НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ ТА ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ПОГОДЖЕНО	**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ**
Декан факультету	**Завідувач кафедри комп'ютерних наук**
_____ **Ігор БОЛБОТ**	_____ **Белла ГОЛУБ**
(підпис)	(підпис)
«___»_____ 2026 р.	«___»_____ 2026 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Програмний додаток перекладу тексту з екрану в реальному часі**»**

Спеціальність «121 Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення»

Гарант освітньої програми

к.т.н., доцент _____ **Ганна ВАЙГАНГ**

(підпис)

Керівник бакалаврської кваліфікаційної роботи

_____ _____ **Ярослав ГОРДІЙ**

(науковий ступінь та вчене звання) (підпис)

Консультант бакалаврської кваліфікаційної роботи

к.т.н., доцент _____ **Белла ГОЛУБ**

(науковий ступінь та вчене звання) (підпис)

Виконав _____ **Максим ЗАКОМІРНИЙ**

(підпис)

КИЇВ – 2026

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ ТА ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ

Завідувач кафедри комп'ютерних наук

к.т.н., доцент _____ **Белла ГОЛУБ**

(підпис)

«___»_____ 2025 р.

ЗАВДАННЯ

ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ ЗДОБУВАЧУ

Закомірному Максиму Сергійовичу

(прізвище, ім'я, по батькові)

Спеціальність «121 Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення»

Тема бакалаврської кваліфікаційної роботи

«Програмний додаток перекладу тексту з екрану в реальному часі»

Затверджена наказом від «19» грудня 2025 р. № 3204 «С»

Термін подання завершеної роботи на кафедру «22» травня 2026 р.

Вихідні дані до бакалаврської кваліфікаційної роботи:

Алгоритми програмного додатку для перекладу тексту з екрану в реальному часі.

Перелік питань, що підлягають дослідженню:

Переклад тексту з екрану як об'єкт дослідження, проєктування програмного забезпечення, розробка програмного додатку, рекомендації щодо впровадження та експлуатації системи

Дата видачі завдання «19» грудня 2025 р.

Керівник бакалаврської кваліфікаційної роботи _____ **Ярослав ГОРДІЙ**

(підпис)

Консультант бакалаврської кваліфікаційної роботи _____ **Белла ГОЛУБ**

(підпис)

Завдання взяв до виконання _____ **Максим ЗАКОМІРНИЙ**

(підпис)

ЗМІСТ

ВСТУП

Актуальність. Під час роботи за комп’ютером користувач часто стикається з інформацією різними мовами. Це можуть бути вебсайти, інтерфейси програм, відео, ігри, технічна документація або інший цифровий контент. На практиці часто виникає ситуація, коли текст потрібно перекласти швидко, але звичайний спосіб із копіюванням і вставкою не підходить. Наприклад, текст може бути частиною зображення або інтерфейсу програми, і його неможливо просто скопіювати. У таких випадках користувачу доводиться вручну переписувати текст або користуватись сторонніми сервісами, що займає зайвий час і відволікає від основної задачі. Особливо це помітно, коли потрібно перекладати багато невеликих фрагментів – процес стає рутинним і неефективним.

Окрім цього, значна частина сучасного контенту взагалі не передбачає можливості взаємодії з текстом як із текстом – наприклад, у відео, стрімах, іграх або спеціалізованих програмах. У таких випадках користувач фактично “обмежений” у доступі до інформації, якщо не володіє мовою на достатньому рівні. Це створює додаткові труднощі як у навчанні, так і в повсякденній роботі.

Через це актуальною є розробка інструменту, який може перекладати текст безпосередньо з екрану. У такому випадку користувач обирає потрібну область і одразу бачить результат перекладу, не переписуючи текст вручну та не відкриваючи окремий перекладач. Особливо це корисно під час навчання, роботи з технічною документацією, перегляду відео або використання програм, які не мають локалізації.

Мета. Метою даної дипломної роботи є розробка програмного додатку для автоматичного розпізнавання тексту з обраної області екрану та його перекладу в режимі, близькому до реального часу. Тобто користувач сам обирає частину екрану, а програма далі самостійно виконує всі необхідні дії – від отримання зображення до відображення перекладеного тексту. Для досягнення цієї мети необхідно реалізувати кілька основних функцій: захоплення області екрану, розпізнавання тексту, переклад отриманого тексту та відображення результату у

зручному вигляді для користувача. Також важливо забезпечити стабільну роботу програми та можливість її налаштування під потреби користувача.

Методи та технології, які використовувалися при розробці. Під час розробки використано сучасні технології та інструменти. Програма реалізована мовою програмування C# з використанням технології WPF для створення графічного інтерфейсу. WPF підходить для цієї задачі, оскільки дає змогу створити звичайне головне вікно програми та окреме overlay-вікно для показу перекладу. Для розпізнавання тексту застосовується OCR (оптичне розпізнавання символів), що дозволяє отримувати текстову інформацію із зображень. Переклад виконується за допомогою підключення до зовнішніх сервісів через API, що дає змогу отримувати актуальні результати перекладу. Також у програмі використовуються асинхронні методи, які дозволяють виконувати обробку даних без “зависання” інтерфейсу, що важливо для комфортної роботи користувача. Окремо реалізовано підтримку гарячих клавіш для швидкого доступу до основних функцій програми.

Апробація. Апробація результатів роботи була здійснена шляхом підготовки та подання тез на тему “Програмний додаток перекладу тексту з екрану в реальному часі” на студентську наукову конференцію “Теоретичні та прикладні аспекти розробки комп’ютерних систем” 2026 року.

Структура записки. Пояснювальна записка складається з чотирьох основних розділів. У першому розділі проведено системний аналіз предметної області, розглянуто основні особливості задачі, визначено вимоги до програмної системи, виконано моделювання предметної області та проаналізовано існуючі рішення. У другому розділі описано процес проєктування програмного забезпечення, зокрема побудовано логічну модель даних, діаграми класів, пакетів та компонентів. У третьому розділі наведено процес розробки системи, включаючи створення інформаційної бази, вибір інструментів та реалізацію основних програмних модулів. Четвертий розділ присвячений тестуванню програмного додатку, перевірці основних сценаріїв його роботи, опису середовища розгортання та визначенню вимог до апаратного й програмного

забезпечення. Загальний обсяг записки становить 60 сторінки, кількість використаних джерел – 25, кількість додатків – 3.

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

У сучасному цифровому середовищі користувачі постійно взаємодіють із великою кількістю інформації, значна частина якої представлена іноземними мовами. Це можуть бути вебсайти, програмні інтерфейси, відео, ігри, технічна документація, а також різноманітні інформаційні системи. У зв’язку з розвитком глобальних сервісів та міжнародної співпраці використання іноземних мов стало звичайним явищем, однак для багатьох користувачів це створює певні труднощі.

Особливо це відчутно під час роботи з технічною або спеціалізованою інформацією. Наприклад, документація до програмного забезпечення, довідкові матеріали або інтерфейси програм часто доступні лише англійською мовою. У таких випадках користувач змушений регулярно звертатися до перекладачів, що уповільнює роботу та створює додаткове навантаження.

Однією з ключових проблем є формат представлення тексту. Не завжди текст доступний у вигляді, в якому його можна легко скопіювати. У багатьох випадках він є частиною зображення, відео або графічного інтерфейсу. Це можуть бути, наприклад, субтитри у відео, текст у комп’ютерних іграх або елементи інтерфейсу програм. У таких ситуаціях стандартні інструменти перекладу є малоефективними, оскільки потребують ручного введення тексту або використання додаткових засобів.

У більшості випадків процес перекладу виглядає як послідовність кількох дій: користувач знаходить текст, копіює або переписує його, відкриває перекладач, вставляє текст і лише після цього отримує результат. Такий підхід є незручним і займає багато часу, особливо якщо подібні дії потрібно виконувати часто.

Ситуація ще більше ускладнюється, якщо текст неможливо скопіювати, приклад на рис. 1. У такому випадку користувач змушений вводити його вручну

або використовувати додаткові інструменти для розпізнавання тексту, що може призводити до помилок і ще більше збільшує час виконання задачі.

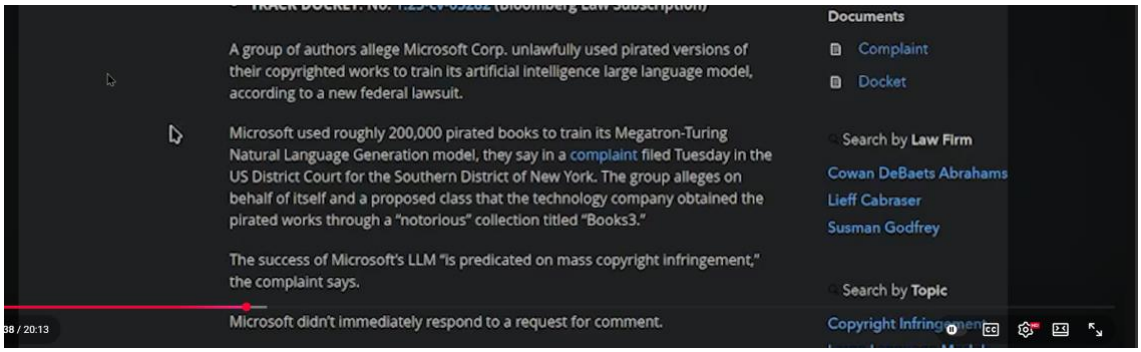

Рис. 1 Приклад тексту який не можна виділити

Важливим фактором також є швидкість отримання перекладу. У деяких випадках користувачу необхідно отримати результат практично миттєво, наприклад під час перегляду відео або роботи з програмами в реальному часі. Затримки навіть у кілька секунд можуть знижувати зручність використання перекладу або робити його менш ефективним.

У зв'язку з цим виникає потреба у створенні програмного рішення, яке дозволяє автоматично отримувати текст безпосередньо з екрану та виконувати його переклад без додаткових дій з боку користувача. Така система повинна працювати швидко, стабільно та бути зручною у використанні.

Для кращого розуміння загальної ідеї роботи додатку процес перекладу можна подати як послідовність кількох етапів. Спочатку користувач обирає область екрану, з якої потрібно отримати текст, і запускає процес перекладу. Після цього система автоматично виконує захоплення зображення, передає його на OCR-обробку, отримує розпізнаний текст, виконує переклад і відображає результат користувачу.

У такому процесі користувач бере участь переважно на початковому етапі, коли визначає область екрану та запускає переклад. Усі подальші дії виконуються програмою автоматично, що зменшує кількість ручних операцій і робить роботу з іншомовним текстом швидшою та зручнішою.

Важливою особливістю предметної області є необхідність роботи в режимі, близькому до реального часу. Це означає, що всі етапи обробки повинні

виконуватись достатньо швидко, щоб користувач не відчував значних затримок. Також потрібно враховувати обмеження апаратних ресурсів, оскільки система має працювати стабільно навіть під час тривалого використання.

Узагальнену схему процесу перекладу тексту з екрану наведено на рис. 2.

Рис. 2 Загальна схема процесу перекладу тексту з екрану

Окрім швидкодії, важливим фактором є зручність використання. Інтерфейс системи повинен бути інтуїтивно зрозумілим, а основні функції – доступними без зайвих дій. Використання гарячих клавіш та автоматизації процесів дозволяє значно підвищити комфорт роботи користувача.

У межах цієї роботи предметна область пов’язана з трьома основними процесами: отриманням зображення з екрану, розпізнаванням тексту та його перекладом. Саме поєднання цих етапів і формує основу розроблюваного додатку.

1.2 Моделювання предметної області

Моделювання предметної області є важливим етапом розробки програмного забезпечення, оскільки дозволяє представити систему з різних точок зору, описати її основні процеси, поведінку та взаємодію з користувачем. Завдяки побудові моделей можна краще зрозуміти логіку роботи системи ще до її реалізації.

У даному підрозділі для опису системи використовуються UML-діаграми, які відображають основні аспекти її функціонування, зокрема загальну структуру, взаємодію з користувачем, послідовність виконання дій та обмін даними між компонентами.

1.2.1 Узагальнена модель системи. Для виділення основних частин системи було побудовано діаграму абстракцій. На рис. 3 можна побачити, які елементи беруть участь у процесі перекладу та за які дії вони відповідають.

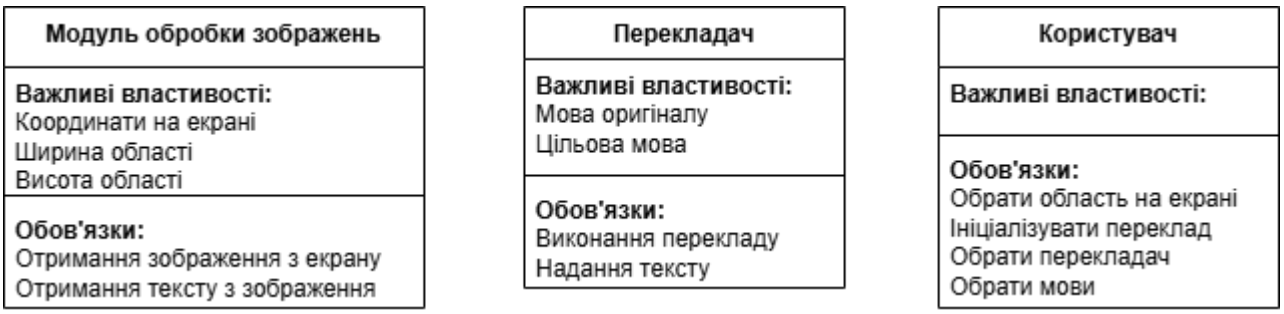

Рис. 3 Узагальнена модель системи

На рисунку показано три ключові складові системи: користувача, модуль обробки зображень і перекладач. Саме між цими елементами розподіляються основні дії, які потрібні для отримання перекладеного тексту.

