору тексту, а й також спричиняє збільшення часу на виправлення помилок через додаткові моторним і когнітивним затрати. Для розв'язання цієї проблеми пропонуються заходи, спрямовані на поліпшення індикації активної розкладки, адаптацію автокорекції до неправильного введення, а також на створення інтуїтивно зрозуміліших механізмів перемикання мов, що дасть змогу зменшити втрати часу і поліпшити загальну якість введення.

Таким чином, для підвищення ефективності текстового введення в багатомовних системах необхідно приділяти значну увагу оптимізації процесів перемикання між розкладками, а також удосконаленню алгоритмів інтелектуального введення, що дасть змогу зменшити витрати часу і навантаження користувачів. Майбутні розробки в галузі адаптивних інтерфейсів являють собою перспективний напрямок для усунення цих проблем і підвищення продуктивності під час набору тексту.

На закінчення, проблема витрат часу на введення тексту в неправильній розкладці клавіатури є критично важливою для підвищення загальної ефективності роботи користувачів в умовах мультилінгвальних середовищ. Розв'язання цієї проблеми вимагає комплексного підходу, що включає поліпшення апаратних засобів, інтерфейсних рішень та інтелектуальних алгоритмів обробки введення. Подальші дослідження в цій царині дадуть змогу розробити більш ефективні системи, здатні динамічно адаптуватися до

потреб користувача та мінімізувати нервове напруження, пов'язане з помилковим введенням тексту.

Можна стверджувати, що ефективне керування розкладками клавіатури та мінімізація помилок введення є одними з ключових чинників забезпечення високої продуктивності та якості роботи в умовах, коли користувачі змушені працювати з кількома мовними системами. Застосування вищеописаних підходів може значно знизити загальні витрати часу на виправлення помилок, підвищити задоволеність користувачів і забезпечити більш гладкий робочий процес у мультимовних інформаційних системах.

Отже, розв'язання проблеми витрат часу на введення тексту в неправильній розкладці клавіатури потребує всебічного дослідження, що включає як кількісну оцінку помилок, так і якісний аналіз когнітивних аспектів. Інтеграція адаптивних технологій, поліпшених інтерфейсних рішень та інтелектуальних алгоритмів автокорекції являє собою оптимальний шлях для подолання цієї проблеми і підвищення ефективності роботи користувачів в умовах багатомовності.

У підсумку, подальші дослідження в цій царині можуть сфокусуватися на розробленні:

а) інтуїтивних і наочних механізмів перемикання між мовними розкладками для мінімізації помилок;

б) інтеграції методів машинного навчання для аналізу користувацького введення, що дають змогу накопичувати дані та передбачати потенційні помилки з метою автоматичного виправлення;

в) поліпшення тактильного і візуального зворотного зв'язку в системах введення, що дасть змогу знизити когнітивне навантаження під час роботи з кількома розкладками.

г) адаптивних алгоритмів автокорекції, здатних враховувати особливості введення в неправильній розкладці й ефективно пропонувати коректні варіанти.

В зазначеній дипломній роботі буде виконано дослідження останнього пункту шляхом розробки алгоритму автокорекції користувацького введення та його програмної реалізації.

2 ПРОЕКТУВАННЯ УТІЛІТИ ДЛЯ АВТОМАТИЧНОГО ВИЗНАЧЕННЯ КОРЕКТНОЇ РОЗКЛАДКИ КЛАВІАТУРИ НА ОСНОВІ КОРИСТУВАЦЬКОГО ВВЕДЕННЯ

2.1 Опис функціоналу програмного забезпечення для автоматичного визначення коректної розкладки клавіатури на основі користувацького введення

Як зазначалося вище, для користувачів, які вводять текст кількома мовами, існує проблема випадкового використання неправильної розкладки клавіатури. Наприклад, у користувача встановлені в операційній системі англійська та українська розкладки клавіатури. Користувач починає вводити текст українською мовою, при цьому в нього активовано англійську розкладку клавіатури. У результаті він вводить безглуздий набір букв латиницею замість потрібного слова кирилицею.

Тож під час введення користувачем тексту утиліті, що розробляється, необхідно з'ясувати, чи в правильній розкладці введено текст. Для цього потрібно реалізувати таку послідовність дій утіліти:

а) Програма після запуску визначає активну розкладку клавіатури в операційній системі.

б) Користувач вводить слово і пробіл після нього.

в) Програма здійснює пошук скопійованого слова у відповідному активній розкладці словнику (якщо активна українська розкладка - то в ukrainian.txt, якщо англійська - то в english.txt).

г) Якщо введене слово знайдено в словнику, то жодних дій робити не потрібно й очікувати введення користувачем наступного слова.

д) Якщо введене слово НЕ знайдено в словнику, що відповідає активній розкладці, то необхідно здійснити перекладку. Під перекладанням мається на увазі зміна символів введеного слова відповідно до розташування латинських і кириличних символів на QWERTY клавіатурі (наприклад, при введенні

«ghbdsn» результатом перекладання буде «привіт», при введенні «цщкл» - «work»).

е) Після здійснення перекладки потрібно виконати пошук перетвореного слова в другому словнику (тобто такому, що не відповідає поточній розкладці).

ж) Якщо слово, отримане в результаті перекладки, було знайдено в другому словнику, то потрібно програмно переключити розкладку клавіатури в операційній системі, щоб вона відповідала мові другого словника, потім у рядку введення користувача видалити введене ним слово, і замість нього вставити слово, отримане в результаті перекладки. Якщо ж слово, отримане в результаті перекладки, НЕ було знайдено в другому словнику, то не робити жодних дій (залишити розкладку і введене користувачем слово без змін).

з) Програма очікує введення наступного слова від користувача.

Описаний текстовий алгоритм демонструє основні функціональні можливості програмного забезпечення для автоматичного визначення коректної розкладки клавіатури на основі користувацького введення.

2.2 Функціональна схема утіліти для автоматичного визначення коректної розкладки клавіатури на основі користувацького введення

Спроектоване програмне забезпечення описується функціональною схемою, наведеною на рисунку 2.1.

Система реалізує механізм обробки введення, який починається з інтеграції з Windows API через низькорівневий перехоплювач клавіатури. Цей підхід забезпечить перехоплення всіх подій введення незалежно від активної програми.

При введенні утиліта динамічно визначає межі слів, і при виявленні завершення введення слова ініціює процедуру перевірки його валідності. Цей етап передбачає звернення до текстових словників, що містять українські та англійські слова.

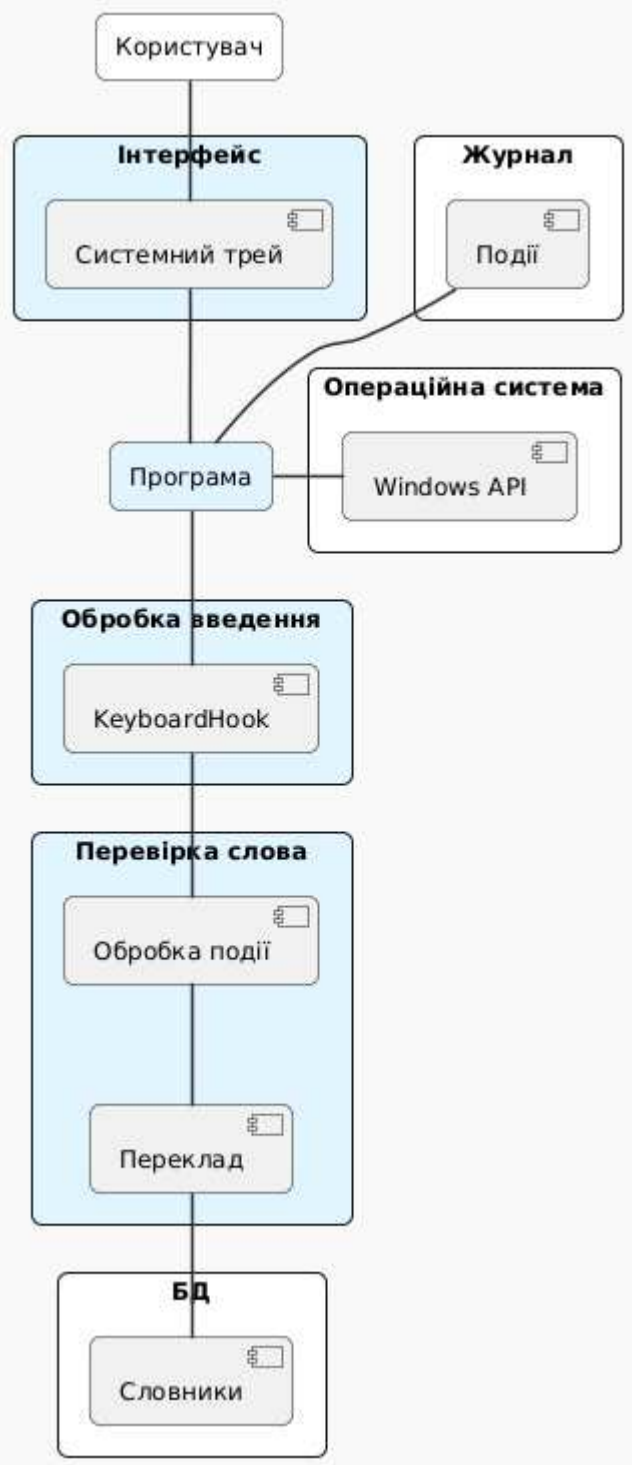

Рисунок 2.1 – Функціональна схема утіліти для автоматичного визначення коректної розкладки клавіатури

У випадку виявлення потенційної помилки розкладки програма застосовує алгоритм транслітерації, який враховує особливості відповідності символів між різними мовними розкладами. Цей алгоритм реалізований з

урахуванням підтримки різних регістрів символів, що забезпечує перекладку навіть з використанням заглавних літер.

Процес автоматичного виправлення помилок реалізований через генерацію послідовності програмних подій, які імітують дії користувача. Система спочатку видаляє неправильно введене слово, потім змінює мовну розкладку за допомогою відповідних системних викликів, після чого вводить виправлений варіант тексту.

Система включає механізм логування, який дозволяє відстежувати основні аспекти перевірки слова, його перекладки та, при необхідності, вставки виправленого замість початкового. Цей механізм фіксує як технічні події, пов'язані з роботою програми, так і дії користувача.

Програма не має візуального інтерфейсу у вигляді екранної форми, але використовує іконку в системному треї, яка дозволяє керувати роботою утиліти.

Зазначеної функціональності проектованого програмного забезпечення для автоматичного визначення коректної розкладки клавіатури на основі користувацького введення достатньо для розв’язання поставленого в роботі завдання.

2.3 Алгоритм функціонування утиліти для автоматичного визначення коректної розкладки клавіатури на основі користувацького введення

На рисунку 2.2 наведено алгоритм роботи програмного забезпечення для автоматичного визначення коректної розкладки клавіатури на основі користувацького введення.

Після запуску програма визначає, яка розкладка клавіатури наразі активна в операційній системі. Коли користувач вводить слово та натискає пробіл, програма перехоплює це слово та перевіряє його наявність у словнику, що відповідає активній розкладці (український або англійський словник).