Модуль обробки зображень відповідає за роботу із графічною інформацією, отриманою з екрану. До його важливих властивостей належать координати області на екрані, її ширина та висота. Саме ці параметри визначають, яка частина екрану буде оброблятися. Основними обов’язками модуля є отримання зображення з екрану, а також подальше отримання тексту із цього зображення.

Перекладач виконує обробку текстової інформації. Його ключовими властивостями є мова оригіналу та цільова мова, які визначають напрям

перекладу. До обов’язків перекладача належить виконання перекладу отриманого тексту та надання результату для подальшого відображення.

Користувач є активним елементом системи, який ініціює всі основні процеси. Його обов’язки включають вибір області на екрані, ініціалізацію перекладу, вибір перекладача та встановлення мов перекладу. Саме через дії користувача система переходить у робочий режим.

У результаті ця діаграма дає загальне уявлення про систему ще без прив’язки до конкретних класів і файлів. На цьому рівні головне – зрозуміти, хто бере участь у процесі перекладу та які функції виконує кожна частина.

1.2.2 Діаграма прецедентів. Після визначення загальної структури системи потрібно показати, які дії можуть виконувати її учасники. Для цього використано діаграму прецедентів, наведену на рис. 4.

З діаграми видно, що взаємодія не обмежується лише користувачем. Окрім нього, у процесі беруть участь модуль обробки зображень і перекладач, які виконують технічні дії після запуску перекладу.

Користувач є основним актором, який ініціює роботу системи. Він виконує такі дії: обирає область на екрані для подальшої обробки, ініціалізує процес перекладу, обирає перекладач, а також встановлює мови перекладу. Саме ці дії визначають параметри роботи системи.

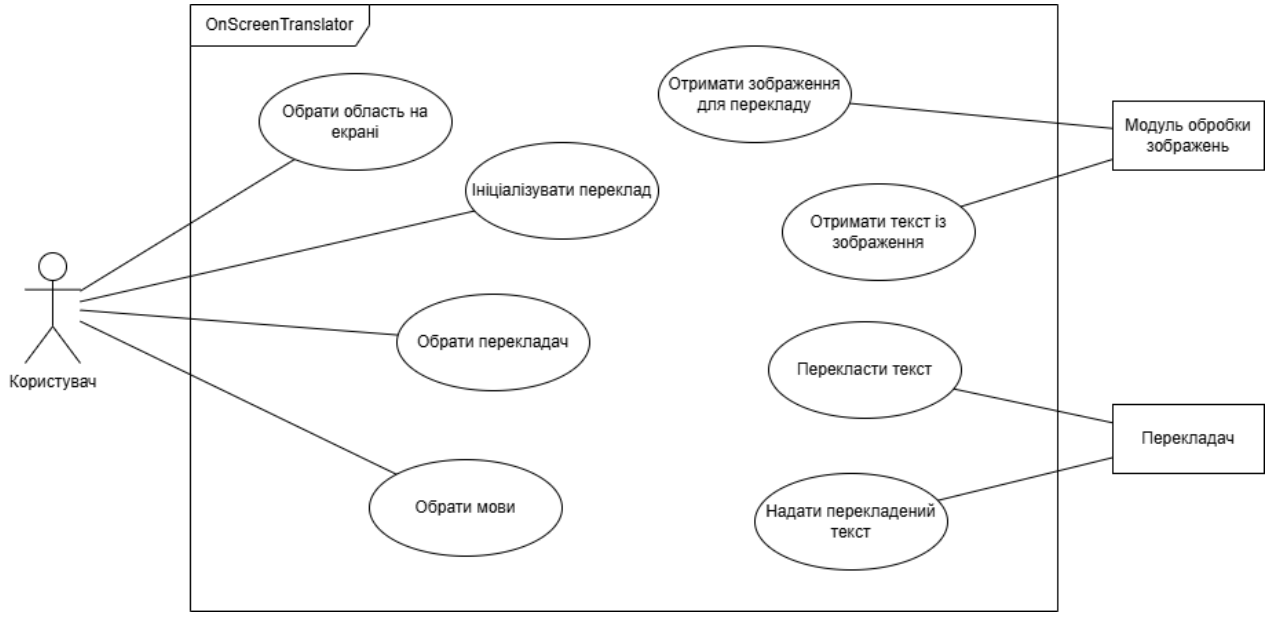

Рис. 4 Діаграма прецедентів

Модуль обробки зображень відповідає за підготовку даних для подальшої обробки. Його функції включають отримання зображення з обраної області екрану та тексту із цього зображення. Таким чином, цей модуль забезпечує перехід від графічної інформації до текстової.

Перекладач виконує обробку тексту, отриманого з модуля обробки зображень. До його функцій належить переклад тексту відповідно до обраних мов, а також передача перекладеного результату для подальшого використання.

Взаємодія між акторами відбувається послідовно: користувач задає параметри роботи, модуль обробки зображень формує текстові дані, після чого перекладач виконує переклад і надає результат.

Ця діаграма фактично показує набір основних сценаріїв, які повинна підтримувати система. Вона також допомагає зрозуміти, які дії виконує користувач, а які вже належать внутрішнім модулям програми.

1.2.3 Діаграма активності. Для опису послідовності дій під час виконання перекладу було побудовано діаграму активності, яка знаходиться в **додатку Б**. Вона показує, які етапи проходить система від вибору області екрану до відображення перекладеного тексту. Окремо на діаграмі показані перевірки, які впливають на подальший хід алгоритму.

На діаграмі видно, що процес починається не з самого перекладу, а з вибору області екрану. Це логічно, оскільки без заданої області система не знає, яку частину екрану потрібно обробляти. На цьому етапі передбачена перевірка: якщо область не вибрана, система повертається до повторного виконання цієї дії. Лише після успішного вибору області відбувається перехід до наступного етапу.

Далі користувач обирає перекладач, який буде використовуватись у системі, а також встановлює мови перекладу. Після цього виконується запуск процесу перекладу.

Наступним кроком є перевірка доступності перекладача. Якщо перекладач недоступний, система повертається до етапу запуску перекладу. У випадку, якщо перекладач доступний, процес продовжується.

Після цього система виконує отримання зображення з обраної області екрану. Отримане зображення передається на етап розпізнавання, де з нього виділяється текстова інформація.

Далі отриманий текст передається до модуля перекладу, де виконується його обробка та переклад відповідно до обраних мов.

На завершальному етапі перекладений текст відображається користувачу.

Діаграма активності добре показує, що процес перекладу не є простою прямою послідовністю. У ньому є перевірки, повернення до попередніх етапів і умови, від яких залежить подальше виконання.

1.2.4 Діаграма послідовності. Щоб детальніше показати обмін повідомленнями між учасниками процесу, було побудовано діаграму послідовності. На рис. 5 показано, у якому порядку користувач, модуль обробки зображень та перекладач передають один одному дані.

У діаграмі беруть участь три актори: користувач, модуль обробки зображень та перекладач. Взаємодія між ними відбувається у вигляді послідовного обміну повідомленнями.

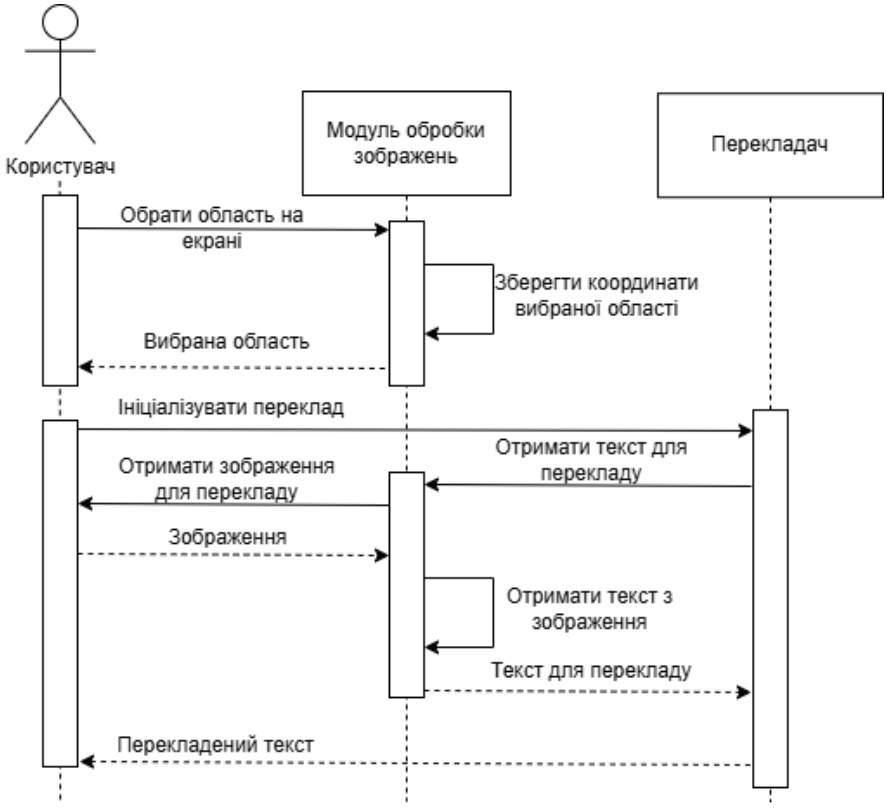

Рис. 5 Діаграма послідовності

Користувач ініціалізує процес перекладу, надсилаючи відповідний запит до перекладача. Отримавши цей запит, перекладач звертається до модуля обробки зображень із вимогою надати текст для перекладу.

Модуль обробки зображень, у свою чергу, не має готового тексту, тому ініціює отримання зображення: він надсилає запит користувачу на отримання зображення з обраної області екрану. Користувач передає зображення назад до модуля.

Після отримання зображення модуль обробки виконує внутрішню операцію розпізнавання тексту. Отриманий текст передається до перекладача для подальшої обробки.

Перекладач виконує переклад отриманого тексту та надсилає результат безпосередньо користувачу у вигляді перекладеного тексту.

На відміну від діаграми активності, ця діаграма більше зосереджена не на самих діях, а на тому, хто саме їх виконує і кому передає результат. Це корисно для подальшого проєктування модулів системи.

1.3 Огляд інформаційних джерел та існуючих рішень

На етапі аналізу предметної області важливо розглянути існуючі програмні рішення, які використовуються для перекладу тексту, а також інструменти, що частково вирішують поставлену задачу. Це дозволяє визначити їхні можливості, обмеження та сформувати вимоги до розроблюваного додатку.

У межах підрозділу спочатку розглянуто загальні інструменти перекладу, після чого виконано аналіз спеціалізованих програм, які реалізують переклад тексту безпосередньо з екрану.

1.3.1 Сучасні інструменти перекладу тексту. Для звичайного перекладу користувачі найчастіше використовують онлайн-сервіси, наприклад **Google Translate** або **DeepL**. Вони забезпечують переклад

тексту між різними мовами з використанням сучасних алгоритмів машинного навчання.

Основною перевагою таких сервісів є висока якість перекладу та підтримка великої кількості мов. Крім того, вони мають простий інтерфейс і не потребують встановлення додаткового програмного забезпечення.

Разом з тим, використання таких інструментів має ряд обмежень. Зокрема, користувач повинен вручну скопіювати текст і вставити його у поле перекладача. Це створює додаткові дії та знижує швидкість роботи. Особливо це критично у випадках, коли текст є частиною зображення або графічного інтерфейсу і не може бути скопійований стандартними засобами.

Таким чином, хоча онлайн-перекладачі забезпечують якісний переклад, вони не вирішують задачу автоматичного перекладу тексту безпосередньо з екрану.

1.3.2 Аналіз спеціалізованих програмних рішень. Серед програм, які безпосередньо реалізують переклад тексту з екрану, можна виділити кілька основних рішень.

1. Realtime Screen OCR Translator – платний десктопний додаток, що дозволяє перекладати текст із вибраної області екрану в режимі реального часу.

Переваги: простий у використанні, наявність режиму тестування OCR.

Недоліки: платна ліцензія, обмежена кількість налаштувань, перекриває частину екрану.

2. RSTGameTranslation – відкритий додаток, орієнтований на переклад тексту з ігор.

Переваги: підтримка декількох мов, велика кількість налаштувань, можливість використання різних OCR і перекладачів.

Недоліки: складний інтерфейс, залежність від сторонніх сервісів.

3. LiveScreenTranslator – відкритий проєкт для перекладу тексту з екрану.

Переваги: простота використання, базова функціональність.

Недоліки: складність встановлення, менш зручний інтерфейс.

4. Translumo – програма для автоматичного перекладу тексту з екрану з акцентом на швидкість роботи.

Переваги: швидкий переклад, мінімалістичний інтерфейс, підтримка різних сервісів.

Недоліки: залежність від сторонніх API.

1.3.3 Порівняльний аналіз існуючих рішень. Для більш детального аналізу було виконано порівняння розглянутих програм в таблиці 1.1 за основними характеристиками.

Таблиця 1.1

Порівняння існуючих рішень

Програма	**Тип**	**Вибір області**	**Робота в реальному часі**	**Over-lay**	**Налашту-вання**	**Простота використання**
Realtime Screen OCR Translator	Комер-ційна	+	+	+	Сильно обмежені	Висока
RSTGameTranslation	Open Source	+	+	+	Гнучкі	Низька
LiveScreen Translator	Open Source	+	Частково	+	Обмежені	Середня
Translumo	Open Source	+	+	+	Гнучкі	Висока

З таблиці видно, що всі розглянуті рішення підтримують базову функціональність, проте відрізняються рівнем зручності використання, кількістю налаштувань та стабільністю роботи.

1.4 Аналіз вимог до програмної системи

Перед проєктуванням додатку потрібно визначити, які функції він має виконувати і які обмеження слід врахувати. Для цієї системи це особливо

важливо, оскільки вона одночасно працює з екраном, OCR-модулем, сервісом перекладу та overlay-вікном.

З урахуванням особливостей предметної області вимоги до розроблюваного програмного додатку доцільно розділити на функціональні та нефункціональні, а також окремо виділити обмеження системи.

1.4.1 Функціональні вимоги. Функціональні вимоги описують те, що користувач зможе робити в програмі, і які дії система має виконувати під час перекладу.

До базових функцій системи належить можливість вибору області екрану, з якої буде здійснюватися зчитування інформації. Після вибору області система повинна виконувати захоплення зображення та передавати його до модуля розпізнавання тексту.

Наступним етапом є розпізнавання тексту із зображення за допомогою OCR. Отриманий текст передається до модуля перекладу, який виконує обробку та повертає результат відповідною мовою. Результат перекладу повинен відображатися користувачу у вигляді overlay-вікна без необхідності перемикання між різними програмами.

Окрім основного функціоналу, система повинна підтримувати автоматичне оновлення перекладу через заданий інтервал часу. Це дозволяє користувачу отримувати актуальний переклад без додаткових дій. Також передбачається можливість одноразового перекладу, коли обробка виконується лише один раз.

Важливою функцією є підтримка різних режимів роботи, наприклад переклад у вигляді окремих рядків або як суцільного тексту. Це дозволяє адаптувати систему під різні сценарії використання.

Користувач повинен мати можливість змінювати мови перекладу, налаштовувати параметри системи та керувати її роботою. Для підвищення зручності передбачається використання гарячих клавіш, що дозволяють швидко виконувати основні дії.

Основні функціональні вимоги наведені у таблиці 1.2.

Таблиця 1.2

Функціональні вимоги системи

№	Вимога	Опис
1	Вибір області	Користувач має можливість обрати довільну область екрану, з якої буде здійснюватися зчитування тексту для подальшого перекладу
2	Захоплення	Система виконує отримання зображення з обраної області екрану у реальному часі
3	OCR	Розпізнавання тексту із зображення за допомогою технології оптичного розпізнавання символів
4	Переклад	Обробка розпізнаного тексту та його переклад на обрану користувачем мову
5	Відображення	Виведення перекладеного тексту у вигляді overlay-вікна поверх інших програм
6	Автооновлення	Автоматичне повторне виконання перекладу через заданий інтервал часу
7	Режими	Підтримка різних режимів роботи (наприклад, разовий переклад або безперервне оновлення)
8	Гарячі клавіші	Можливість керування основними функціями системи за допомогою гарячих клавіш

1.4.2 Нефункціональні вимоги. Нефункціональні вимоги стосуються не окремих кнопок чи команд, а загальної якості роботи програми: швидкодії, стабільності, зручності та сумісності.

Однією з ключових вимог є продуктивність. У системі передбачено швидке виконання операцій розпізнавання та перекладу, щоб користувач міг отримувати результат без значних затримок.

Також важливою є надійність. Програма повинна працювати стабільно протягом тривалого часу та коректно обробляти можливі помилки.

Зручність використання є ще одним важливим фактором. Інтерфейс системи повинен бути інтуїтивно зрозумілим, щоб користувач міг швидко освоїти основні можливості програми.

Окремо слід відзначити ефективність використання ресурсів. Система не повинна створювати значне навантаження на процесор або пам’ять, оскільки вона може працювати паралельно з іншими програмами.

Сумісність із сучасними операційними системами також є важливою вимогою, що забезпечує можливість використання програми на різних пристроях.

Основні нефункціональні вимоги наведені у таблиці 1.3.

Таблиця 1.3

Нефункціональні вимоги системи

№	Вимога	Опис
1	Продуктивність	Для системи важливо швидке виконання операцій розпізнавання та перекладу без значних затримок
2	Надійність	Програма має стабільно працювати протягом тривалого часу та коректно обробляти можливі помилки
3	Зручність	Інтерфейс користувача повинен бути інтуїтивно зрозумілим та не вимагати додаткового навчання
4	Сумісність	Програмний додаток повинен коректно працювати на підтримуваних версіях операційної системи
5	Ресурси	Система повинна використовувати мінімально необхідні ресурси комп’ютера під час роботи

1.4.3 Обмеження системи. Під час розробки необхідно враховувати ряд обмежень, які можуть впливати на роботу системи.

Якість розпізнавання тексту залежить від якості зображення. У випадку нечіткого або зашумленого зображення можливі помилки.

Швидкість перекладу залежить від роботи зовнішніх сервісів, що використовуються через API. При нестабільному інтернет-з’єднанні можливі затримки.

Також можуть існувати обмеження на кількість запитів до сервісів перекладу, що потрібно враховувати при використанні системи.

Окрім цього, продуктивність залежить від апаратних характеристик комп’ютера користувача.

1.5 Постановка завдання

На основі проведеного аналізу предметної області та існуючих рішень можна сформулювати задачу, яку необхідно вирішити в межах даної роботи. Вона полягає у створенні програмного додатку, що забезпечує автоматичний переклад тексту безпосередньо з екрану користувача.

1.5.1 Загальний опис задачі. Необхідно розробити програму, яка дозволяє користувачу обрати певну область екрану, після чого система автоматично виконує захоплення зображення, розпізнає текст та здійснює його переклад. Результат повинен відображатися у вигляді overlay-вікна поверх інших програм.

Додаток орієнтований на ситуації, коли текст неможливо скопіювати стандартними засобами, наприклад у відео, іграх або графічних інтерфейсах.

1.5.2 Функціональні вимоги. Розроблюваний додаток повинен реалізовувати такі основні функції:

- вибір області екрану для подальшої обробки;
- захоплення зображення з обраної області;
- розпізнавання тексту із зображення;
- переклад отриманого тексту на обрану мову;
- відображення перекладу у вигляді overlay-вікна;
- запуск і зупинка процесу перекладу;

- можливість налаштування параметрів (мови, інтервал оновлення, вибір перекладача).

1.5.3 Нефункціональні вимоги. Окрім функціональних можливостей, система повинна відповідати ряду нефункціональних вимог.

Програма повинна працювати у середовищі операційної системи Windows та не створювати значного навантаження на ресурси комп'ютера. Важливо забезпечити стабільну роботу при тривалому використанні, а також виконання основних операцій без затримок, що можуть впливати на зручність роботи.

Інтерфейс користувача повинен бути простим та зрозумілим, без необхідності додаткового навчання.

1.5.4 Вхідні та вихідні дані. У процесі роботи система використовує як вхідні, так і вихідні дані.

До вхідних даних належать:

- координати та розміри вибраної області екрану;
- зображення, отримане з цієї області;
- параметри, задані користувачем (мови перекладу, режим роботи, інтервал оновлення).

Вихідним результатом є перекладений текст, який відображається на екрані у вигляді overlay-вікна.

1.5.5 Структура системи. З точки зору реалізації програмний додаток складається з кількох основних компонентів:

- модуль захоплення зображення з екрану;
- модуль розпізнавання тексту (OCR);
- модуль перекладу;
- модуль відображення результату (overlay);
- модуль керування та налаштувань.

Взаємодія цих компонентів забезпечує повний цикл обробки даних – від отримання зображення до відображення перекладеного тексту.

1.5.6 Очікуваний результат. У результаті виконання роботи має бути створений програмний додаток, який забезпечує швидке отримання перекладу тексту з екрану без необхідності ручного введення або копіювання. Користувач повинен мати можливість обрати потрібну область екрану, запустити переклад і отримати результат у спеціальному overlay-вікні.

Розроблений додаток має виконувати повний цикл обробки даних: захоплення зображення з обраної області, розпізнавання тексту за допомогою OCR, переклад отриманого тексту та відображення результату користувачу. Такий підхід повинен зменшити кількість ручних дій, пришвидшити роботу з іншомовною інформацією та забезпечити доступ до перекладу в режимі реального часу.

Також очікується, що система буде зручною у використанні, стабільною під час тривалої роботи та достатньо швидкою для повсякденних задач, зокрема під час роботи з відео, програмами, іграми або технічною документацією.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Загальна характеристика

Після виконання аналізу предметної області наступним етапом є проєктування програмного забезпечення. На цьому етапі формується загальна структура системи, визначаються її основні складові та принципи взаємодії між ними. Також уточнюється модель даних, яка використовується в програмному додатку.

Основною задачею проєктування є побудова такої структури системи, яка буде достатньо простою для розуміння, але водночас гнучкою для подальшого розвитку. Важливо, щоб окремі частини програми були логічно відокремлені одна від одної, що спрощує як реалізацію, так і підтримку коду.

При розробці структури враховуються особливості предметної області. Зокрема, система повинна працювати в режимі, близькому до реального часу, не створювати зайвого навантаження на ресурси комп’ютера та залишатися зручною для користувача. Це впливає як на вибір архітектурних рішень, так і на спосіб організації взаємодії між компонентами.

У межах даного розділу розглядаються такі питання:

- побудова логічної моделі даних;
- визначення основних компонентів програмної системи;
- опис взаємозв’язків між окремими частинами програми;
- організація архітектури додатку на рівні класів, пакетів та компонентів.

Окрему увагу приділено моделі даних. Незважаючи на те, що система не передбачає зберігання великого обсягу інформації, правильна організація структури даних дозволяє спростити реалізацію та забезпечити можливість подальшого розширення функціональності без суттєвих змін у коді.

2.2 Логічна модель даних

У процесі проєктування було сформовано логічну модель даних, яка описує структуру інформації, що використовується в системі, а також зв'язки між окремими її частинами.

Особливістю даного програмного додатку є те, що він не працює з великими обсягами даних. Основна інформація, яка зберігається, – це налаштування користувача та параметри роботи системи. Саме тому модель даних орієнтована не на складні взаємозв'язки, а на зручну організацію конфігурації.

Логічну структуру цих даних показано на рис. 6. Діаграма потрібна для того, щоб відокремити різні групи налаштувань і показати, як вони пов'язані між собою.

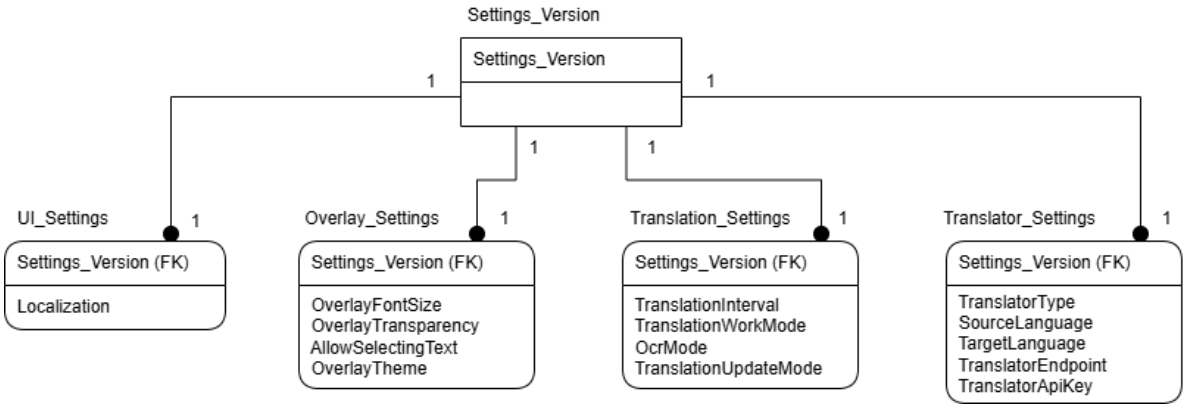

Рис. 6 Логічна модель даних системи

На діаграмі видно, що налаштування не зберігаються хаотично, а поділені на кілька логічних груп. Такий поділ спрощує подальшу реалізацію, оскільки кожна група відповідає за окрему частину роботи програми.

Основні сутності моделі:

1) UI_Settings: дана сутність містить параметри, що відповідають за інтерфейс користувача. До них належить, зокрема, мова локалізації. Це дозволяє адаптувати програму під потреби користувача.

2) Overlay_Settings: відповідає за налаштування відображення перекладу на екрані. Тут зберігаються такі параметри, як розмір шрифту, прозорість, тема оформлення та можливість взаємодії з текстом.

3) Translation_Settings: містить параметри, що визначають логіку роботи перекладу. Сюди входять інтервал оновлення, режим роботи перекладу та налаштування OCR.

4) Translator_Settings: описує параметри взаємодії з сервісом перекладу. Зокрема, визначає тип перекладача, мови перекладу, а також технічні параметри підключення, такі як endpoint та API-ключ.

На рис. 7 наведено, як ці дані зберігаються в програмі.

```
public class SettingsModel
{
    8 references
    public string Localization { get; set; } = "en";
    7 references
    public string SourceLanguage { get; set; } = "en";
    6 references
    public string TargetLanguage { get; set; } = "uk";
    7 references
    public int OverlayFontSize { get; set; } = 12;
    7 references
    public int OverlayTransparency { get; set; } = 80;
    7 references
    public bool OverlayAllowSelectingText { get; set; } = false;
    7 references
    public string OverlayTheme { get; set; } = "dark";
    7 references
    public int TranslationInterval { get; set; } = 10;
    7 references
    public string TranslationWorkMode { get; set; } = "translation";
    7 references
    public string TranslationOcrMode { get; set; } = "block";
    8 references
    public string TranslationUpdateMode { get; set; } = "onetime";
    6 references
    public string Translator { get; set; } = Translators.GoogleFreeTranslator.ToString();
    10 references
    public string LibreTranslatorEndpoint { get; set; } = "http://localhost:5000";
    5 references
    public string LibreTranslatorApikey { get; set; } = "";
}
```

Рис. 7 Модель даних налаштувань

На рис. 8 показано, як ці параметри записуються у JSON-файл.

```
string jsonString = JsonSerializer.Serialize(_settings, options);

File.WriteAllText(_filePath, jsonString);
```

Рис. 8 Збереження даних у JSON-файл

2.2.1 Зв’язки між сутностями. Усі зазначені сутності пов’язані з Settings_Version через зовнішній ключ. Такий підхід дозволяє об’єднати всі налаштування в єдину логічну структуру.