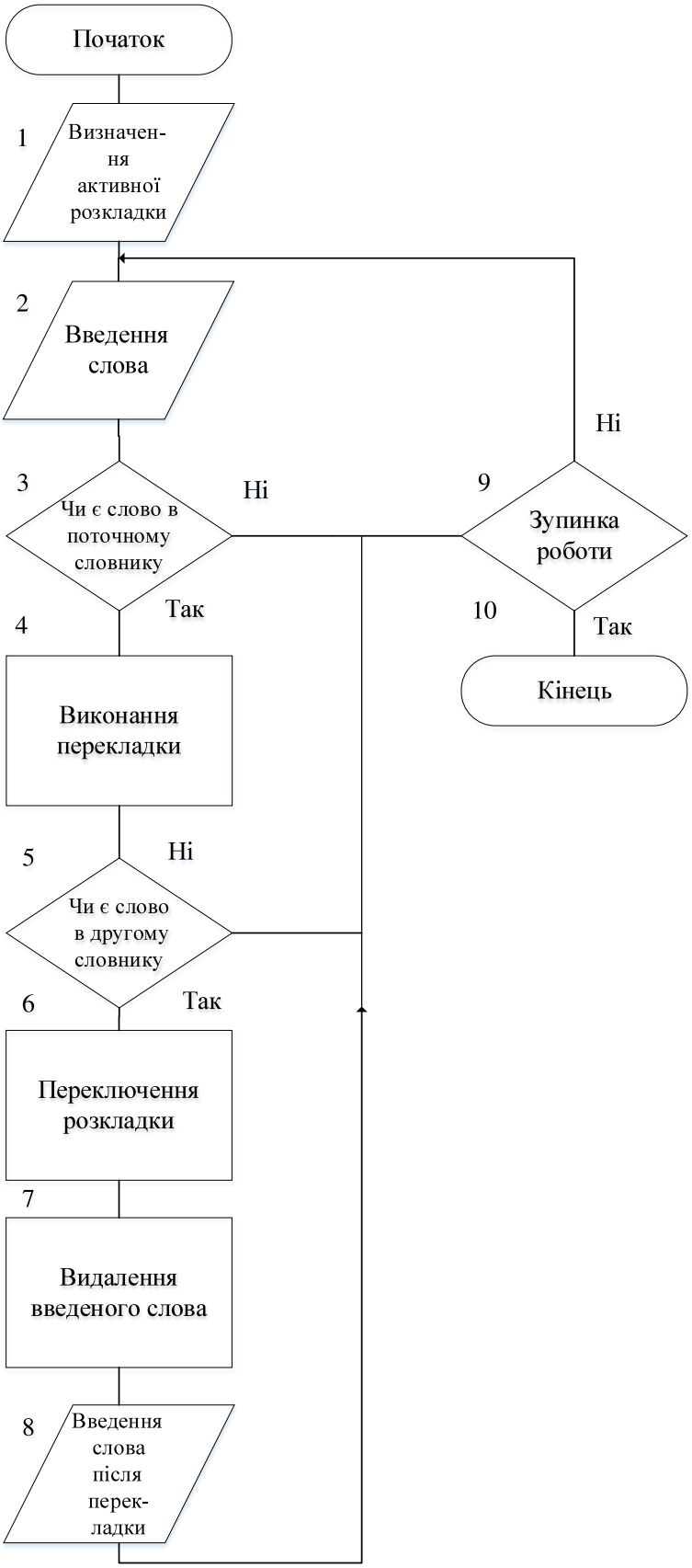

Рисунок 2.2 – Алгоритм роботи утиліти для автоматичного визначення коректної розкладки клавіатури на основі користувацького введення

Якщо слово присутнє у відповідному словнику, програма не виконує жодних дій. Вона реєструє, що слово введене правильно, і продовжує працювати в режимі очікування наступного слова. Це дозволяє не переривати процес набору тексту, якщо користувач використовує правильну розкладку і правильно набирає слова.

В іншому випадку, якщо слово у відповідному словнику не знайдено, програма припускає, що користувач міг помилково ввести слово в невірній розкладці. Утиліта виконує перекладання введеного слова — змінює символи згідно з розміщенням клавіш на QWERTY-клавіатурі між латиницею та кирилицею. Потім вона перевіряє наявність отриманого перекладеного слова в іншому словнику — тому, що відповідає іншій розкладці.

У разі, якщо перекладене слово наявне в другому словнику (у тому, який відповідає мові, відмінній від активної розкладки), програма автоматично змінює розкладку клавіатури на відповідну, видаляє з рядка введене користувачем слово й вставляє натомість перекладене.

У випадку, коли перекладене слово не знайдене в другому словнику, програма не вживає жодних дій і залишає все без змін. Далі вона знову переходить у режим очікування. Утиліта готова відреагувати на введення наступного слова, дотримуючись тієї ж логіки перевірки, перекладки та потенційного автоматичного виправлення.

Утиліта працює в циклічному режимі допоки користувач не завершить її роботу.

2.4 Вимоги до апаратного забезпечення для утиліти для автоматичного визначення коректної розкладки клавіатури на основі користувацького введення

У таблиці 2.1 приведені вимоги до апаратної складової для програмного комплексу автоматичного визначення коректної розкладки клавіатури на основі користувацького введення.

Таблиця 2.1 – Вимоги до апаратного забезпечення

Параметр	Вимоги
Центральний процесор	Intel i3 і вище, частота від 2 GHz
Операційна система	Windows 8 - Windows 11, 32-х та 64-х розрядні версії
Оперативна пам'ять	2 ГБ (для 32-розрядної версії Windows) або 4 ГБ (для 64-розрядної версії Windows)
Обсяг пам'яті на жорсткому диску	40 МБ та більше (розмір словників може збільшуватися при додаванні записів)
Пристрої введення	Клавіатура
Додаткові засоби для установки	CD-ROM або DVD-ROM для встановлення з CD або DVD-диску
Мережеві вимоги	Мережевий адаптер та доступ до Інтернет для завантаження та інсталяції
Порти USB	Наявність USB-портів для встановлення з USB-накопичувача

В таблиці наведені мінімальні системні вимоги до апаратного забезпечення з боку проектованої утиліти та інтерфейси для завантаження інсталяційних файлів.

2.5 Вимоги до засобів розробки утиліти для автоматичного визначення коректної розкладки клавіатури на основі користувацького введення

Створення програмного забезпечення вимагає виваженого підходу до вибору інструментів розробки. В даному підпункті приділено увагу вибору технологічної основи для реалізації утиліти. Після аналізу доступних варіантів було прийнято рішення використовувати платформу .NET у поєднанні з

мовою програмування C#, оскільки це рішення належним чином відповідає поставленим завданням.

.NET [21,22] - це відкрита та кросплатформна технологія, яка дозволяє створювати застосунки, що однаково вдало працюють на різних операційних системах. Така гнучкість важлива для проєктів, які передбачають широке розгортання або мають специфічні вимоги до середовища виконання. Крім того, ця платформа відзначається високою продуктивністю.

Мова програмування C# [23] доповнює .NET, адже вона не тільки потужна й багатофункціональна, а й достатньо зручна для розробників. Завдяки своїй об’єктно-орієнтованій природі, C# дозволяє писати чистий, зрозумілий і доволі структурований код. Це значно полегшує процес підтримки проєкту в майбутньому, а також сприяє командній роботі. До того ж, активна підтримка Microsoft і розвинене професійне середовище навколо цієї екосистеми гарантують стабільність, актуальність і швидке вирішення можливих труднощів.

Загалом, вибір на користь .NET та C# є не лише технічно виправданим, а й стратегічно продуманим кроком, який дозволяє реалізувати функціональний, масштабований і надійний програмний продукт.

3 РОЗРОБКА УТІЛІТИ ДЛЯ АВТОМАТИЧНОГО ВИЗНАЧЕННЯ КОРЕКТНОЇ РОЗКЛАДКИ КЛАВІАТУРИ НА ОСНОВІ КОРИСТУВАЦЬКОГО ВВЕДЕННЯ

3.1 Загальний принцип роботи розробленої програми

На основі результатів проектування було розроблено програмне забезпечення для автоматичного визначення коректної розкладки клавіатури на основі користувацького введення, діаграма класів якого наведено на рисунку 3.1.

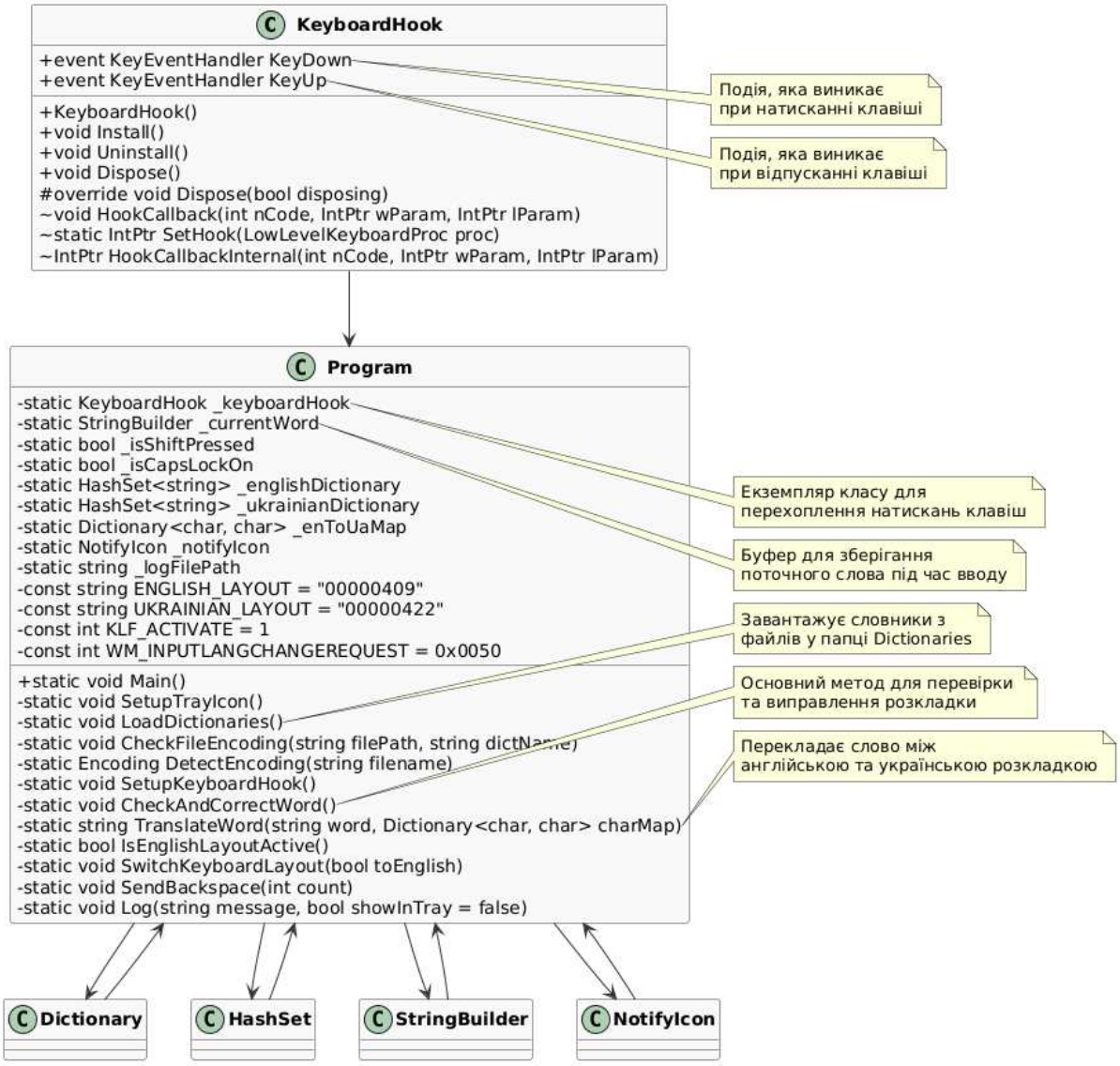

Рисунок 3.1 – Діаграма класів системи для автоматичного визначення коректної розкладки клавіатури

Основна логіка роботи утиліти реалізується класом Program.cs, який відповідає за координацію всіх її основних процесів. Він реалізує логіку автоматичного виправлення розкладки клавіатури, використовуючи низькорівневе перехоплення вводу та системні API.