Фактично це означає, що система працює з однією конфігурацією, яка складається з кількох логічно розділених частин. Це спрощує як доступ до даних, так і їх зміну.

2.2.2 Особливості логічної моделі. Побудована модель має кілька важливих особливостей:

- чітке розділення налаштувань за їх призначенням;
- відсутність зайвих залежностей між сутностями;
- простота структури, що відповідає задачам системи;
- можливість розширення (наприклад, додавання нових параметрів без зміни всієї структури).

Цю модель не потрібно сприймати як готову структуру файлу або бази даних. Вона показує саме логічний поділ налаштувань, а вже фізичне збереження реалізується простіше – через JSON-файл.

2.3 Діаграма класів

Щоб показати, з яких класів складається програмна система і які зв’язки між ними передбачені, було побудовано діаграму класів, яка наведена на рис. 9. Вона допомагає пов’язати логіку предметної області з реальною структурою програмного коду.

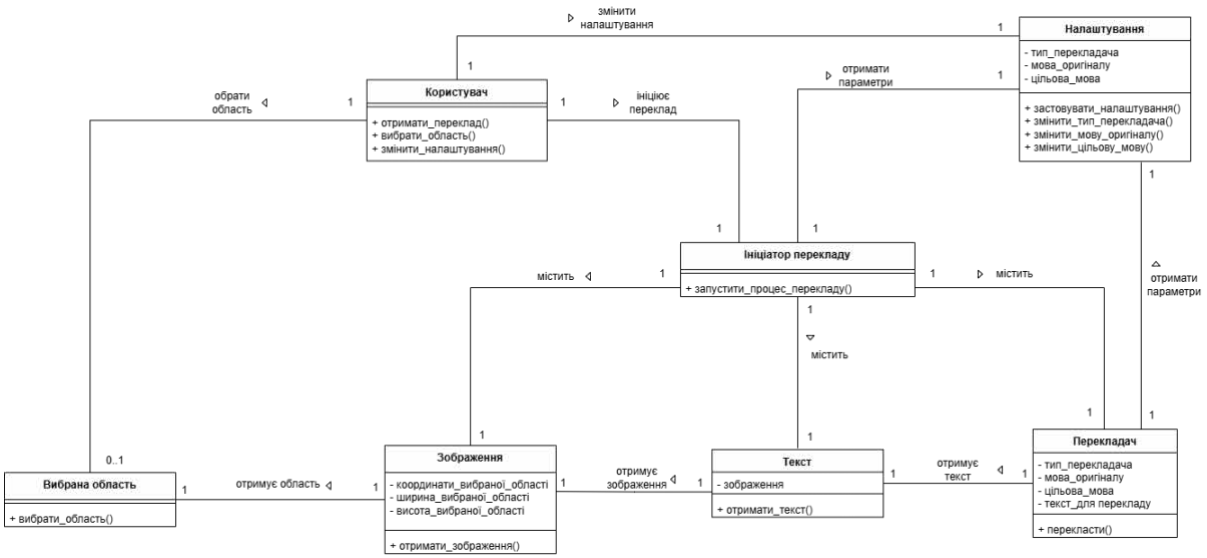

Рис. 9 Діаграма класів програмної системи

Слід зазначити, що дана діаграма поєднує як логічні елементи предметної області, так і класи, що безпосередньо реалізовані у програмному коді. Частина класів використовується для опису загального процесу (зображення, текст,

переклад), тоді як інші відповідають за реалізацію функціональності (сервіси, адаптери, фабрики). Такий підхід дозволяє показати зв’язок між предметною областю та її програмною реалізацією.

На діаграмі можна виділити кілька основних груп класів.

Клас Користувач виступає як ініціатор дій у системі. Він виконує вибір області на екрані, ініціалізує переклад та змінює налаштування.

Клас Вибрана область відповідає за вибір частини екрану. Після вибору координати передаються далі для отримання зображення.

Клас Зображення містить параметри області (координати, ширину та висоту) та забезпечує отримання зображення з екрану. Отримане зображення передається до класу Текст, який виконує розпізнавання тексту.

Клас Перекладач відповідає за переклад тексту. Він містить такі параметри, як мова оригіналу, цільова мова та текст для перекладу, а також метод виконання перекладу.

Клас Налаштування використовується для зберігання параметрів роботи системи та їх зміни.

Клас Ініціатор перекладу відповідає за запуск процесу перекладу та координацію основних етапів.

Важливою особливістю є використання інтерфейсів (IOcr, ITranslator), що дозволяє зменшити залежність між компонентами системи та забезпечує можливість заміни конкретних реалізацій без зміни основної логіки.

2.4 Діаграми кооперацій

Кооперація перекладача. Оскільки система може працювати з різними сервісами перекладу, окремо було розглянуто кооперацію перекладача. На рис. 10 показано, як сервіс перекладу взаємодіє з інтерфейсом, фабрикою та конкретними адаптерами.

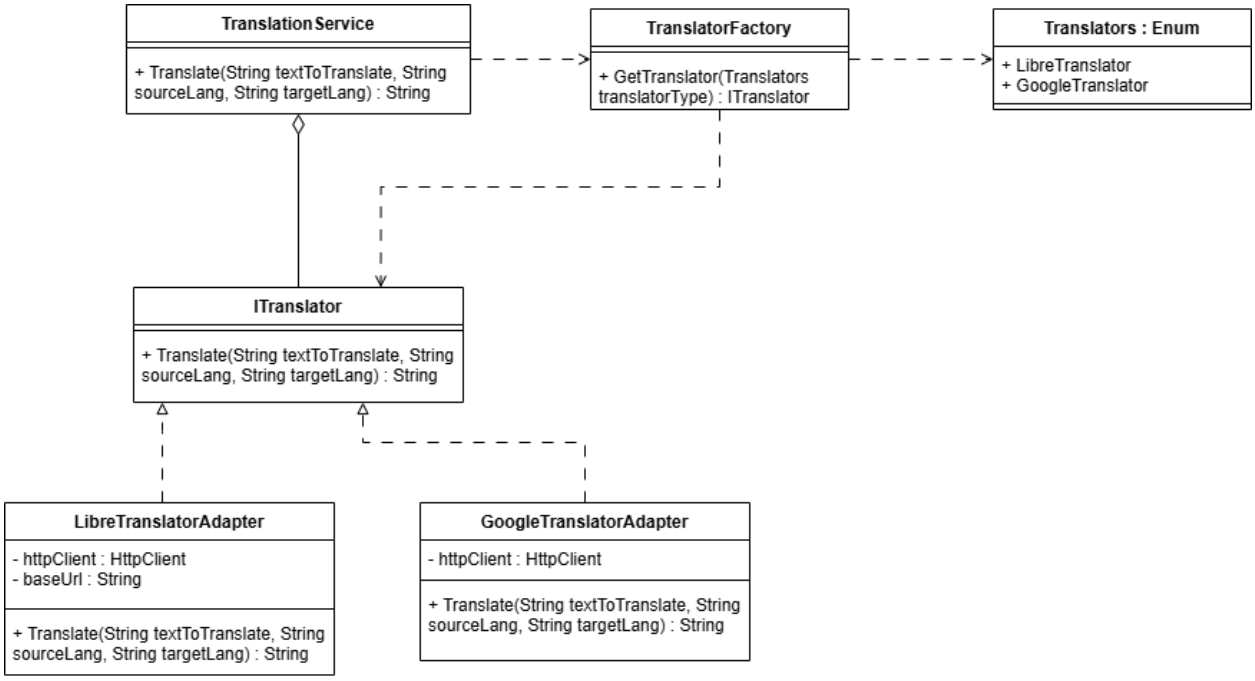

Рис. 10 Кооперація перекладача

Основним класом є TranslationService, який виконує переклад тексту. Для забезпечення гнучкості використовується інтерфейс ITranslator, що визначає загальний метод перекладу.

Клас TranslatorFactory відповідає за створення конкретного перекладача залежно від обраного типу. Для цього використовується перелік Translators, який містить доступні варіанти.

Класи LibreTranslatorAdapter та GoogleTranslatorAdapter реалізують інтерфейс ITranslator та виконують взаємодію з відповідними сервісами перекладу. Такий підхід дозволяє додавати нові перекладачі без зміни основної логіки програми.

Кооперація захоплення та розпізнавання тексту. Друга кооперація стосується тієї частини системи, яка готує текст для подальшого перекладу. На рис. 11 показано, як вибрана область екрану перетворюється спочатку на зображення, а потім на текст.

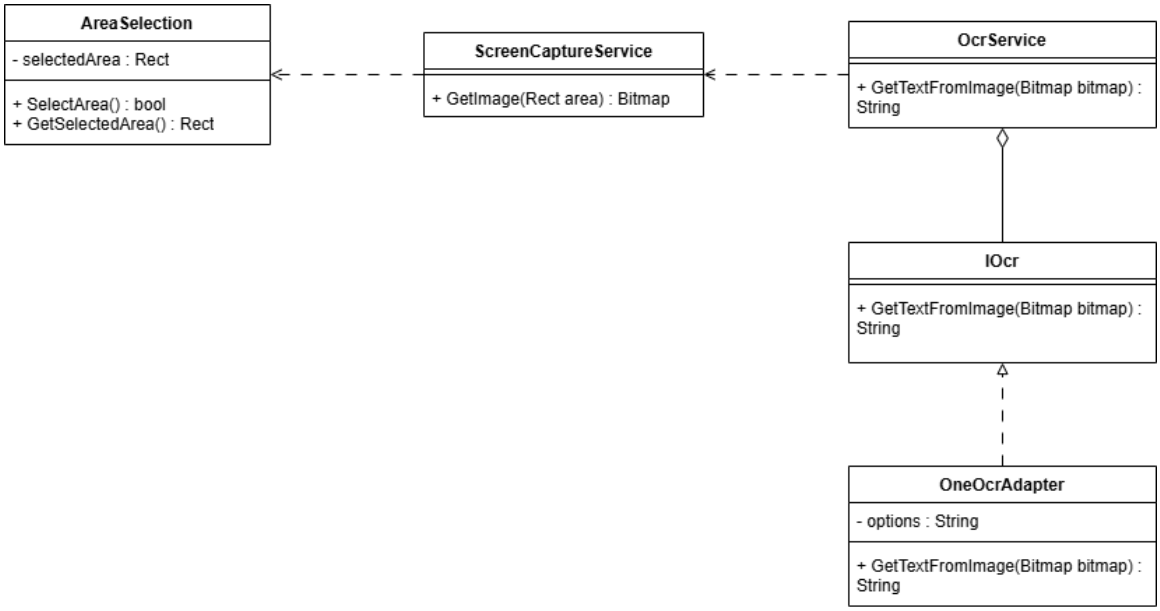

Рис. 11 Кооперація захоплення та розпізнавання тексту

Клас AreaSelection відповідає за вибір області екрану та зберігає її параметри. Далі ці дані передаються до ScreenCaptureService, який отримує зображення.

Отримане зображення передається до OcrService, який виконує розпізнавання тексту. Для цього використовується інтерфейс IOcr, що дозволяє використовувати різні реалізації OCR.

Клас OneOcrAdapter реалізує інтерфейс IOcr та забезпечує конкретну реалізацію розпізнавання тексту.

Кооперація виконання перекладу. Центральним елементом є клас TranslationProcess, який координує всі етапи обробки. Клас SettingsManager забезпечує отримання параметрів системи, таких як тип перекладача та мови.

Процес виконується послідовно: вибір області, отримання зображення, розпізнавання тексту, передача тексту до TranslationService, отримання перекладу. Такий підхід дозволяє чітко розділити відповідальність між компонентами та спростити підтримку системи.

На рис. 12 показано вже не окремий модуль, а загальну кооперацію під час виконання перекладу. Ця діаграма об’єднує вибір області, отримання тексту, звернення до сервісу перекладу та відображення результату.

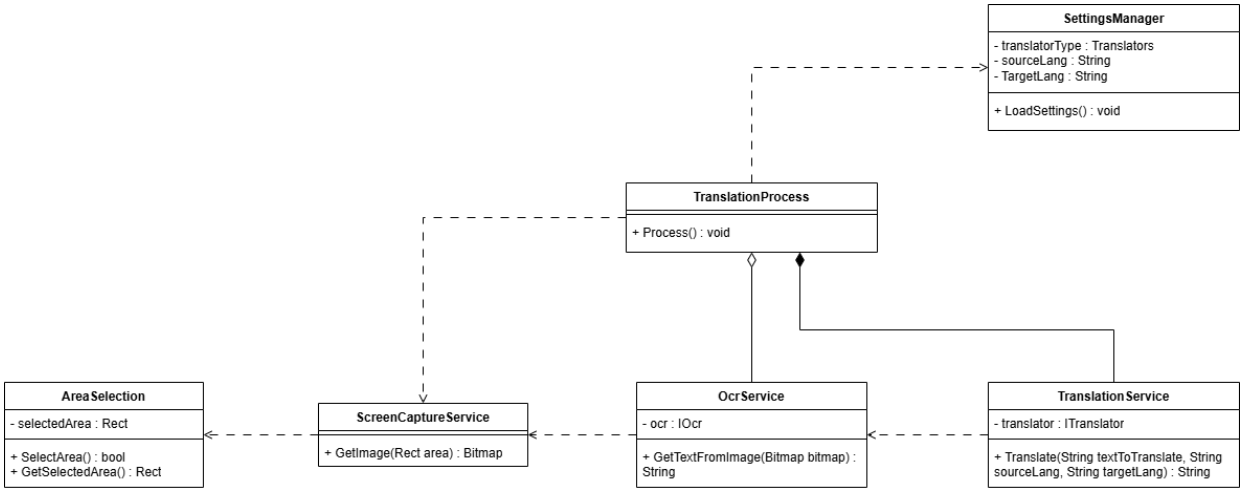

Рис. 12 Кооперація виконання перекладу

2.5 Діаграма пакетів

Структура проєкту також була розглянута на рівні пакетів. Це потрібно для того, щоб показати, як програмний код поділений за призначенням і які частини системи залежать одна від одної.

Як видно з діаграми пакетів, яка наведена на рис. 13, програмний додаток поділений на кілька основних пакетів: ui, services, adapters, settings, resources та models.

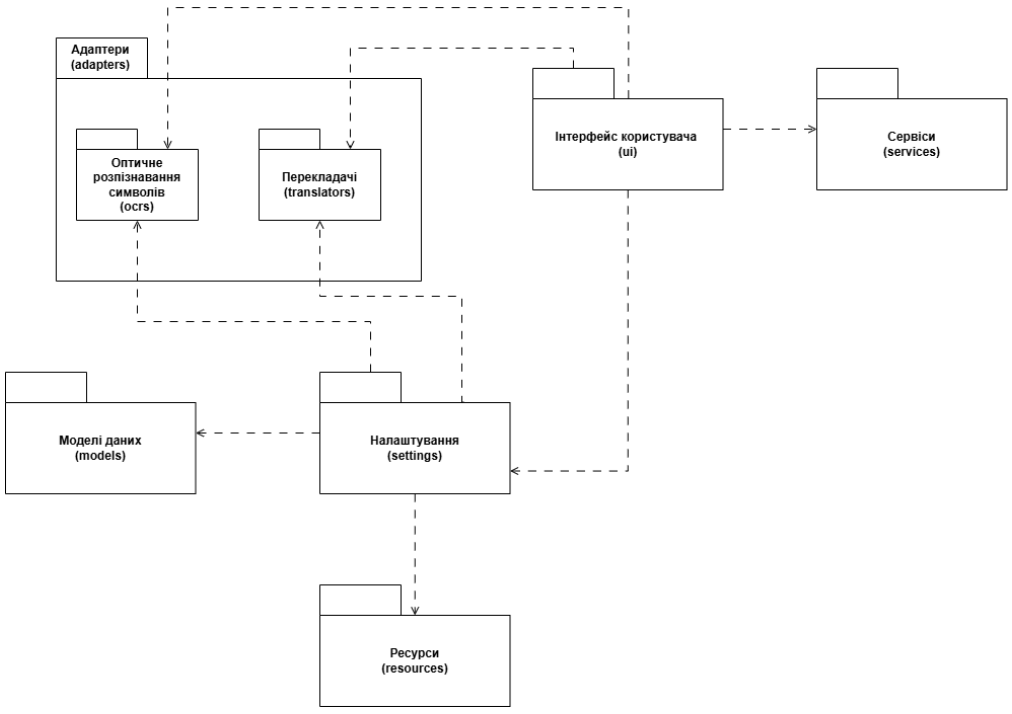

Рис. 13 Діаграма пакетів системи

Пакет ui містить класи користувацького інтерфейсу, зокрема MainWindow, OverlayWindow та інші допоміжні вікна. Основна логіка керування програмою реалізована саме в MainWindow, тому цей пакет має залежності від інших частин системи.

Пакет services містить класи, що реалізують основну обробку даних. До нього входять ScreenCaptureService, OcrService та TranslationService, які відповідають за отримання зображення, розпізнавання тексту та переклад відповідно.

Пакет adapters призначений для абстрагування від конкретних реалізацій зовнішніх сервісів. Він містить два вкладені пакети: ocrs та translators.

У пакеті ocrs знаходяться інтерфейс IOcr та його реалізація (OneOcrAdapter). Це дозволяє змінювати OCR-модуль без впливу на інші частини системи.

Пакет translators містить інтерфейс ITranslator, адаптери перекладачів, перелік Translators та клас TranslatorFactory. Така структура дозволяє гнучко обирати та змінювати сервіси перекладу.

Пакет settings містить класи SettingsManager та LocalizationManager, які відповідають за роботу з налаштуваннями та локалізацією програми.

Пакет resources містить ресурси програми, зокрема файли локалізації та графічні елементи інтерфейсу.

Пакет models містить допоміжні структури даних, такі як модель відповіді від перекладача та моделі налаштувань.

Такий поділ пакетів робить структуру проєкту більш зрозумілою. Наприклад, код інтерфейсу не змішується з адаптерами перекладачів, а робота з налаштуваннями винесена в окремий пакет.

2.6 Діаграма компонентів

Для більш детального розуміння структури програмного додатку та взаємодії його складових було побудовано діаграму компонентів. На відміну від діаграми класів, яка відображає внутрішню структуру об’єктів, діаграма

компонентів дозволяє розглянути систему на вищому рівні абстракції – як набір взаємопов’язаних модулів.

Використання такого підходу дає можливість чітко виділити окремі частини системи, визначити їх призначення та зрозуміти, як саме відбувається обмін даними між ними. Це особливо важливо для програмного додатку, який взаємодіє із зовнішніми сервісами та використовує сторонні бібліотеки.

У даному випадку діаграма компонентів відображає не лише внутрішню структуру програмного продукту, але й взаємодію із зовнішніми залежностями, такими як OCR-бібліотека та сервіси перекладу. Такий підхід дозволяє показати повний ланцюг обробки даних – від отримання зображення до відображення перекладеного тексту користувачу.

Крім того, діаграма дає змогу оцінити ступінь незалежності окремих модулів, а також визначити можливості для подальшого розширення системи, наприклад шляхом додавання нових перекладачів або заміни OCR-модуля.

Як видно з діаграми, яка наведена на рис. 14, центральним компонентом системи є виконуваний файл OnScreenTranslator.exe, який містить основну логіку роботи програми. У його межах реалізовані всі ключові модулі, що забезпечують процес перекладу тексту з екрану.

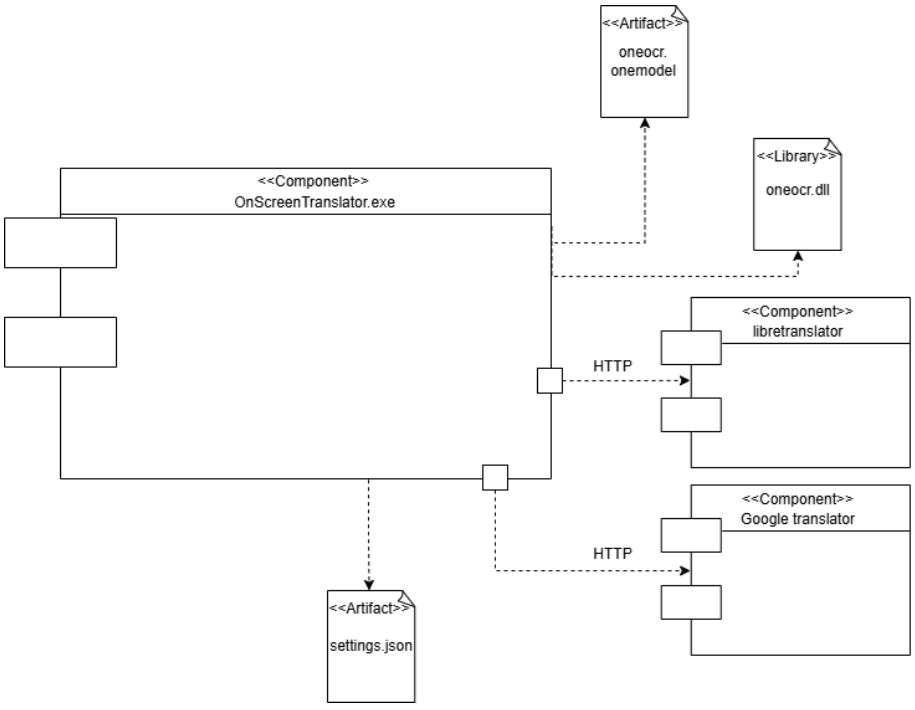

Рис. 14 Діаграма компонентів програмного додатку

Всередині основного компоненту можна виділити кілька підсистем.

Підсистема отримання та обробки зображення включає компоненти ScreenCaptureService та OcrService. Перший відповідає за отримання зображення з обраної області екрану, після чого зображення передається до OcrService для розпізнавання тексту. Для забезпечення гнучкості використовується інтерфейс IOcr, який реалізується через компонент OneOcrAdapter.

Компонент OneOcrAdapter взаємодіє із зовнішньою бібліотекою oneocr.dll, яка забезпечує безпосереднє розпізнавання тексту. Таким чином, система не залежить від конкретної реалізації OCR і може бути розширена у майбутньому.

Підсистема перекладу реалізована через компонент TranslationService, який працює через інтерфейс ITranslator. Це дозволяє використовувати різні сервіси перекладу без зміни основної логіки програми.

Реалізація перекладачів представлена компонентами LibreTransAdapter та GoogleTransAdapter, які взаємодіють із зовнішніми сервісами перекладу через HTTP-запити. На діаграмі це показано як зв’язок із компонентами libretranslator та Google translator.

Користувацький інтерфейс представлений компонентами MainWindow та OverlayWindow. MainWindow відповідає за керування програмою, вибір області та запуск перекладу, тоді як OverlayWindow використовується для відображення перекладеного тексту поверх інших вікон.

Компонент SettingsManager відповідає за роботу з налаштуваннями системи. Збереження параметрів виконується у файлі settings.json, що дозволяє зберігати конфігурацію між запусками програми.

Загалом взаємодія компонентів виглядає наступним чином: користувач через MainWindow ініціює процес перекладу, після чого система отримує зображення, виконує розпізнавання тексту, передає його до модуля перекладу та відображає результат у OverlayWindow.

З діаграми видно, що основна логіка зосереджена в додатку OnScreenTranslator.exe, але окремі задачі винесені в самостійні компоненти. Це

спрощує підтримку програми і залишає можливість у майбутньому замінити OCR або сервіс перекладу без повної переробки системи.

3 РОЗРОБКА ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Загальна характеристика розробки програмного забезпечення

Після завершення етапу проєктування наступним кроком є безпосередня реалізація програмного додатку. На цьому етапі прийняті раніше проєктні рішення переходять у програмний код, визначаються способи обробки даних і уточнюється взаємодія між окремими частинами системи.

Розробка виконувалась з урахуванням побудованої архітектури, яка передбачає поділ системи на окремі модулі. До них належать інтерфейс користувача, сервіси обробки даних, адаптери для взаємодії із зовнішніми сервісами, модуль налаштувань та компоненти для відображення результату перекладу. Такий поділ дозволяє не змішувати різні частини логіки в одному місці та робить структуру програми більш зрозумілою.

Особливістю реалізації є те, що основна логіка роботи системи побудована навколо послідовної взаємодії кількох сервісів. Спочатку виконується захоплення зображення з обраної області екрану, потім зображення передається на розпізнавання тексту, після цього отриманий текст надсилається до сервісу перекладу, а результат виводиться в overlay-вікні. Таким чином, кожен модуль відповідає за окремий етап обробки.

Окрему увагу під час розробки було приділено роботі із зовнішніми сервісами. Для цього використано підхід із застосуванням інтерфейсів та адаптерів. Завдяки цьому основна логіка програми не прив’язується до конкретного перекладача або OCR-модуля. У майбутньому це спрощує заміну окремих компонентів або додавання нових реалізацій.

Також при реалізації враховувались вимоги до продуктивності та зручності використання. У програмі використовується асинхронна обробка даних, що

дозволяє не блокувати інтерфейс під час виконання OCR або очікування відповіді від перекладача. Крім того, реалізовано підтримку гарячих клавіш, за допомогою яких користувач може швидко запускати основні дії без постійного звернення до головного вікна програми.

У межах цього розділу розглядаються основні етапи реалізації програмного додатку: організація збереження даних, розробка інформаційної бази, вибір інструментів розробки, а також алгоритмізація та програмування основних модулів системи.

3.2 Система управління інформаційною базою

У процесі розробки програмного додатку одним із важливих завдань було визначення способу організації та збереження даних, які використовуються системою під час роботи. На відміну від класичних інформаційних систем, у даному випадку обсяг даних є відносно невеликим і не потребує складної обробки або зберігання великих масивів інформації.

Основні дані, які використовуються у програмі, пов’язані з налаштуваннями користувача та параметрами роботи системи. До них належать:

- мова інтерфейсу;
- вихідна та цільова мови перекладу;
- параметри overlay-вікна (розмір шрифту, прозорість, тема);
- налаштування процесу перекладу (інтервал оновлення, режими роботи);
- параметри підключення до сервісів перекладу (тип перекладача, endpoint, API-ключ).

З урахуванням того, що ці дані не змінюються дуже часто та не потребують складних запитів або багатокористувацького доступу, було прийнято рішення не використовувати повноцінну систему керування базами даних. Використання СУБД у даному випадку призвело б до ускладнення реалізації та збільшення ресурсних витрат без суттєвих переваг.

Як альтернатива було обрано зберігання даних у форматі JSON. Такий підхід є достатньо поширеним для збереження конфігураційних параметрів у десктопних додатках, оскільки дозволяє організувати структуру даних у зрозумілому вигляді та не потребує використання додаткових бібліотек або серверної частини.