Основа роботи класу будується навколо взаємодії з Windows API через оголошені зовнішні методи. Ці методи дозволяють отримувати інформацію про активне вікно, визначати поточну розкладку клавіатури, керувати станом клавіш-модифікаторів та змінювати мову вводу. Всі ці операції виконуються через механізм виклику платформи (P/Invoke), що забезпечує взаємодію з некерованим кодом операційної системи.

Клас використовує екземпляр KeyboardHook для перехоплення подій клавіатури. Під час ініціалізації створюється глобальний хук, який дозволяє відстежувати натискання клавіш у будь-якому вікні системи. Цей механізм дозволяє аналізувати введені символи та визначати необхідність корекції розкладки.

Для зберігання поточного стану введення використовуються приватні поля, такі як _currentWord для накопичення символів поточного слова, а також прапорці для відстеження стану клавіш Shift та Caps Lock. Ці дані оновлюються в реальному часі при обробці подій клавіатури.

Також реалізовано механізм визначення поточної мовної розкладки. Він базується на аналізі ідентифікатора розкладки клавіатури для активного потоку введення. Це дозволяє визначити, яка мова введення зараз активна, і прийняти рішення про необхідність перемикання розкладки.

При виявленні завершення введення слова відбувається перевірка його валідності в поточній розкладці. Якщо слово не знайдено в словнику поточної мови, система намагається знайти його в словнику іншої мови, використовуючи таблицю відповідності символів між розкладками.

Таблиця відповідності реалізована як статичний словник, що співставляє символи англійської розкладки з відповідними символами української. Вона включає всі літери англійського алфавіту у нижньому регістрі, містить

відповідності для цифр та спеціальних символів, враховує різні регістри та містить відповідності для спецсимволів, що використовуються з клавішею Shift.

Далі наведено фрагмент коду, що реалізує вищеописану таблицю відповідності.

```
// Мапінг між англійською та українською розкладкою
        private static readonly Dictionary<char, char>
_enToUaMap = new Dictionary<char, char>

        {
            {'q', 'й'}, {'w', 'ц'}, {'e', 'у'}, {'r', 'к'},
{'t', 'е'}, {'y', 'н'}, {'u', 'г'}, {'i', 'ш'}, {'o', 'щ'},
{'p', 'з'},
            {'[', 'х'}, {']', 'ї'}, {'a', 'ф'}, {'s', 'і'},
{'d', 'в'}, {'f', 'а'}, {'g', 'п'}, {'h', 'р'}, {'j', 'о'},
{'k', 'л'},
            {'l', 'д'}, {';', 'ж'}, {'\'', 'є'}, {'z', 'я'},
{'x', 'ч'}, {'c', 'с'}, {'v', 'м'}, {'b', 'и'}, {'n', 'т'},
{'m', 'ь'},
            {',', 'б'}, {'.', 'ю'}, {'`', '`'}, {'~', '~'},
{'!', '!'}, {'@', '"'}, {'#', '№'}, {'$', ';'}, {'%', '%'},
{'^', ':'},
            {'&', '?'}, {'*', '*'}, {'(', '('}, {')', ')'},
{'_', '_'}, {'+', '+'}, {'{', 'Х'}, {'}', 'Ї'}, {'|', '/'},
{':', 'Ж'},
            {'"', 'Є'}, {'<', 'Б'}, {'>', 'Ю'}, {'?', ','}, {'
', ' '}

        };
```

У разі виявлення помилки розкладки система автоматично виконує необхідну послідовність дій для її виправлення: видаляє неправильно набране користувачем слово, змінює мовну розкладку на правильну та вводить виправлений варіант.

Весь цей процес відбувається наочно для користувача, створюючи ефект миттєвого виправлення помилок.

На рисунку 3.2, своєю чергою, зображено діаграму послідовностей, що демонструє перевірку та виправлення слова.

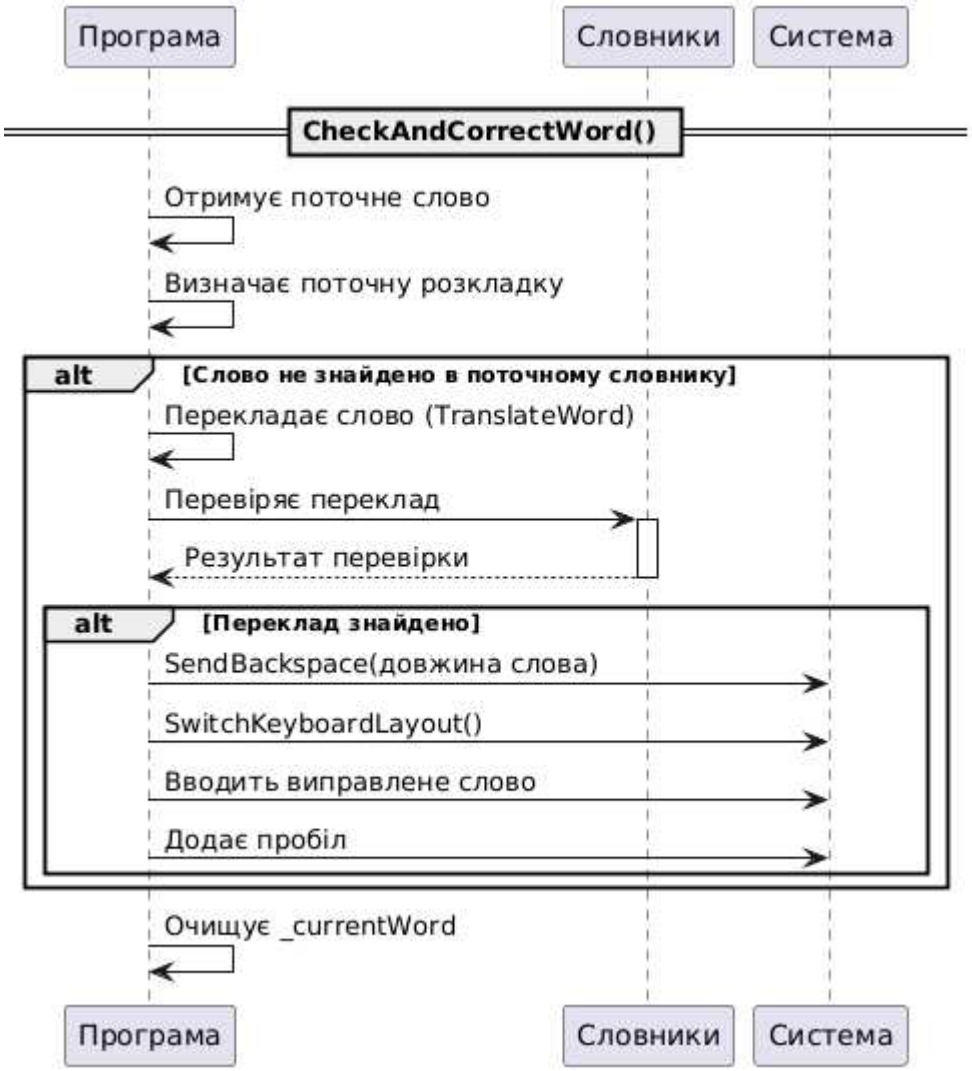

Рисунок 3.2 – Діаграма послідовності для перевірки та виправлення введеного слова

В наступному лістингу продемонстровано функцію зміни розкладки клавіатури в операційній системі.

```
private static void SwitchKeyboardLayout(bool toEnglish)
        {
            try
            {
                string layout = toEnglish ? ENGLISH_LAYOUT : 
UKRAINIAN_LAYOUT;
                IntPtr hkl = LoadKeyboardLayout(layout, 
KLF_ACTIVATE);

                // Отримуємо дескриптор поточного вікна та 
потоку
```

```
                IntPtr hwnd = GetForegroundWindow();
                uint threadId = GetWindowThreadProcessId(hwnd,
IntPtr.Zero);

                // Надсилаємо повідомлення про зміну розкладки
                PostMessage(hwnd, WM_INPUTLANGCHANGEREQUEST,
IntPtr.Zero, hkl);

                // Також активуємо розкладку для поточного
потоку
                ActivateKeyboardLayout(hkl, 0);

                Log($"Змінено розкладку на {(toEnglish ?
"англійську" : "українську")}");
            }
            catch (Exception ex)
            {
                Log($"Помилка зміни розкладки: {ex.Message}");
            }
        }
```

Для забезпечення зручності використання утиліти реалізовано роботу у фоновому режимі з наявною іконкою в системному треї.

Система також включає механізм логування, який дозволяє відстежувати роботу програми та діагностувати можливі проблеми. Логи зберігаються у файл, що спрощує аналіз роботи програми при виникненні помилок або несподіваної поведінки.

3.2 Принцип реалізації перехоплення подій клавіатури

Клас KeyboardHook реалізує механізм низькорівневого перехоплення подій клавіатури на рівні операційної системи. Його робота базується на використанні Windows API функцій для встановлення глобального хука, який дозволяє перехоплювати всі події введення незалежно від активного вікна.

Створення екземпляра класу призводить до ініціалізації необхідних системних ресурсів. Під час ініціалізації відбувається реєстрація користувацької процедури обробки подій клавіатури. Ця процедура викликається операційною системою при кожній події зміни стану клавіші.

Основний механізм роботи ґрунтується на використанні функції SetWindowsHookEx, яка встановлює глобальний хук для моніторингу подій клавіатури. Після встановлення хука кожна подія клавіатури спочатку проходить через зареєстровану процедуру обробки, перш ніж потрапити до цільового додатку.

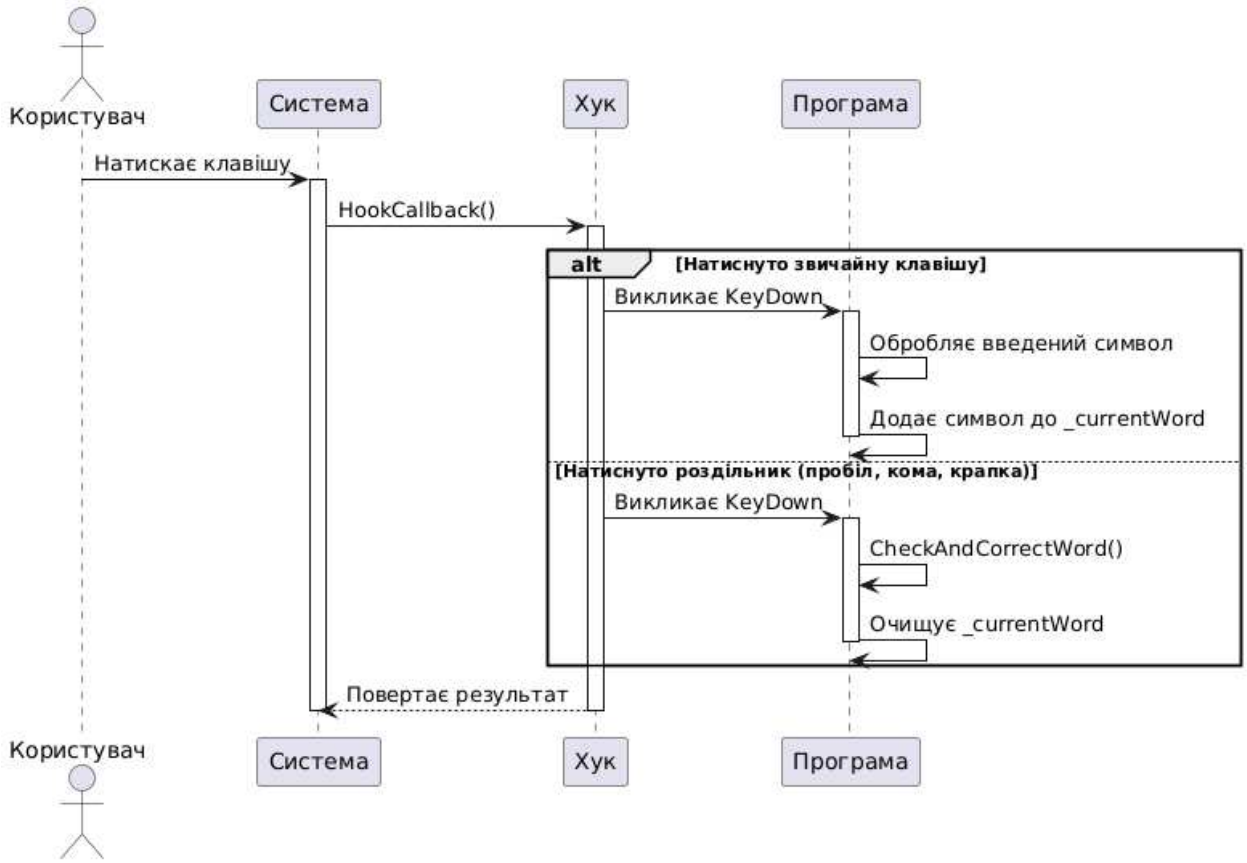

Рисунок 3.3 – Діаграма послідовності для обробки введення користувача

При отриманні події клавіатури система викликає зареєстрований зворотний виклик, який аналізує отримані дані. Ці дані містять інформацію про код клавіші, стан модифікаторів та інші параметри введення. Після аналізу даних генерується відповідна подія, яка сповіщає підписників про настання події.

Для забезпечення коректної роботи необхідно реалізувати механізм видалення хука при завершенні роботи. Це досягається шляхом реалізації

інтерфейсу IDisposable, який забезпечує звільнення системних ресурсів при знищенні об'єкта.

Особливістю роботи класу є обробка ситуацій, коли подія вже оброблена іншим обробником. У цьому випадку система може припинити подальшу обробку події або передати її далі по ланцюжку обробників.

Продуктивність реалізації забезпечується за рахунок мінімізації часу перебування в контексті хука. Всі обчислення та обробка даних виконуються асинхронно, що запобігає блокуванню вхідного потоку повідомлень.

Безпека роботи забезпечується перевіркою вхідних параметрів та обробкою виняткових ситуацій. У разі виникнення критичних помилок система намагається безпечно завершити роботу, звільнивши всі зайняті ресурси.

Взаємодія з основним додатком відбувається через публічні події, які дозволяють підписуватися на сповіщення про події клавіатури. Це забезпечує гнучкість у використанні класу та дозволяє інтегрувати його в існуючу архітектуру додатку.

3.3 Використання маршалінгу

Маршалінг являє собою процес перетворення об’єктів, структур або даних з одного формату в інший для забезпечення їх правильної передачі між різними середовищами виконання, мовами програмування чи платформами. У контексті програмування маршалінг дозволяє підготувати дані так, щоб вони могли бути коректно інтерпретовані іншою стороною, навіть якщо внутрішнє представлення цих даних у пам’яті відрізняється.

Маршалінг у цій програмі відіграє ключову роль у забезпеченні взаємодії між керованим середовищем .NET і некерованим кодом Windows API. Основне завдання маршалінгу в даному контексті — забезпечити коректний обмін даними між цими двома середовищами.

Рисунок 3.4 – Схема реалізації взаємодії утиліти з бібліотекою Windows «user32.dll»

Клас використовує атрибут DllImport для імпорту функцій з бібліотеки user32.dll, що є частиною Windows API. Цей атрибут вказує на необхідність маршалінгу параметрів і значень, що повертаються, між керованим і некерованим кодом.

Для кожного імпортованого методу визначено точний спосіб маршалінгу параметрів. Наприклад, для функцій, які працюють з дескрипторами вікон, використовується тип IntPtr, який є .NET-представленням вказівника в некерованому коді. Це дозволяє безпечно передавати дескриптори між керованим і некерованим кодом.

Атрибут MarshalAs застосовується для точного визначення способу маршалінгу значень, що повертаються. Він вказує, що певні значення мають бути інтерпретовані як логічні значення типу bool при поверненні з некерованого коду.

Для делегата LowLevelKeyboardProc визначено сигнатуру, яка відповідає очікуваній функції зворотного виклику в Windows API. Цей делегат

використовується для отримання повідомлень від системи про події клавіатури. Маршалінг забезпечує коректне перетворення параметрів між некерованим форматом Windows API та керованим кодом .NET.

Під час виклику функції SetWindowsHookEx відбувається маршалінг вказівника на функцію зворотного виклику з керованого коду в некерований. Це дозволяє системі викликати керований код при настанні подій клавіатури.

Маршалінг також відповідає за керування життєвим циклом даних, які передаються між керованим і некерованим кодом. Він забезпечує коректне звільнення ресурсів і запобігає витокам пам'яті, особливо при роботі з дескрипторами та вказівниками.

У разі виникнення помилок під час виклику некерованого коду маршалінг забезпечує перетворення кодів помилок Windows у відповідні винятки .NET, що дозволяє обробляти їх зручним способом.

В наступному наведеному програмному коді продемонстровано оголошення зовнішніх методів з атрибутом DllImport, маршалінг параметрів та делегатів:

```
    [DllImport("user32.dll", CharSet = CharSet.Auto, SetLastError
= true)]
    private static extern IntPtr SetWindowsHookEx(int idHook,
LowLevelKeyboardProc lpfn, IntPtr hMod, uint dwThreadId);

    [DllImport("user32.dll", CharSet = CharSet.Auto, SetLastError
= true)]
    [return: MarshalAs(UnmanagedType.Bool)]
    private static extern bool UnhookWindowsHookEx(IntPtr hhk);

    [DllImport("user32.dll", CharSet = CharSet.Auto, SetLastError
= true)]
    private static extern IntPtr CallNextHookEx(IntPtr hhk, int
nCode, IntPtr wParam, IntPtr lParam);

    [DllImport("kernel32.dll", CharSet = CharSet.Auto,
SetLastError = true)]
    private static extern IntPtr GetModuleHandle(string
lpModuleName);

    public delegate IntPtr LowLevelKeyboardProc(int nCode, IntPtr
wParam, IntPtr lParam);
```

Розглянемо функцію HookCallback, що є головним елементом у роботі клавіатурного хука, де маршалінг використовується для обробки даних, що надходять з некерованого коду Windows API:

```
private IntPtr HookCallback(int nCode, IntPtr wParam, IntPtr
lParam)
        {
            if (nCode >= 0)
            {
                int vkCode = Marshal.ReadInt32(lParam);
                var keyData = (Keys)vkCode;
                var args = new KeyEventArgs(keyData);

                if (wParam == (IntPtr)WM_KEYDOWN || wParam ==
(IntPtr)WM_SYSKEYDOWN)
                {
                    KeyDown?.Invoke(this, args);
                    if (!args.Handled)
                        KeyPressed?.Invoke(this, args);
                }
                else if (wParam == (IntPtr)WM_KEYUP || wParam ==
(IntPtr)WM_SYSKEYUP)
                {
                    KeyUp?.Invoke(this, args);
                }

                if (args.Handled)
                {
                    return (IntPtr)1;
                }
            }

            return CallNextHookEx(_hookID, nCode, wParam,
lParam);
        }
```

Основний сценарій маршалінгу в цьому методі включає кілька важливих аспектів. При отриманні повідомлення від системного хука, параметр lParam містить вказівник на структуру KBDLLHOOKSTRUCT, яка містить детальну інформацію про подію клавіатури. Для отримання коду натиснутої клавіші використовується метод Marshal.ReadInt32, який виконує читання 32-бітного цілого значення з вказаної пам'яті. Цей процес є прикладом маршалінгу даних з некерованої пам'яті в керований об'єкт .NET.

Параметри методу, такі як nCode, wParam та lParam, автоматично маршалуються при зворотному виклику. Це забезпечує прозору роботу з даними в керованому коді, хоч вони і надходять з рівня операційної системи.

Перетворення числового значення коду клавіші в тип перерахування Keys виконується через явне приведення типів, що також є частиною процесу маршалінгу. Це дозволяє зручно працювати з кодами клавіш у контексті .NET.

При перевірці значень wParam використовується приведення типів до IntPtr для порівняння з константами, що визначають тип події (натискання або відпускання клавіші). Це необхідно через те, що wParam передається як IntPtr, але порівнюється з числовими константами Windows API.

Значення методу, що повертається, також підлягає маршалінгу. Метод повертає IntPtr, який автоматично перетворюється в значення, зрозуміле Windows API. У випадку обробки події повертається ненульове значення, яке вказує системі на те, що подія оброблена і подальша її обробка не потрібна.

Останній виклик CallNextHookEx також використовує маршалінг для передачі параметрів у вихідний ланцюжок обробки подій Windows. Це забезпечує коректну роботу інших зареєстрованих обробників подій у системі.

Таким чином, маршалінг у цій програмі є важливою частиною механізму перехоплення подій клавіатури, забезпечуючи взаємодію між керованим кодом програми та низькорівневими функціями Windows API.

3.4 Демонстрація роботи

Після запуску програма автоматично активує перехоплення подій натискання клавіш та починає працювати у фоновому режимі, при цьому відбувається поява іконки у системному треї Windows.

Під час введення тексту в будь-якому додатку програма аналізує кожен введений символ.

Якщо користувач починає набирати слово, яке відповідає іншій мовній розкладці, наприклад, "ghbdsn" замість "привіт", програма фіксує це, але нічого не змінює до моменту завершення введення слова (рисунок 3.5).

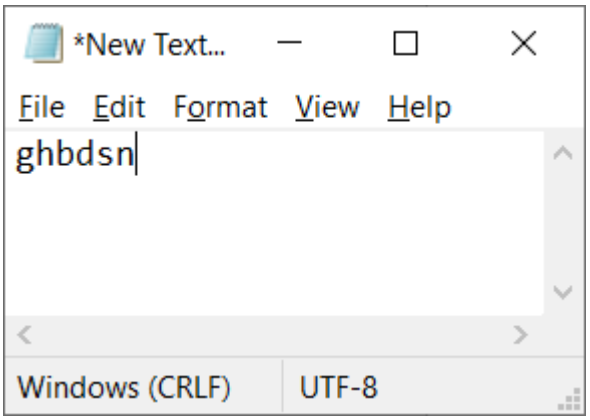

Рисунок 3.5 – Введення слова «привіт» в англійській розкладці

У момент натискання пробілу або іншого роздільника програма автоматично виконує перевірку.