JSON-файл виконує роль локальної інформаційної бази, яка завантажується під час запуску програми. При зміні налаштувань користувачем відповідні значення оновлюються у пам’яті, після чого зберігаються у файл. Таким чином забезпечується збереження стану програми між її запусками.

Робота з інформаційною базою реалізована через спеціалізований клас SettingsManager, який виконує такі функції:

- зчитування даних з JSON-файлу при запуску програми;
- перетворення даних у внутрішні структури;
- надання доступу до налаштувань іншим компонентам системи;
- збереження змінених параметрів у файл.

Використання окремого класу для роботи з даними дозволяє ізолювати цю логіку від інших частин програми та уникнути дублювання коду. Крім того, це спрощує можливу зміну способу зберігання даних у майбутньому.

Для цього проєкту такого способу збереження достатньо, оскільки програма працює переважно з налаштуваннями, а не з великими наборами даних.

3.3 Розробка інформаційної бази

Фізична структура збереження налаштувань показана на рис. 15. На відміну від логічної моделі, тут уже відображено те, як дані фактично організовані у вигляді структури AppSettings. Враховуючи характер задачі, вся інформація зберігається у вигляді одного JSON-файлу, який містить конфігураційні параметри системи.

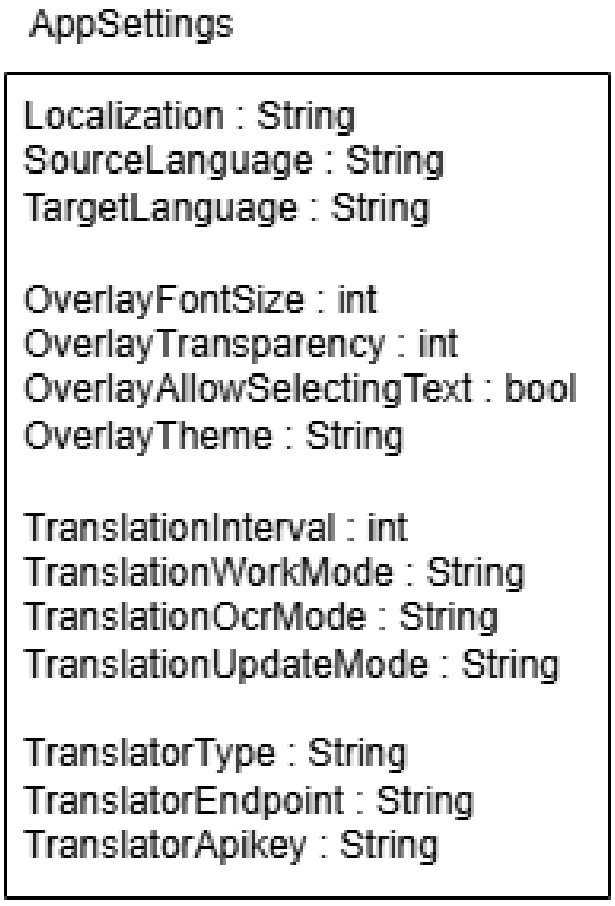

Рис. 15 Структура інформаційної моделі (AppSettings)

Як видно з рисунка, всі параметри об’єднані в одну сутність AppSettings, яка представляє собою сукупність налаштувань програми. Такий підхід дозволяє спростити доступ до даних та уникнути зайвої складності у структурі.

У межах цієї моделі можна виділити кілька логічних груп параметрів.

Перша група – це параметри, пов’язані з інтерфейсом користувача. Вони включають мову локалізації та визначають, як саме відображаються елементи програми. Завдяки цьому користувач може працювати з додатком у зручному для себе мовному середовищі.

Друга група – це налаштування overlay-вікна. До них належать розмір шрифту, рівень прозорості, тема оформлення та можливість взаємодії з текстом. Ці параметри впливають на зовнішній вигляд і зручність відображення перекладеного тексту.

Третя група – це параметри процесу перекладу. Вони включають інтервал оновлення перекладу, режим роботи (наприклад, безперервний або одноразовий),

а також налаштування OCR. Ці параметри визначають поведінку системи під час виконання основного функціоналу.

Окремо виділяється група параметрів, пов’язаних із перекладачем. Вона містить тип перекладача, адресу сервісу та API-ключ. Ці дані використовуються для встановлення з’єднання із зовнішніми сервісами перекладу та виконання запитів.

На фізичному рівні всі ці параметри зберігаються у JSON-файлі у вигляді єдиного об’єкта. Така структура є денормалізованим представленням логічної моделі, оскільки не передбачає поділу даних на окремі таблиці або сутності.

Використання такого підходу має ряд переваг:

- простота реалізації та підтримки;
- відсутність необхідності у використанні СУБД;
- швидкий доступ до даних;
- можливість легко змінювати або доповнювати структуру.

Разом з тим, для даного типу задачі відсутність нормалізації не є недоліком, оскільки система працює з обмеженим набором даних і не виконує складних операцій над ними.

У підсумку JSON-файл виконує роль невеликого локального сховища налаштувань. Для такого додатку це практичніше, ніж використання окремої бази даних.

3.4 Вибір інструментарію для створення прикладного програмного забезпечення

Під час розробки програмного додатку одним із важливих етапів є вибір інструментів та технологій, які будуть використовуватись для реалізації системи. Для реалізації додатку потрібно було обрати інструменти, які добре підходять саме для десктопної програми під Windows. У цьому проєкті важливими були

робота з вікнами, обробка зображень, асинхронні запити та можливість створення overlay-вікна.

Оскільки розроблюваний додаток є десктопним і орієнтований на операційну систему Windows, було доцільно обрати технології, які забезпечують глибоку інтеграцію з цією платформою.

Основною мовою програмування було обрано C#. Вона є однією з найпоширеніших мов для розробки десктопних додатків під Windows, має зрозумілий синтаксис, розвинену стандартну бібліотеку та підтримує сучасні підходи до програмування. Особливо важливою для даного проєкту є підтримка асинхронності (async/await), яка дозволяє реалізувати переклад у режимі, близькому до реального часу, без блокування інтерфейсу.

Для створення графічного інтерфейсу використано WPF (Windows Presentation Foundation). Ця технологія дозволяє створювати гнучкий і достатньо сучасний інтерфейс користувача, а також реалізовувати складні елементи відображення. Однією з ключових причин вибору WPF є можливість створення overlay-вікна, яке може відображатися поверх інших програм і не заважати роботі користувача. Крім того, WPF підтримує прив’язку даних та розділення логіки і представлення, що спрощує структуру програми.

Середовищем розробки було обрано Microsoft Visual Studio, яке забезпечує повний набір інструментів для розробки, включаючи підсвічування синтаксису, налагодження, аналіз коду та роботу з проєктами. Це спрощує розробку, оскільки в одному середовищі можна писати код, запускати програму, налагоджувати її та швидко знаходити помилки.

Для розпізнавання тексту використовується OCR (Optical Character Recognition). У проєкті застосовано адаптер, який реалізує інтерфейс IOcr. Такий підхід дозволяє відокремити конкретну реалізацію OCR від основної логіки системи. У разі необхідності можна замінити або додати інший механізм розпізнавання без зміни структури програми.

Для перекладу тексту використовується взаємодія із зовнішніми сервісами через HTTP-запити. У системі реалізовано інтерфейс ITranslator, який визначає

загальний контракт для перекладачів. Конкретні реалізації (наприклад, LibreTranslate або Google Translate) виконані у вигляді адаптерів. Для створення екземплярів перекладачів використовується патерн “фабрика” (TranslatorFactory), що дозволяє обирати необхідний сервіс у залежності від налаштувань користувача.

Для виконання запитів до зовнішніх API використовується стандартний клас HttpClient, який входить до складу платформи .NET. Його використання дозволяє виконувати асинхронні запити, обробляти відповіді сервера та забезпечувати стабільну взаємодію із сервісами перекладу.

Захоплення зображення з екрану реалізовано за допомогою окремого сервісу (ScreenCaptureService), який використовує можливості .NET для роботи з графікою. Це дозволяє отримувати зображення вибраної області екрану та передавати його до OCR-модуля для подальшої обробки.

Для організації процесу перекладу використовується сервісна архітектура. Основні операції розподілені між окремими класами, такими як:

- OcrService – відповідає за розпізнавання тексту;
- TranslationService – виконує переклад;
- ScreenCaptureService – отримує зображення;
- SettingsManager – працює з налаштуваннями.

За такої структури кожен сервіс відповідає за свою частину роботи, тому код легше читати й підтримувати.

Важливим аспектом реалізації є використання асинхронного програмування. У програмі застосовуються Task, async/await та CancellationToken, що дозволяє:

- виконувати переклад у фоновому режимі;
- уникати блокування інтерфейсу;
- коректно зупиняти процес перекладу;
- контролювати виконання довготривалих операцій.

Крім того, у програмі використано можливості операційної системи Windows для роботи з гарячими клавішами через WinAPI. Це дозволяє користувачу швидко виконувати основні дії, такі як запуск перекладу або вибір області, без необхідності взаємодії з інтерфейсом.

Для збереження налаштувань використовується формат JSON, який є простим у використанні та не потребує додаткових інструментів. Це рішення добре підходить для зберігання конфігураційних даних і не ускладнює систему.

Обраний набір технологій добре підходить саме для цього типу задачі: WPF використовується для інтерфейсу, C# – для основної логіки, а зовнішні API – для перекладу.

3.5 Алгоритмізація та програмування програмних модулів

Одним із ключових етапів розробки програмного додатку є реалізація алгоритмів, які забезпечують виконання основного функціоналу системи. У даному випадку основною задачею є автоматичний переклад тексту з екрану користувача в режимі, близькому до реального часу.

Робота системи базується на взаємодії кількох окремих модулів. Один модуль відповідає за отримання зображення з екрану, інший – за розпізнавання тексту, ще один – за переклад, а overlay-вікно використовується для відображення результату поверх інших програм.

Після запуску перекладу система циклічно виконує обробку обраної області екрану. Спочатку отримується зображення, після чого з нього виділяється текст за допомогою OCR. Далі цей текст передається до модуля перекладу, а результат відображається користувачу. Завдяки цьому процес перекладу частково переноситься на програму, а не залишається повністю на користувачеві.

У межах даного підрозділу розглядаються блок-схема алгоритму, особливості реалізації інтерфейсної частини, бек-частини, асинхронної обробки та механізм збереження налаштувань.

3.5.1 Блок-схема алгоритму перекладу. Основну логіку перекладу доцільно подати у вигляді блок-схеми, оскільки алгоритм містить не лише послідовні дії, а й перевірки. На рис. 16 показано, як система поводиться залежно від наявності overlay-вікна, результату OCR та зміни тексту.

Рис. 16 Блок-схема алгоритму перекладу тексту з екрану

Відповідно до блок-схеми, спочатку система перевіряє, чи створене overlay-вікно. Якщо воно відсутнє, переклад не виконується, оскільки результат немає де відобразити. У такому випадку система переходить у режим очікування.

Якщо overlay-вікно створене, система переходить до перевірки перетину overlay з вибраною областю екрану. Це потрібно для того, щоб overlay не

потрапив у знімок екрану. Якщо перетин є, overlay тимчасово приховується, після чого виконується захоплення зображення.

Отримане зображення передається до OCR-модуля, який виконує розпізнавання тексту. Якщо текст не знайдено або він порожній, переклад не виконується, і система переходить до наступної ітерації циклу.

Після отримання тексту виконується перевірка, чи змінився він порівняно з попереднім результатом. Якщо текст залишився тим самим, повторний переклад не виконується. Це дозволяє зменшити кількість зайвих запитів до сервісів перекладу.

Якщо переклад необхідний, текст передається до сервісу перекладу. Отриманий результат відображається у overlay-вікні, після чого система очікує визначений інтервал часу та повторює алгоритм знову.

3.5.2 Візуальна реалізація алгоритму. Після запуску перекладу система переходить у режим активної роботи. Користувач бачить overlay-вікно, яке відображається поверх інших програм і містить перекладений текст.

При натисканні кнопки “Start” запускається цикл перекладу. Кнопка автоматично змінює свій стан на “Stop”, що дозволяє користувачу зрозуміти, що переклад уже активний.

У процесі роботи overlay-вікно автоматично оновлює текст відповідно до змін у вибраній області екрану. Якщо користувач натискає кнопку “Stop”, виконання циклу перекладу припиняється, а overlay перестає оновлюватися.

Такий підхід робить взаємодію з програмою більш зрозумілою та дозволяє швидко керувати процесом перекладу.

3.5.3 Реалізація алгоритму на фронт-частині. Фронт-частина програмного додатку реалізована за допомогою технології WPF та відповідає за взаємодію користувача з системою.

Основним елементом керування є кнопка типу ToggleButton, яка використовується для запуску та зупинки перекладу. Особливістю цього елемента є підтримка двох станів – Checked та Unchecked. Завдяки цьому для запуску і зупинки використовується один обробник подій.

На рис. 17 показана реалізація описаної кнопки.

```
<!-- Start/stop translation button -->
<ToggleButton Grid.Column="1" Grid.Row="0" x:Name="TlgBtnStartStopTranslation" Height="45"
              Width="100" HorizontalAlignment="Right" VerticalAlignment="Center" Margin="0 0 10 0"
              Checked="StartStopTranslation" Unchecked="StartStopTranslation" IsEnabled="False">
    <ToggleButton.Style>
        <Style TargetType="ToggleButton" BasedOn="{StaticResource {x:Type ToggleButton}}" >
            <Setter Property="Content" Value="{Binding Start}" />
            <Style.Triggers>
                <Trigger Property="IsChecked" Value="True">
                    <Setter Property="Content" Value="{Binding Stop}" />
                </Trigger>
            </Style.Triggers>
        </Style>
    </ToggleButton.Style>
</ToggleButton>
```

Рис. 17 Реалізація кнопки запуску та зупинки перекладу

При натисканні кнопки викликається метод StartStopTranslation, який аналізує поточний стан кнопки. Якщо кнопка переходить у стан Checked, запускається процес перекладу. Якщо ж вона переходить у стан Unchecked, цикл перекладу завершується.