Якщо введене слово не знайдено в словнику поточної розкладки, але знайдено відповідність у словнику іншої мови, відбувається автоматичне виправлення. Користувач бачить, як помилково введене слово замінюється на правильний варіант у відповідній розкладці (при цьому емулюється натискання клавіші Backspace).

На рисунку 3.6 видно, що після емуляції видалення слова та набору слова в правильній розкладці також додається розділовий знак, який слугував тригером для реєстрації закінчення введення слова.

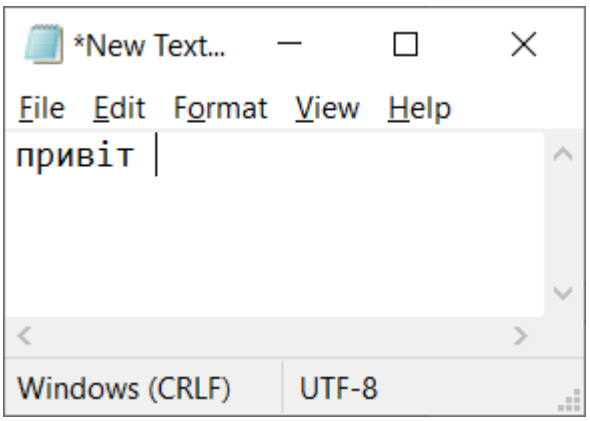

Рисунок 3.6 – Виправлення введеного слова та перемикання на українську розкладку

Важливо зазначити, що після заміни слова відбувається перемикання розкладки операційної системи Windows, що дозволяє користувачеві вводити наступне слово вже в коректній розкладці.

На рисунках 3.7 та 3.8 візуалізується введення слова «hello» після того, як програма вже виконала зміну розкладки операційної системи на українську. Відповідно рисунку зрозуміло, що вводиться набір букв кирилицею, але після натискання клавіші пробіл відбувается перетворення слова на англійське «hello».

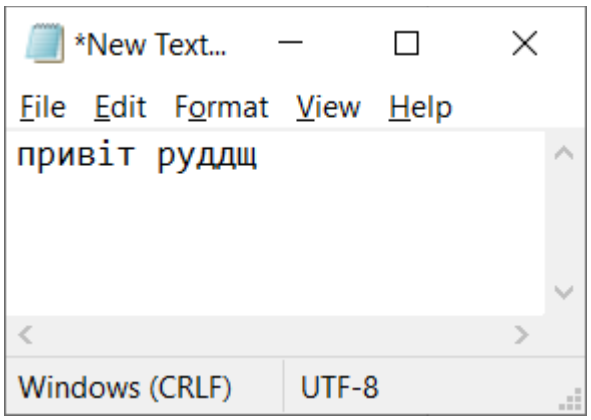

Рисунок 3.7 – Користувач вводить «hello», але розкладка в системі до цього перемкнулася на українську

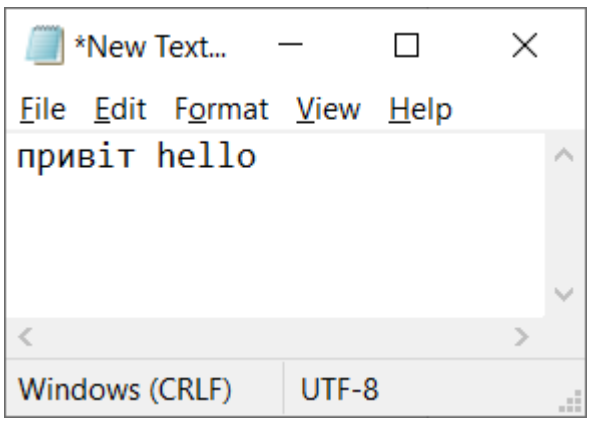

Рисунок 3.8 – Виконання перекладки слова «hello»

Якщо відбувається введення слова в правильній розкладці (тобто, слово може бути знайдене в словнику мови, що відповідає активній розкладці), то

введене слово не зазнає змін. Це продемонстровано на рисунку 3.9 при введенні слова «world».

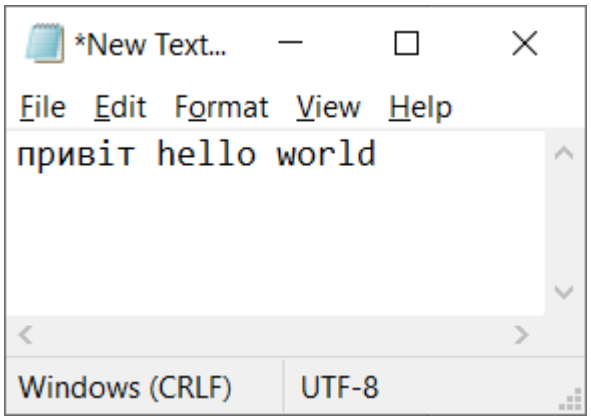

Рисунок 3.9 – При введенні у правильній розкладці введене слово не зазнає змін

Огляд зазначених прикладів наочно демонструє працездатність розробленої утіліти для автоматичного визначення коректної розкладки клавіатури на основі користувацького введення.

ВИСНОВКИ

В ході виконання кваліфікаційної роботи було розроблено утиліту для автоматичного виправлення розкладки клавіатури. Вона являє собою ефективний інструмент для підвищення продуктивності роботи з текстом у багатомовному середовищі. Отримані результати демонструють значний потенціал запропонованого рішення у сфері оптимізації введення текстових даних.

Основною перевагою розробленого програмного забезпечення є його здатність автоматично виправляти помилки, пов'язані з неправильним вибором розкладки клавіатури. Це досягнуто завдяки реалізації низькорівневого перехоплення подій введення та ефективного алгоритму перевірки слів. Програма працює у фоновому режимі, не заважаючи роботі інших застосунків, що робить її зручною у використанні.

Технічна реалізація програми базується на ефективних підходах до розробки програмного забезпечення. Використання колекцій для швидкого пошуку слів, а також ефективне управління ресурсами забезпечують стабільну роботу розробленого додатку.

Практична цінність роботи полягає в тому, що розроблений інструмент значною мірою спрощує роботу з текстом для користувачів, які постійно перемикаються між різними мовними розкладами. Це особливо актуально в умовах, коли необхідно швидко набирати текст, не відволікаючись на певні технічні аспекти. Вона може стати помічником для всіх, хто стикається з необхідністю постійного перемикання між різними мовними розкладами під час роботи з текстами.

Програма буде корисна в офісному середовищі, де працівникам часто доводиться готувати документи; для фахівців технічних спеціальностей, які працюють з програмним кодом; в освітній сфері, де може сприяти більш ефективній роботі студентів та викладачів; для тих, хто вивчає іноземні мови тощо. Розроблена утиліта дозволяє знизити навантаження на користувача,

автоматично виправляючи типові помилки, пов'язані з неправильним вибором розкладки клавіатури.

Таким чином, розроблена утиліта є завершеним програмним рішенням, яке вирішує поставлене завдання та має потенціал для подальшого вдосконалення. Отримані результати свідчать про доцільність подальшої роботи над проектом та його впровадження у повсякденну практику для економії часу в ході набору тексту у мультимовних середовищах.

ПЕРЕЛІК ПОСИЛАНЬ

1. Strasser S. Evaluation of smartphones with touchscreens as writing tools. Tampere, 2014. 96 p.

2. Alsultan A. F. A. Free-text keystroke dynamics authentication with a reduced need for training and language independency : thesis. 2017. URL: http://centaur.reading.ac.uk/69412/

3. Arif A. S., Stuerzlinger W. Predicting the cost of error correction in character-based text entry technologies. *The 28th international conference, Atlanta, Georgia, USA*, 10–15 April 2010. New York, USA.

4. Baldwin T., Chai J. Towards online adaptation and personalization of key-target resizing for mobile devices. *Proceedings of the 2012 ACM International Conference on Intelligent User Interfaces (IUI '12)*. New York, NY, USA, 2012. P. 11–20. URL: https://doi.org/10.1145/2166966.2166969.

5. Borghouts J. W. Task, Interrupted: Understanding the Effect of Time Costs on Task Interruptions During Data Entry : Ph.D. Dissertation. UCL Interaction Centre, Department of Psychology & Language Sciences, University College London, 2023.

6. Soukoreff R. W., MacKenzie I. S. Measuring errors in text entry tasks: An application of the Levenshtein string distance statistic. *Extended Abstracts of the ACM Conference on Human Factors in Computing Systems (CHI '01)*. Seattle, 2001. P. 319–320.

7. MacKenzie I. S. KSPC (Keystrokes per Character) as a Characteristic of Text Entry Techniques. Human Computer Interaction with Mobile Devices. Berlin, Heidelberg, 2002. P. 195–210. URL: https://doi.org/10.1007/3-540-45756-9_16

8. Wobbrock J. O. Measures of Text Entry Performance. *Text Entry Systems*. 2007. P. 47–74. URL: https://doi.org/10.1016/b978-012373591-1/50003-6

9. Yamada H. A historical study of typewriters and typing methods: From the position of planning Japanese parallels. *Journal of Information Processing*. 1980. Vol. 2. P. 175–202.

10. Brun D., Gouin-Vallerand C., George S. Design and evaluation of a versatile text input device for virtual and immersive workspaces. *Human–Computer Interaction*. 2024. P. 1–46. URL: https://doi.org/10.1080/07370024.2024.2352705 (date of access: 31.05.2025).

11. Dai M. Performance evaluation of QWERTY keyboards on foldable smartphones: keyboard layout and phrase complexity. *IASDR 2023: Life-Changing Design*. 2023. URL: https://doi.org/10.21606/iasdr.2023.404

12. Levenshtein V. I. Binary codes capable of correcting deletions, insertions, and reversals. 1965. Vol. 163. P. 845–848.

13. Wagner R. A., Fischer M. J. The string-to-string correction problem. *Journal of the Association for Computing Machinery.* 1974. Vol. 21. P. 168–173.

14. Damerau F. A technique for computer detection and correction of spelling errors. *Communications of the ACM*. 1964. Vol. 7, no. 3. P. 171–176.

15. Morgan H. L. Spelling correction in systems programs. *Communications of the ACM*. 1970. Vol. 13. P. 90–94.

16. A glossary of terms including a classification of typing errors / D. R. Gentner et al. *Cognitive aspects of skilled typewriting / ed. by W. E. Cooper*. New York, 1984. P. 39–43.

17. Chen T. Investigating retrospective interoperability between the accessible and mobile webs with regard to user input : thesis. 2011.

18. Kirk A. Improving the Accuracy of Mobile Touchscreen QWERTY Keyboards : Master's thesis. 2018. URL: https://digitalcommons.du.edu/etd/1512.