Зміна тексту кнопки реалізована за допомогою механізму прив’язки даних (Binding) та тригерів (Trigger). Це дозволяє автоматично оновлювати інтерфейс без додаткового програмного коду.

3.5.4 Реалізація логіки на бек-частині. Бек-частина системи реалізує основний алгоритм перекладу тексту з екрану, код алгоритму можна переглянути в Додатку А. Вона працює у вигляді циклу, який постійно виконує отримання зображення, розпізнавання тексту та його переклад.

Перед початком обробки система перевіряє наявність overlay-вікна. Якщо overlay відсутній, виконання тимчасово призупиняється для економії ресурсів. Окремо реалізовано перевірку, чи не перекриває overlay область захоплення. Якщо перекриття є, overlay тимчасово приховується перед створенням знімка екрану. Це дозволяє уникнути ситуації, коли система перекладає власний переклад.

Після отримання зображення воно передається до OCR-сервісу. У випадку блочного режиму додатково прибираються переноси рядків, щоб текст перекладався як один суцільний блок. Далі система перевіряє, чи змінився текст або параметри перекладу. Якщо текст залишився тим самим, переклад повторно

не виконується. Такий підхід дозволяє зменшити кількість зайвих запитів до API перекладача та знизити навантаження на систему.

Після отримання тексту виконується переклад через TranslationService. Отриманий результат передається в overlay-вікно та відображається користувачу через Dispatcher.Invoke, оскільки оновлення інтерфейсу виконується з фонового потоку. На рис. 18 показано фрагмент алгоритму процесу перекладу.

```
// if overlay is hidden do not spend resources to translate anything
if (_overlayWindow == null)
{
    await Task.Delay(400);
    continue;
}

Bitmap image = ScreenCaptureService.GetImage(_selectedScreenArea.Value);

// getting text from image
string text = _ocrService.GetTextFromImage(image).Replace("\r\n", " ");
image.Dispose();

// check if text has to be translated
if (!string.IsNullOrEmpty(text) && (!_previousText.Equals(text) ||
    !_previousSourceLang.Equals(source) || !_previousTargetLang.Equals(target)))
{
    // update previous data
    _previousText = text;
    _previousSourceLang = source;
    if (isOcrMode)
        _previousTargetLang = target = source;
    else
        _previousTargetLang = target;

    // translate text if source and target languages are different
    string translated = source == target ? text : await _translationService.TranslateAsync(
        text, source, target, apikey
    );
    _previousTranslatedText = translated;

    // update text in overlay
    Dispatcher.Invoke(() =>
    {
        _overlayWindow?.TxtOverlay.Visibility = Visibility.Visible;
        _overlayWindow?.TxtOverlay.Text = translated;
    });
```

Рис. 18 Фрагмент реалізації логіки перекладу

3.5.5 Асинхронна обробка даних. Однією з важливих особливостей реалізації системи є використання асинхронної обробки даних. Це необхідно для того, щоб інтерфейс програми залишався активним навіть під час виконання довготривалих операцій.

У процесі роботи система постійно виконує:

- отримання зображення з екрану;

- OCR-обробку;
- надсилання запитів до сервісів перекладу;
- оновлення overlay-вікна.

Деякі з цих операцій можуть займати певний час, особливо під час роботи з мережею. Якщо виконувати їх у головному потоці, інтерфейс програми почав би “зависати”, а користувач не зміг би взаємодіяти із системою.

Щоб інтерфейс не зависав під час OCR або очікування відповіді від перекладача, основний цикл винесено у фоновий потік. Завдяки цьому користувач може продовжувати працювати з програмою навіть під час виконання довготривалих операцій. Це особливо важливо для режиму постійного оновлення перекладу.

3.5.6 Реалізація збереження даних. Для збереження параметрів системи використовується JSON-файл. У ньому зберігаються налаштування користувача, зокрема мови перекладу, локалізація інтерфейсу та інші параметри роботи програми.

Використання JSON дозволяє спростити реалізацію механізму збереження даних та уникнути необхідності використання повноцінної системи керування базами даних. На рис. 19 показана реалізація збереження даних у додатку.

```
public void SaveSettings(MainWindow mainWindow)
{
    var options = new JsonSerializerOptions { WriteIndented = true };

    _settings.Localization = mainWindow.ComBoxLocalization.SelectedValue.ToString();
    _settings.SourceLanguage = mainWindow.ComBoxSourceLang.SelectedValue.ToString();
    _settings.TargetLanguage = mainWindow.ComBoxTargetLang.SelectedValue.ToString();

    string jsonString = JsonSerializer.Serialize(_settings, options);

    File.WriteAllText(_filePath, jsonString);
}
```

Рис. 19 Реалізація збереження налаштувань у JSON-файл

Метод SaveSettings() викликається при зміні параметрів або під час завершення роботи програми. Спочатку оновлюються значення параметрів, після чого об’єкт налаштувань серіалізується у JSON-рядок та записується у файл.

4 ТЕСТУВАННЯ ТА ВПРОВАДЖЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Організація тестування програмного додатку

Після завершення розробки програмного додатку було виконано його тестування. Основною метою тестування була перевірка працездатності системи, правильності виконання основних функцій та відповідності поставленим вимогам.

Оскільки розроблений додаток працює з кількома різними процесами, тестування проводилось не лише для окремих функцій, а й для повного сценарію роботи системи. Перевірялась взаємодія між інтерфейсом користувача, модулем захоплення зображення, OCR-модулем, сервісом перекладу та overlay-вікном.

Під час тестування основна увага була приділена таким функціям:

- вибір області екрану для перекладу;
- захоплення зображення з обраної області;
- розпізнавання тексту за допомогою OCR;
- переклад отриманого тексту;
- відображення результату в overlay-вікні;
- запуск і зупинка процесу перекладу;
- збереження та відновлення налаштувань;
- робота системи при зміні мов і параметрів перекладу.

На першому етапі перевірялась робота механізму вибору області екрану. Було встановлено, що користувач може виділити потрібну частину екрана, після чого координати цієї області зберігаються і використовуються під час подальшого захоплення зображення. Це є важливим етапом, оскільки саме вибрана область визначає, який текст буде передано на розпізнавання.

Далі проводилась перевірка захоплення зображення. Система повинна отримувати знімок саме тієї області, яку обрав користувач. Окремо було

перевірено ситуацію, коли overlay-вікно може перекривати область перекладу. У такому випадку програма тимчасово приховує overlay перед створенням знімку, щоб до OCR не потрапив уже перекладений текст.

Окремо тестувався модуль розпізнавання тексту. Перевірка виконувалась на різних типах зображень: текст у браузері, елементи інтерфейсу програм, субтитри у відео та текстові фрагменти з різною якістю відображення. У більшості випадків система коректно визначала текст, однак при низькій якості зображення, дрібному шрифті або нестандартному оформленні можливі помилки. Це є очікуваним обмеженням для OCR-технологій.

Також було перевірено роботу модуля перекладу. Система коректно формує запит до вибраного перекладача, передає розпізнаний текст і отримує перекладений результат. Додатково перевірялась поведінка програми у випадку, якщо перекладач тимчасово недоступний або виникає помилка під час звернення до API. У таких ситуаціях програма продовжує роботу та виводить вікно з помилкою. Приклад помилки наведено на рис. 20.

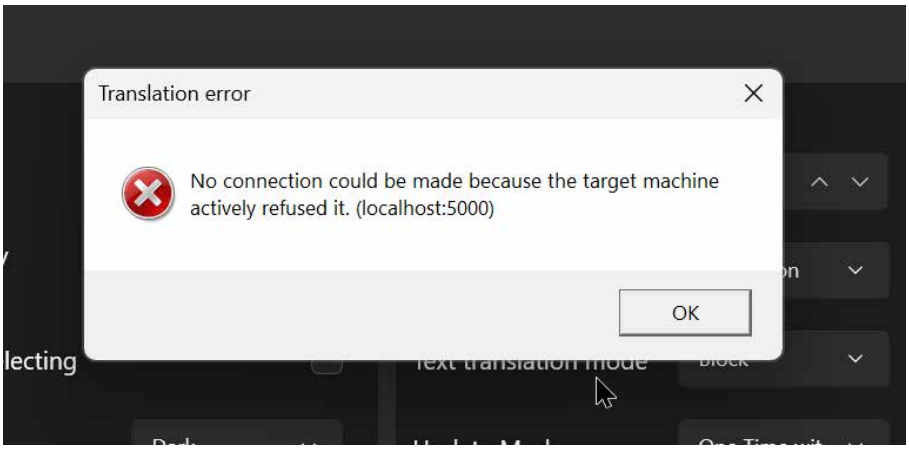

Рис. 20 Приклад помилки при роботі додатку

Важливою частиною тестування стала перевірка overlay-вікна. Воно повинно відображатися поверх інших програм, показувати перекладений текст і не заважати роботі користувача з іншими вікнами. Під час перевірки було підтверджено, що overlay коректно оновлює текст і може використовуватись під час роботи з браузером, відео або іншими програмами.

Окремо перевірялась робота з налаштуваннями. Було встановлено, що вибрані параметри зберігаються у JSON-файлі та відновлюються після

повторного запуску програми. Завдяки цьому користувачу не потрібно повторно налаштовувати мови, режим роботи, інтервал оновлення та інші параметри після кожного запуску додатку.

У процесі тестування також перевірялись нестандартні ситуації, зокрема відсутність тексту в обраній області, порожній результат OCR, відсутність інтернет-з’єднання, помилки під час звернення до сервісу перекладу та повторний запуск перекладу. У перевірених випадках система продовжувала роботу без критичних збоїв.

4.2 Перевірка основних сценаріїв роботи системи

Для більш наочного підтвердження працездатності програмного додатку було перевірено типовий сценарій його використання. Він включає запуск програми, створення overlay-вікна, вибір області екрану, налаштування параметрів перекладу, запуск процесу перекладу та отримання результату.

Після запуску програми відкривається головне вікно системи. У ньому розміщені основні елементи керування: вибір мов, вибір перекладача, кнопки запуску перекладу, створення overlay-вікна та вибору області екрану. Саме через головне вікно користувач керує основним процесом перекладу. Це вікно наведено на рис. 21.

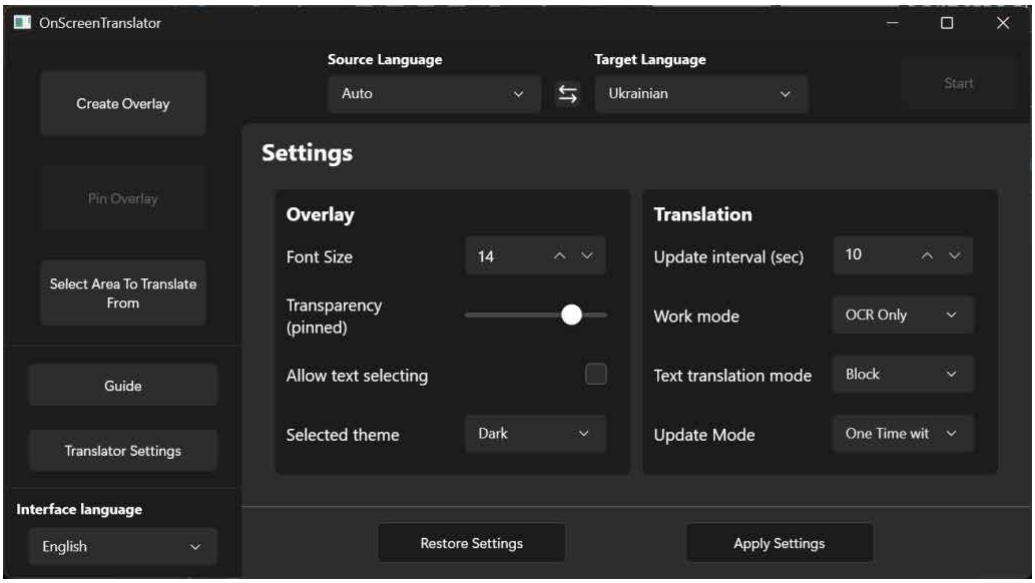

Рис. 21 Демонстрація інтерфейсу додатку

Перед запуском перекладу користувач повинен створити overlay-вікно, оскільки саме воно використовується для виведення перекладеного тексту. Без overlay-вікна, результату перекладу не буде де відображатися, тому цей етап є обов’язковим для повноцінної роботи системи. На рис. 22 наведено приклад, як виглядає overlay-вікно.