19. Reyal S. M. Investigation of Keyboard and Speech Based Text Entry on Mobile Devices. 2018. 252 p.

20. Does One Keyboard Fit All? Comparison and Evaluation of Device-Free Augmented Reality Keyboard Designs / M. Schenkluhn et al. *VRST 2023: 29th ACM Symposium on Virtual Reality Software and Technology.* New York, NY, USA, 2023. URL: https://doi.org/10.1145/3611659.3615692

21. Welcome to .NET [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/uk-ua/dotnet/welcome

22. .NET is the free, open-source, cross-platform framework for building modern apps and powerful cloud services. [Електронний ресурс] – Режим доступу до ресурсу: https://dotnet.microsoft.com/en-us/

23. C# - The modern, innovative, open-source programming language for building all your apps [Електронний ресурс] – Режим доступу до ресурсу: https://dotnet.microsoft.com/en-us/languages/csharp

Додаток А - Діаграма взаємодії компонентів

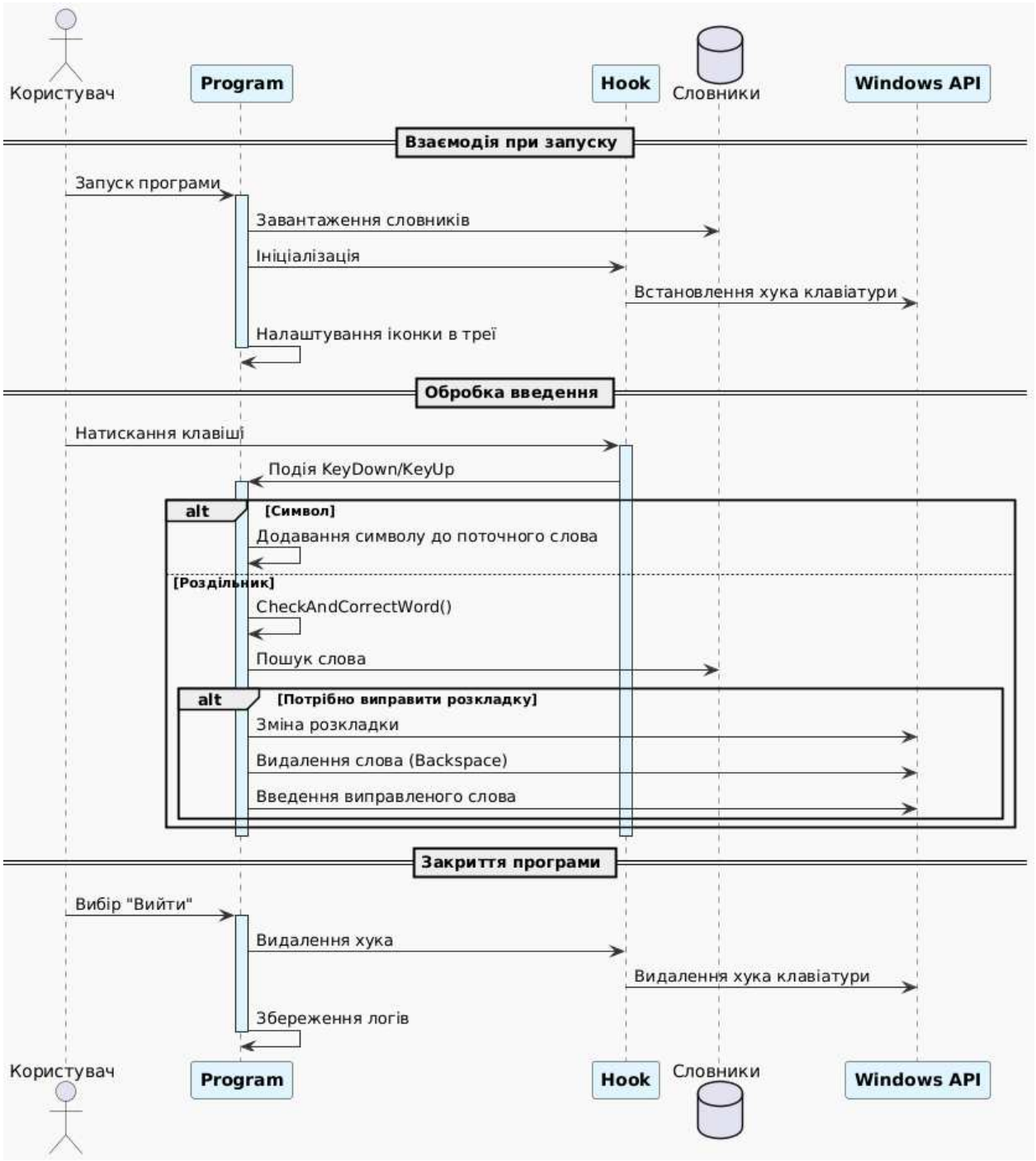

Рисунок А.1 – Діаграма взаємодії компонентів утіліти для автоматичного визначення коректної розкладки клавіатури на основі користувацького введення

Додаток Б - Діаграма станів

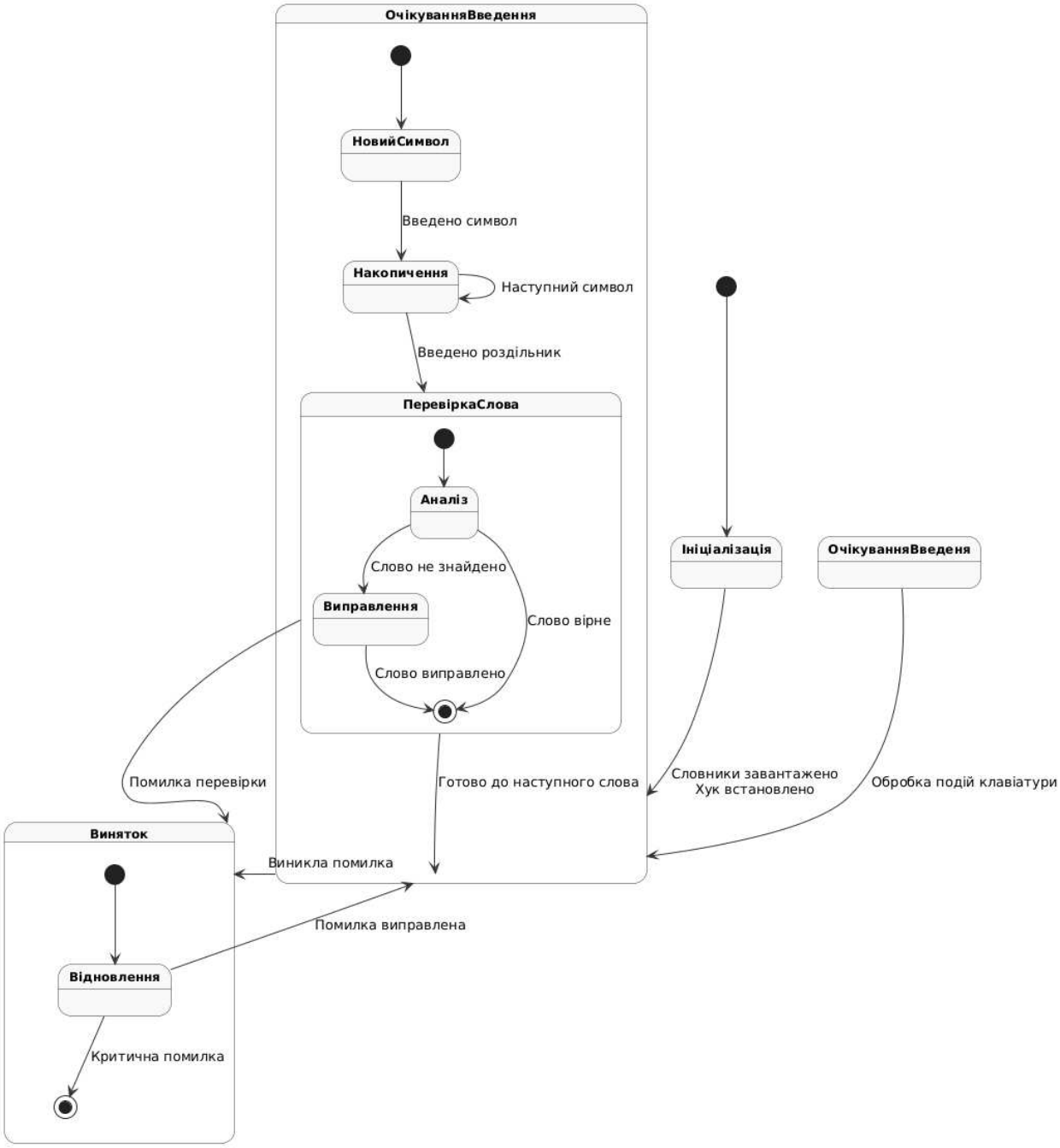

Рисунок Б.1 – Діаграма станів утіліти для автоматичного визначення коректної розкладки клавіатури на основі користувацького введення

Додаток В - Програмний код

Program.cs

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Drawing;
using System.Globalization;
using System.IO;
using System.Linq;
using System.Runtime.InteropServices;
using System.Text;
using System.Threading;
using System.Windows.Forms;

namespace KS2
{
    internal static class Program
    {
        #region Win32 API
        [DllImport("user32.dll")]
        private static extern IntPtr GetForegroundWindow();

        [DllImport("user32.dll")]
        private static extern uint
GetWindowThreadProcessId(IntPtr hwnd, IntPtr proccess);

        [DllImport("user32.dll")]
        private static extern IntPtr GetKeyboardLayout(uint
thread);

        [DllImport("user32.dll")]
        private static extern bool PostMessage(IntPtr hWnd, uint
Msg, IntPtr wParam, IntPtr lParam);

        [DllImport("user32.dll")]
        private static extern IntPtr LoadKeyboardLayout(string
pwszKLID, uint Flags);

        [DllImport("user32.dll")]
        private static extern bool GetKeyboardState(byte[]
lpKeyState);

        [DllImport("user32.dll")]
        private static extern int ToUnicodeEx(uint wVirtKey,
uint wScanCode, byte[] lpKeyState,
            [Out, MarshalAs(UnmanagedType.LPWStr)] StringBuilder
pwszBuff, int cchBuff,
            uint wFlags, IntPtr dwhkl);

        [DllImport("user32.dll")]
```

```
        private static extern uint MapVirtualKey(uint uCode,
uint uMapType);

        [DllImport("user32.dll")]
        private static extern IntPtr
ActivateKeyboardLayout(IntPtr hkl, uint Flags);

        private const uint KLF_ACTIVATE = 1;
        private const uint WM_INPUTLANGCHANGEREQUEST = 0x0050;
        private const string UKRAINIAN_LAYOUT = "00000422";
        private const string ENGLISH_LAYOUT = "00000409";
        #endregion

        private static NotifyIcon _trayIcon;
        private static KeyboardHook _keyboardHook = new
KeyboardHook();
        private static StringBuilder _currentWord = new
StringBuilder();
        private static bool _isShiftPressed;
        private static bool _isCapsLockOn;
        private static HashSet<string> _englishDictionary = new
HashSet<string>(StringComparer.OrdinalIgnoreCase);
        private static HashSet<string> _ukrainianDictionary =
new HashSet<string>(StringComparer.OrdinalIgnoreCase);
        private static readonly string _logPath =
Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "ks2.log");

        // Мапінг між англійською та українською розкладкою
        private static readonly Dictionary<char, char>
_enToUaMap = new Dictionary<char, char>
        {
            {'q', 'й'}, {'w', 'ц'}, {'e', 'у'}, {'r', 'к'},
{'t', 'е'}, {'y', 'н'}, {'u', 'г'}, {'i', 'ш'}, {'o', 'щ'},
{'p', 'з'},
            {'[', 'х'}, {']', 'ї'}, {'a', 'ф'}, {'s', 'і'},
{'d', 'в'}, {'f', 'а'}, {'g', 'п'}, {'h', 'р'}, {'j', 'о'},
{'k', 'л'},
            {'l', 'д'}, {';', 'ж'}, {'\'', 'є'}, {'z', 'я'},
{'x', 'ч'}, {'c', 'с'}, {'v', 'м'}, {'b', 'и'}, {'n', 'т'},
{'m', 'ь'},
            {',', 'б'}, {'.', 'ю'}, {'`', '`'}, {'~', '~'},
{'!', '!'}, {'@', '"'}, {'#', '№'}, {'$', ';'}, {'%', '%'},
{'^', ':'},
            {'&', '?'}, {'*', '*'}, {'(', '('}, {')', ')'},
{'_', '_'}, {'+', '+'}, {'{', 'Х'}, {'}', 'Ї'}, {'|', '/'},
{':', 'Ж'},
            {'"', 'Є'}, {'<', 'Б'}, {'>', 'Ю'}, {'?', ','}, {'
', ' '}
        };


        [STAThread]
```

```
        static void Main()
        {
            Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

            // Завантаження словників
            LoadDictionaries();


            // Ініціалізація програми
            InitializeTrayIcon();
            SetupKeyboardHook();


            // Запуск обробки вводу
            Application.Run();
        }


        private static void LoadDictionaries()
        {
            try
            {
                string dictPath =
Path.Combine(AppDomain.CurrentDomain.BaseDirectory,
"Dictionaries");
                string engDictPath = Path.Combine(dictPath,
"english.txt");
                string uaDictPath = Path.Combine(dictPath,
"ukrainian.txt");

                Log($"Пошук словників за шляхом: {dictPath}");
                Log($"Шлях до англійського словника:
{engDictPath}");
                Log($"Шлях до українського словника:
{uaDictPath}");

                // Перевіряємо існування директорії
                if (!Directory.Exists(dictPath))
                {
                    Log("ПОМИЛКА: Директорія Dictionaries не
знайдена", true);
                    Directory.CreateDirectory(dictPath);
                    Log("Створено директорію Dictionaries");
                }

                // Перевіряємо існування файлів словників
                if (!File.Exists(engDictPath))
                {
                    Log("ПОМИЛКА: Англійський словник не
знайдено. Створюю порожній файл.", true);
```

```
                    File.WriteAllText(engDictPath,
"hello\nworld\ntest\nkeyboard\nlayout");
                }

                if (!File.Exists(uaDictPath))
                {
                    Log("ПОМИЛКА: Український словник не
знайдено. Створюю порожній файл.", true);
                    File.WriteAllText(uaDictPath,
"привіт\nсвіт\nтест\nклавіатура\nрозкладка", Encoding.UTF8);
                }

                // Завантажуємо англійський словник
                if (File.Exists(engDictPath))
                {
                    var englishWords =
File.ReadAllLines(engDictPath)
                        .Where(line =>
!string.IsNullOrWhiteSpace(line))
                        .Select(word => word.Trim().ToLower())
                        .Distinct()
                        .ToList();

                    _englishDictionary = new
HashSet<string>(englishWords, StringComparer.OrdinalIgnoreCase);
                    Log($"Завантажено англійський словник. Слів:
{_englishDictionary.Count}");
                    Log("Приклади слів: " + string.Join(", ",
_englishDictionary.Take(10)) + "...");
                }

                // Завантажуємо український словник
                if (File.Exists(uaDictPath))
                {
                    var ukrainianWords =
File.ReadAllLines(uaDictPath, Encoding.UTF8)
                        .Where(line =>
!string.IsNullOrWhiteSpace(line))
                        .Select(word => word.Trim().ToLower())
                        .Distinct()
                        .ToList();

                    _ukrainianDictionary = new
HashSet<string>(ukrainianWords,
StringComparer.OrdinalIgnoreCase);
                    Log($"Завантажено український словник. Слів:
{_ukrainianDictionary.Count}");
                    Log("Приклади слів: " + string.Join(", ",
_ukrainianDictionary.Take(10)) + "...");
                }

                // Перевіряємо кодування файлів
```

```
                CheckFileEncoding(engDictPath, "Англійський
словник");
                CheckFileEncoding(uaDictPath, "Український
словник");
            }
            catch (Exception ex)
            {
                Log($"КРИТИЧНА ПОМИЛКА при завантаженні
словників: {ex.Message}", true);
                Log($"Stack Trace: {ex.StackTrace}");
            }
        }

        private static void CheckFileEncoding(string filePath,
string dictName)
        {
            try
            {
                var encoding = DetectEncoding(filePath);
                Log($"{dictName} кодування:
{encoding.EncodingName}");

                // Перевіряємо перші кілька рядків на
коректність
                var lines = File.ReadAllLines(filePath,
encoding);
                Log($"{dictName} перші 3 рядки: {string.Join(" |
", lines.Take(3))}");
            }
            catch (Exception ex)
            {
                Log($"Помилка перевірки кодування {dictName}:
{ex.Message}");
            }
        }

        private static Encoding DetectEncoding(string filename)
        {
            // Читаємо перші кілька байт для визначення BOM
            var bom = new byte[4];
            using (var file = new FileStream(filename,
FileMode.Open, FileAccess.Read))
            {
                file.Read(bom, 0, 4);
            }

            // Аналізуємо BOM
            if (bom[0] == 0x2b && bom[1] == 0x2f && bom[2] ==
0x76) return Encoding.UTF7;
            if (bom[0] == 0xef && bom[1] == 0xbb && bom[2] ==
0xbf) return Encoding.UTF8;
```

```
            if (bom[0] == 0xff && bom[1] == 0xfe) return
Encoding.Unicode; //UTF-16LE
            if (bom[0] == 0xfe && bom[1] == 0xff) return
Encoding.BigEndianUnicode; //UTF-16BE
            if (bom[0] == 0 && bom[1] == 0 && bom[2] == 0xfe &&
bom[3] == 0xff) return Encoding.UTF32;

            // Якщо BOM немає, намагаємося визначити кодування
за вмістом
            return Encoding.GetEncoding(1251); // Windows-1251
за замовчуванням для кирилиці
        }


        private static void InitializeTrayIcon()
        {
            // Створення контекстного меню
            var contextMenu = new ContextMenuStrip();
            contextMenu.Items.Add("Вихід", null, (s, e) =>
Application.Exit());

            // Створення іконки в треї
            _trayIcon = new NotifyIcon
            {
                Icon = new Icon(SystemIcons.Application, 40,
40),
                ContextMenuStrip = contextMenu,
                Text = "KS2 - Коректор розкладки",
                Visible = true
            };

            // Обробник подвійного кліку по іконці
            _trayIcon.DoubleClick += (s, e) =>
            {
                // Тут можна додати відкриття головного вікна
програми
                MessageBox.Show("KS2 - Коректор
розкладки\nПрацює у фоновому режимі.", "KS2",
                    MessageBoxButtons.OK,
MessageBoxIcon.Information);
            };
        }


        private static void SetupKeyboardHook()
        {
            _keyboardHook.KeyDown += (sender, e) =>
            {
                try
                {
                    // Обробка модифікаторів
```

```
                    if (e.KeyCode == Keys.LShiftKey || e.KeyCode
== Keys.RShiftKey)
                    {
                        _isShiftPressed = true;
                        return;
                    }
                    else if (e.KeyCode == Keys.Capital)
                    {
                        _isCapsLockOn = !_isCapsLockOn;
                        return;
                    }

                    // Визначаємо, чи є символ роздільником
                    bool isSeparator = e.KeyCode == Keys.Space
|| e.KeyCode == Keys.Enter ||
                                       e.KeyCode == Keys.Tab ||
e.KeyCode == Keys.OemPeriod ||
                                       e.KeyCode == Keys.Oemcomma
|| e.KeyCode == Keys.OemQuestion ||
                                       e.KeyCode == Keys.Oem1 ||
e.KeyCode == Keys.Oem7 ||
                                       e.KeyCode == Keys.OemMinus
|| e.KeyCode == Keys.OemOpenBrackets ||
                                       e.KeyCode == Keys.Oem6 ||
e.KeyCode == Keys.Oem5 ||
                                       e.KeyCode ==
Keys.OemCloseBrackets ||
                                       e.KeyCode == Keys.Return ||
e.KeyCode == Keys.Escape;

                    if (isSeparator)
                    {
                        if (_currentWord.Length > 0)
                        {
                            CheckAndCorrectWord();
                            _currentWord.Clear();
                        }
                        return;
                    }
                    else if (e.KeyCode == Keys.Back)
                    {
                        if (_currentWord.Length > 0)
                        {

_currentWord.Remove(_currentWord.Length - 1, 1);
                        }
                        return;
                    }

                    // Ігноруємо службові клавіші
                    if (Control.ModifierKeys.HasFlag(Keys.Alt)
||
```

```
Control.ModifierKeys.HasFlag(Keys.Control) ||
                        Control.ModifierKeys.HasFlag(Keys.LWin)
||
                        Control.ModifierKeys.HasFlag(Keys.RWin))
                    {
                        return;
                    }

                    // Обробка літер та цифр
                    if ((e.KeyCode >= Keys.A && e.KeyCode <=
Keys.Z) ||
                        (e.KeyCode >= Keys.D0 && e.KeyCode <=
Keys.D9) ||
                        e.KeyCode == Keys.Oem1 || e.KeyCode ==
Keys.Oem7 ||
                        e.KeyCode == Keys.OemMinus || e.KeyCode
== Keys.OemOpenBrackets ||
                        e.KeyCode == Keys.Oem6 || e.KeyCode ==
Keys.OemCloseBrackets ||
                        e.KeyCode == Keys.OemPeriod || e.KeyCode
== Keys.Oemcomma ||
                        e.KeyCode == Keys.OemQuestion ||
e.KeyCode == Keys.OemQuotes ||
                        e.KeyCode == Keys.OemSemicolon ||
e.KeyCode == Keys.OemBackslash ||
                        e.KeyCode == Keys.OemPipe || e.KeyCode
== Keys.Oemtilde)
                    {
                        // Отримуємо поточний стан клавіатури
                        byte[] keyboardState = new byte[256];
                        if (!GetKeyboardState(keyboardState))
                        {
                            Log("Не вдалося отримати стан
клавіатури");
                            return;
                        }

                        // Отримуємо поточну розкладку
                        IntPtr hwnd = GetForegroundWindow();
                        uint threadId =
GetWindowThreadProcessId(hwnd, IntPtr.Zero);
                        IntPtr keyboardLayout =
GetKeyboardLayout(threadId);

                        // Конвертуємо віртуальний ключ у символ
                        uint virtualKey = (uint)e.KeyCode;
                        uint scanCode =
MapVirtualKey(virtualKey, 0);

                        // Встановлюємо правильний стан Shift
```

```
                        keyboardState[(int)Keys.ShiftKey] =
(byte)(_isShiftPressed ? 0x80 : 0);
                        keyboardState[(int)Keys.Capital] =
(byte)(_isCapsLockOn ? 0x01 : 0);

                        // Скидаємо інші модифікатори
                        keyboardState[(int)Keys.ControlKey] = 0;
                        keyboardState[(int)Keys.Menu] = 0; //
Alt

                        StringBuilder stringBuilder = new
StringBuilder(10);
                        int result = ToUnicodeEx(
                            virtualKey,
                            scanCode,
                            keyboardState,
                            stringBuilder,
                            stringBuilder.Capacity,
                            0,
                            keyboardLayout);

                        if (result > 0)
                        {
                            char c = stringBuilder[0];
                            // Перевіряємо, що символ не є
пробілом або керуючим символом
                            if (!char.IsControl(c) &&
!char.IsWhiteSpace(c))
                            {
                                _currentWord.Append(c);
                                Log($"Додано символ: '{c}' (код:
{(int)c})");
                            }
                        }
                    }
                }
                catch (Exception ex)
                {
                    Log($"Помилка в обробнику клавіатури:
{ex.Message}");
                }
            };

            _keyboardHook.KeyUp += (sender, e) =>
            {
                if (e.KeyCode == Keys.LShiftKey || e.KeyCode ==
Keys.RShiftKey)
                {
                    _isShiftPressed = false;
                }
            };
```

```
            _keyboardHook.Install();
            Log("Клавіатурний хук успішно встановлено");
        }


        private static void CheckAndCorrectWord()
        {
            if (_currentWord.Length == 0) return;

            string word = _currentWord.ToString();
            if (string.IsNullOrWhiteSpace(word)) return;

            // Запам'ятовуємо, чи був пробіл після слова
            bool hasTrailingSpace = false;
            if (word.EndsWith(" "))
            {
                hasTrailingSpace = true;
                word = word.TrimEnd();
                if (word.Length == 0) return;
            }

            // Отримуємо поточну розкладку
            bool isEnglishLayout = IsEnglishLayoutActive();
            string currentLayout = isEnglishLayout ?
"англійська" : "українська";
            Log($"\n--- Початок перевірки слова ---");
            Log($"Перевіряємо слово: '{word}' (довжина:
{word.Length})");
            Log($"Поточна розкладка: {currentLayout}");
            Log($"Пробіл після слова: {hasTrailingSpace}");

            // Функція для перевірки слова у словнику з
можливими варіантами
            bool IsWordInDictionary(string wordToCheck,
HashSet<string> dictionary)
            {
                if (string.IsNullOrEmpty(wordToCheck)) return
false;

                // Перевіряємо слово як є
                if (dictionary.Contains(wordToCheck.ToLower()))
                    return true;

                // Перевіряємо варіанти з різним регістром
                string lower = wordToCheck.ToLower();
                string title = char.ToUpper(wordToCheck[0]) +
                               (wordToCheck.Length > 1 ?
wordToCheck.Substring(1).ToLower() : "");

                return dictionary.Contains(lower) ||
dictionary.Contains(title);
            }
```

```
            // Перевіряємо слово в поточному словнику
            bool wordFound = isEnglishLayout
                ? IsWordInDictionary(word, _englishDictionary)
                : IsWordInDictionary(word,
_ukrainianDictionary);

            Log($"Слово '{word}' у поточному словнику:
{(wordFound ? "ЗНАЙДЕНО" : "не знайдено")}");

            if (wordFound)
            {
                Log($"Слово '{word}' коректне для поточної
розкладки. Вихід з методу.");
                Log("--- Кінець перевірки слова ---\n");
                return;
            }


            // Намагаємося перекласти слово
            string translatedWord = isEnglishLayout
                ? TranslateWord(word, _enToUaMap)
                : TranslateWord(word, _enToUaMap.ToDictionary(x
=> x.Value, x => x.Key));

            Log($"Переклад слова: '{word}' ->
'{translatedWord}'");

            if (string.IsNullOrEmpty(translatedWord) ||
translatedWord == word)
            {
                Log($"Не вдалося перекласти слово або переклад
збігається з оригіналом. Вихід з методу.");
                Log("--- Кінець перевірки слова ---\n");
                return;
            }


            // Перевіряємо перекладене слово в іншому словнику
            bool translatedWordFound = isEnglishLayout
                ? IsWordInDictionary(translatedWord,
_ukrainianDictionary)
                : IsWordInDictionary(translatedWord,
_englishDictionary);

            Log($"Перекладене слово '{translatedWord}' у
цільовому словнику: {(translatedWordFound ? "ЗНАЙДЕНО" : "не
знайдено")}");

            if (translatedWordFound)
            {
```

```
                // Видаляємо поточне слово
                SendBackspace(word.Length + (hasTrailingSpace ?
1 : 0));

                // Змінюємо розкладку
                bool switchToEnglish = !isEnglishLayout;
                Log($"Змінюємо розкладку на {(switchToEnglish ?
"англійську" : "українську")}");
                SwitchKeyboardLayout(switchToEnglish);

                // Чекаємо трохи для застосування змін
                System.Threading.Thread.Sleep(300);

                // Вводимо виправлене слово з пробілом після
нього
                string wordToType = translatedWord + " ";

                Log($"Вводимо виправлене слово:
'{wordToType}'");

                // Надсилаємо слово
                //SendKeys.SendWait("{BACKSPACE}");
                SendKeys.SendWait(wordToType);

                Log($"Виправлено: '{word}' -> '{translatedWord}'
(змінено розкладку на {(isEnglishLayout ? "українську" :
"англійську")})");
            }
            else
            {
                Log($"Слово не знайдено в жодному зі словників:
'{word}'. Переклад: '{translatedWord}'");
            }

            Log("--- Кінець перевірки слова ---\n");
        }


        private static string TranslateWord(string word,
Dictionary<char, char> charMap)
        {
            if (string.IsNullOrEmpty(word)) return word;

            StringBuilder result = new StringBuilder();
            foreach (char c in word)
            {
                // Пробіли та інші роздільники залишаємо як є
                if (char.IsWhiteSpace(c) ||
char.IsPunctuation(c) || char.IsDigit(c))
                {
                    result.Append(c);
                    continue;
```

```
                }

                // Намагаємося знайти символ у словнику
                char lowerChar = char.ToLower(c);
                bool isUpper = char.IsUpper(c);

                if (charMap.TryGetValue(lowerChar, out char
translatedChar))
                {
                    // Зберігаємо регістр
                    result.Append(isUpper ?
char.ToUpper(translatedChar) : translatedChar);
                }
                else
                {
                    // Якщо символ не знайдено, залишаємо як є
                    result.Append(c);
                }
            }
            return result.ToString();
        }


        private static bool IsEnglishLayoutActive()
        {
            try
            {
                IntPtr foregroundWindow = GetForegroundWindow();
                uint foregroundProcess =
GetWindowThreadProcessId(foregroundWindow, IntPtr.Zero);
                int keyboardLayout =
GetKeyboardLayout(foregroundProcess).ToInt32() & 0xFFFF;
                return keyboardLayout == 0x0409; // 0x0409 -
англійська (США)
            }
            catch
            {
                return true; // За замовчуванням вважаємо, що
активна англійська розкладка
            }
        }


        private static void SwitchKeyboardLayout(bool toEnglish)
        {
            try
            {
                string layout = toEnglish ? ENGLISH_LAYOUT :
UKRAINIAN_LAYOUT;
                IntPtr hkl = LoadKeyboardLayout(layout,
KLF_ACTIVATE);
```

```
                // Отримуємо дескриптор поточного вікна та
потоку
                IntPtr hwnd = GetForegroundWindow();
                uint threadId = GetWindowThreadProcessId(hwnd,
IntPtr.Zero);

                // Надсилаємо повідомлення про зміну розкладки
                PostMessage(hwnd, WM_INPUTLANGCHANGEREQUEST,
IntPtr.Zero, hkl);

                // Також активуємо розкладку для поточного
потоку
                ActivateKeyboardLayout(hkl, 0);

                Log($"Змінено розкладку на {(toEnglish ?
"англійську" : "українську")}");
            }
            catch (Exception ex)
            {
                Log($"Помилка зміни розкладки: {ex.Message}");
            }
        }


        private static void SendBackspace(int count)
        {
            for (int i = 0; i < count; i++)
            {
                //System.Threading.Thread.Sleep(300);
                SendKeys.SendWait("{BACKSPACE}");
            }
        }


        private static void Log(string message, bool
showInTooltip = true)
        {
            try
            {
                string logMessage = $"[{DateTime.Now:yyyy-MM-dd
HH:mm:ss}] {message}";
                File.AppendAllText(_logPath, logMessage +
Environment.NewLine);

                if (showInTooltip && _trayIcon != null)
                {
                    _trayIcon.ShowBalloonTip(3000, "KS2",
message, ToolTipIcon.Info);
                }
            }
            catch
            {
```

```
                // Ігноруємо помилки логування
            }
        }
    }
}
```

KeyboardHooks.cs

```
using System;
using System.Runtime.InteropServices;
using System.Windows.Forms;

namespace KS2
{
    public class KeyboardHook : IDisposable
    {
        private const int WH_KEYBOARD_LL = 13;
        private const int WM_KEYDOWN = 0x0100;
        private const int WM_KEYUP = 0x0101;
        private const int WM_SYSKEYDOWN = 0x0104;
        private const int WM_SYSKEYUP = 0x0105;

        [DllImport("user32.dll", CharSet = CharSet.Auto,
SetLastError = true)]
        private static extern IntPtr SetWindowsHookEx(int
idHook, LowLevelKeyboardProc lpfn, IntPtr hMod, uint
dwThreadId);

        [DllImport("user32.dll", CharSet = CharSet.Auto,
SetLastError = true)]
        [return: MarshalAs(UnmanagedType.Bool)]
        private static extern bool UnhookWindowsHookEx(IntPtr
hhk);

        [DllImport("user32.dll", CharSet = CharSet.Auto,
SetLastError = true)]
        private static extern IntPtr CallNextHookEx(IntPtr hhk,
int nCode, IntPtr wParam, IntPtr lParam);

        [DllImport("kernel32.dll", CharSet = CharSet.Auto,
SetLastError = true)]
        private static extern IntPtr GetModuleHandle(string
lpModuleName);

        public delegate IntPtr LowLevelKeyboardProc(int nCode,
IntPtr wParam, IntPtr lParam);

        public event EventHandler<KeyEventArgs> KeyPressed;
        public event EventHandler<KeyEventArgs> KeyDown;
        public event EventHandler<KeyEventArgs> KeyUp;
```

```
        private readonly LowLevelKeyboardProc _proc;
        private IntPtr _hookID = IntPtr.Zero;

        public KeyboardHook()
        {
            _proc = HookCallback;
        }

        public void Install()
        {
            _hookID = SetHook(_proc);
        }

        public void Uninstall()
        {
            UnhookWindowsHookEx(_hookID);
        }


        private IntPtr SetHook(LowLevelKeyboardProc proc)
        {
            using (var curProcess =
System.Diagnostics.Process.GetCurrentProcess())
            using (var curModule = curProcess.MainModule)
            {
                return SetWindowsHookEx(WH_KEYBOARD_LL, proc,
                    GetModuleHandle(curModule.ModuleName), 0);
            }
        }

        private IntPtr HookCallback(int nCode, IntPtr wParam,
IntPtr lParam)
        {
            if (nCode >= 0)
            {
                int vkCode = Marshal.ReadInt32(lParam);
                var keyData = (Keys)vkCode;
                var args = new KeyEventArgs(keyData);

                if (wParam == (IntPtr)WM_KEYDOWN || wParam ==
(IntPtr)WM_SYSKEYDOWN)
                {
                    KeyDown?.Invoke(this, args);
                    if (!args.Handled)
                        KeyPressed?.Invoke(this, args);
                }
                else if (wParam == (IntPtr)WM_KEYUP || wParam ==
(IntPtr)WM_SYSKEYUP)
                {
                    KeyUp?.Invoke(this, args);
                }
```

```
                if (args.Handled)
                {
                    return (IntPtr)1;
                }
            }

            return CallNextHookEx(_hookID, nCode, wParam,
lParam);
        }

        public void Dispose()
        {
            Uninstall();
        }
    }
}
```