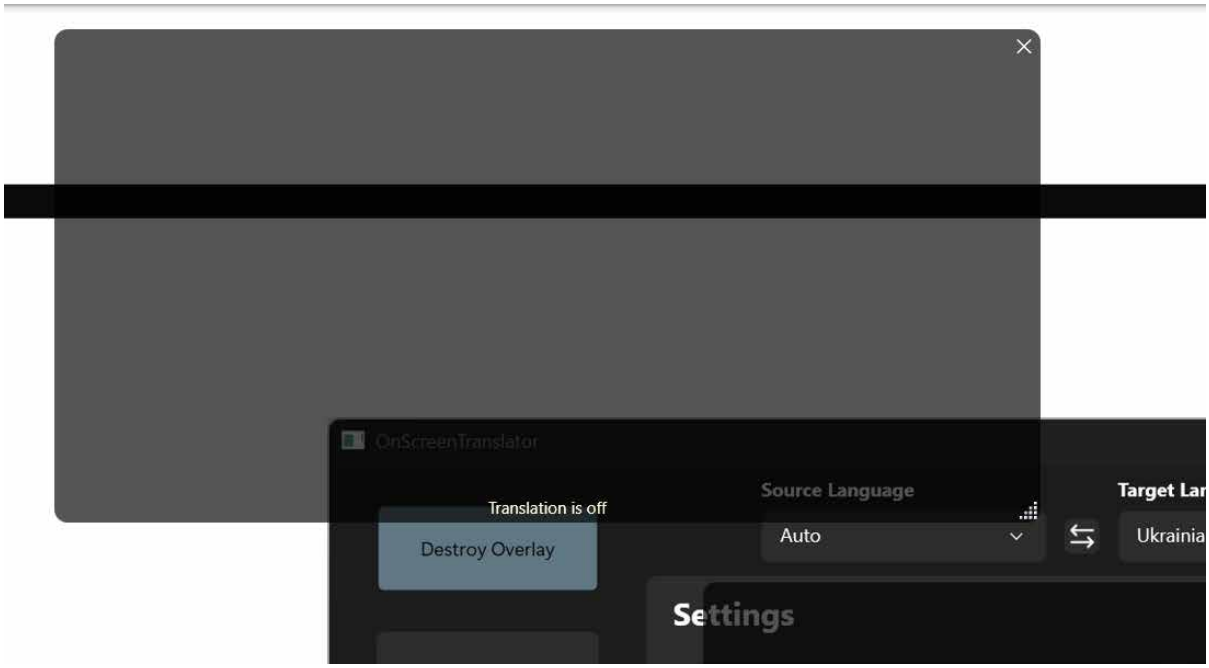

Рис. 22 Створення overlay-вікна

Наступним кроком є вибір області екрану, з якої буде виконуватись розпізнавання тексту. Під час тестування було перевірено, що користувач може виділити потрібний фрагмент екрана, після чого система коректно зберігає координати цієї області та використовує їх у процесі перекладу. На рис. 23 можна побачити вибір області екрану.

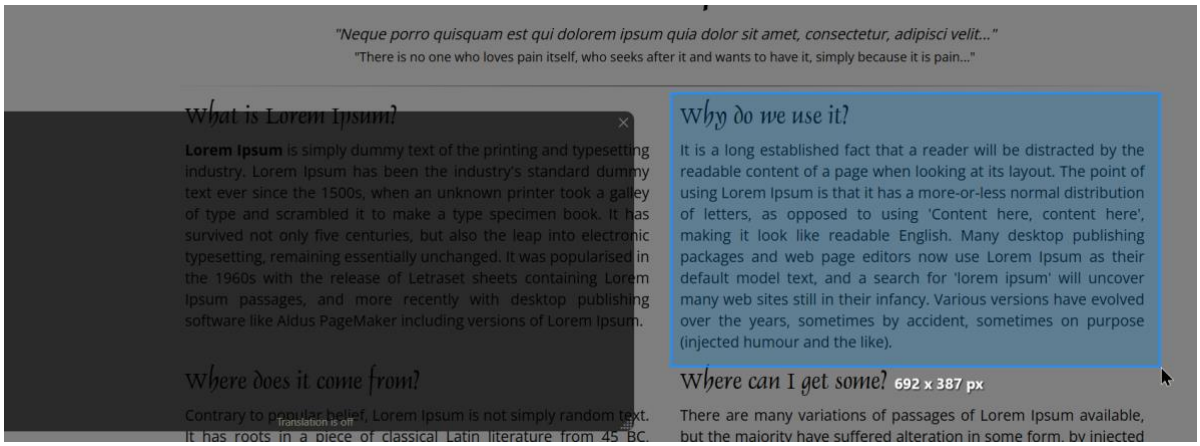

Рис. 23 Вибір області екрану для перекладу

Після вибору області користувач налаштовує параметри перекладу. До таких параметрів належать мова оригінального тексту, мова перекладу, сервіс перекладу, режим OCR та інтервал оновлення. Перевірка показала, що зміна цих параметрів впливає на подальшу роботу системи та враховується під час наступного циклу обробки. Приклад налаштувань наведено на рис. 24.

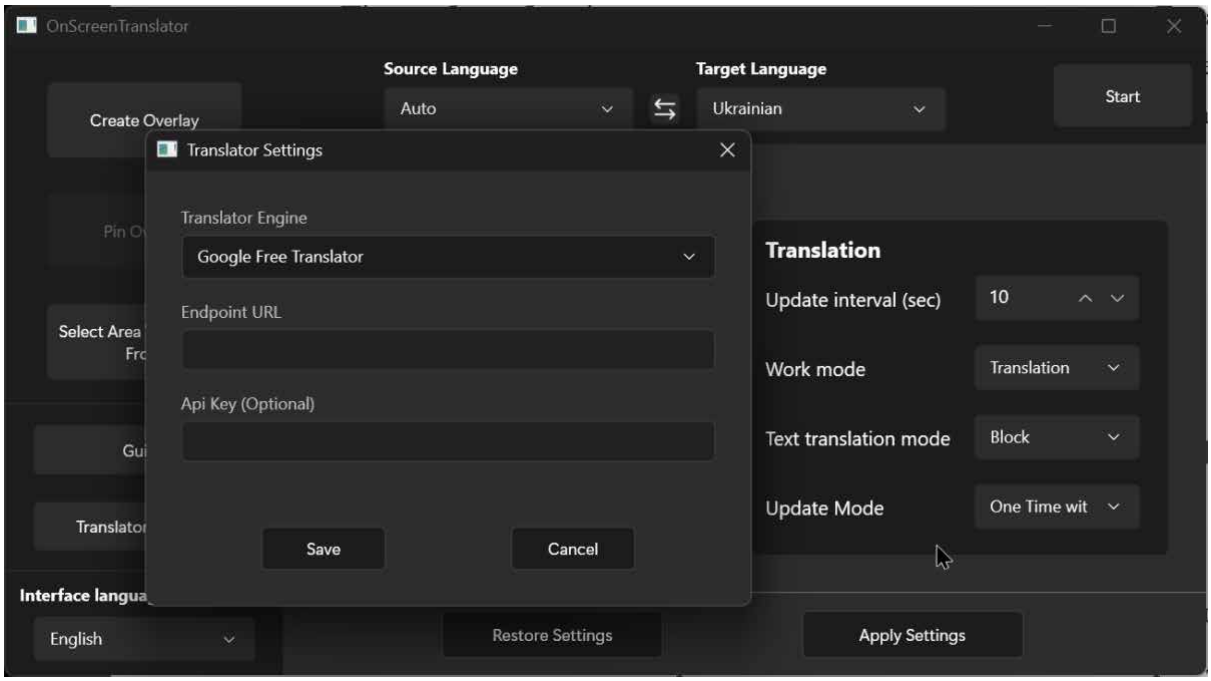

Рис. 24 Налаштування параметрів перекладу

Після завершення налаштування користувач запускає переклад за допомогою кнопки “Start” або гарячої клавіші. У цей момент система переходить у режим активної роботи: отримує зображення з вибраної області, розпізнає текст, передає його до перекладача та виводить результат в overlay-вікні.

Результат роботи програми відображається у вигляді перекладеного тексту поверх інших вікон. Це дозволяє користувачу не відкривати окремий перекладач і не копіювати текст вручну. Такий сценарій є зручним під час роботи з відео, іграми, програмами без локалізації або текстом, який неможливо виділити стандартними засобами.

Приклад такого перекладу можна побачити на рис. 25.

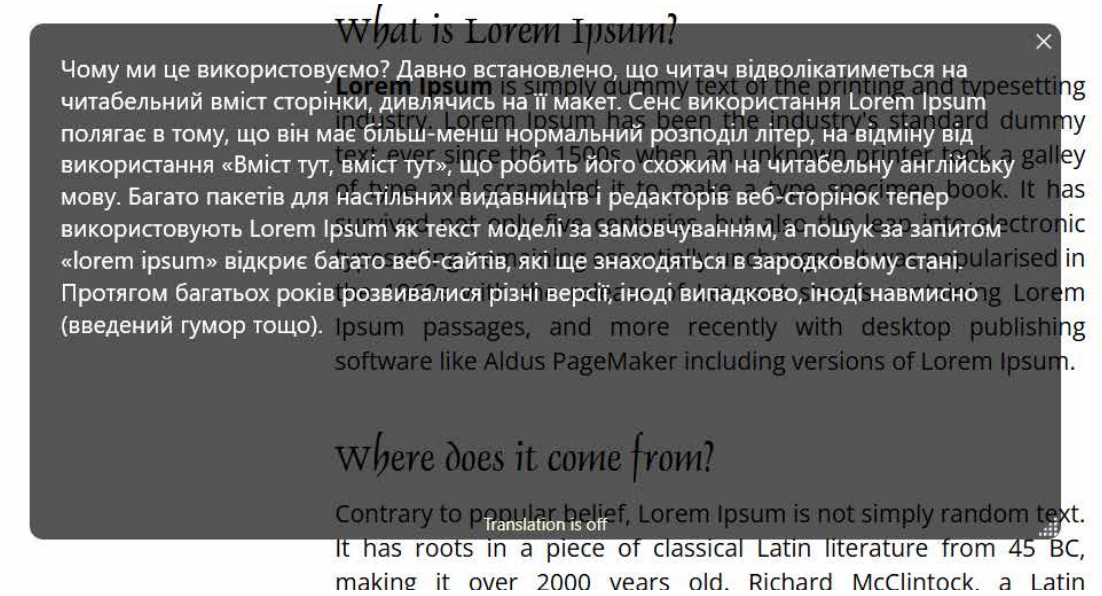

Рис. 25 Демонстрація успішного перекладу

Також було перевірено зупинку процесу перекладу. Після натискання кнопки “Stop” цикл обробки завершується, а overlay-вікно перестає оновлювати текст. Це підтверджує, що користувач може керувати роботою системи без необхідності закривати програму.

У результаті перевірки основних сценаріїв було встановлено, що додаток коректно виконує заявлені функції. Система дозволяє вибрати область екрану, отримати з неї текст, виконати переклад і відобразити результат у overlay-вікні. Крім того, програма зберігає налаштування користувача та коректно відновлює їх після повторного запуску.

4.3 Середовище розгортання програмного додатку та склад інсталяційного пакету

4.3.1 Середовище розгортання. Після перевірки основних сценаріїв роботи необхідно визначити середовище, у якому буде використовуватися програмний додаток. Оскільки система є десктопною, основна частина її компонентів розміщується безпосередньо на комп’ютері користувача. Водночас для виконання перекладу програма може звертатися до зовнішніх сервісів через інтернет.

Для опису фізичного розміщення основних компонентів було побудовано діаграму розгортання, яка наведена на рис. 26. Вона показує, які файли знаходяться на комп’ютері користувача, які додаткові компоненти потрібні для роботи OCR, а також з якими зовнішніми сервісами взаємодіє програма.

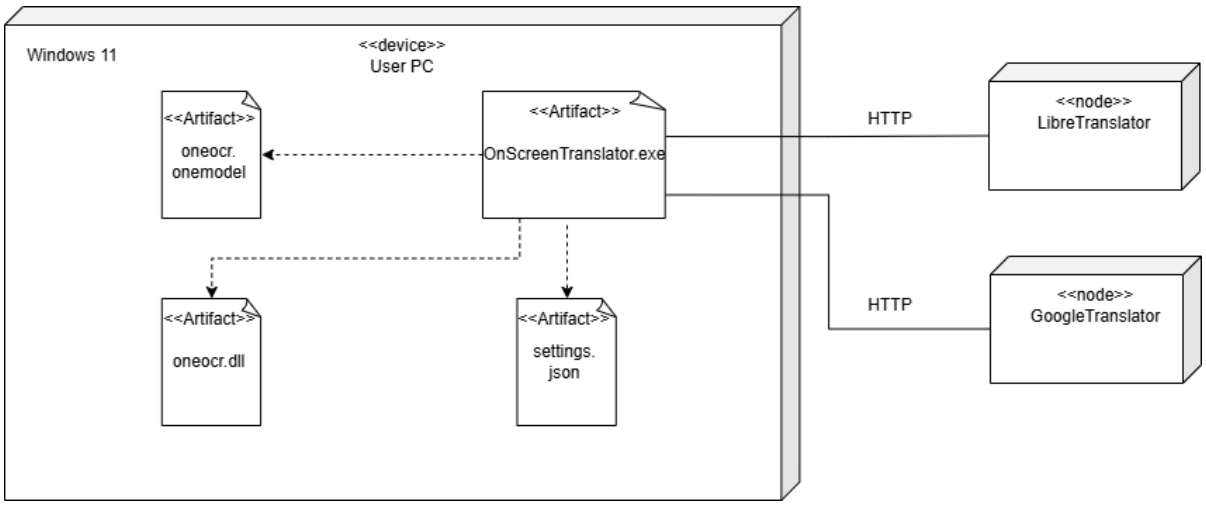

Рис. 26 Діаграма розгортання системи

Як видно з діаграми, основним середовищем виконання програми є персональний комп’ютер користувача під керуванням операційної системи Windows. Саме на цьому пристрої розміщується основний виконуваний файл програми – OnScreenTranslator.exe.

Цей файл містить основну логіку роботи системи: взаємодію з користувачем, керування процесом перекладу, захоплення зображення з екрану, запуск OCR-обробки та відображення результату в overlay-вікні.

Окрім основного виконуваного файлу, на комп’ютері користувача використовуються додаткові файли, необхідні для роботи системи. Файл oneocr.dll відповідає за роботу модуля розпізнавання тексту, а файл oneocr.onemodel містить модель, яка використовується під час OCR-обробки. Ці компоненти потрібні для того, щоб система могла отримувати текст із зображення.

Також у середовищі виконання використовується файл settings.json. Він потрібний для збереження налаштувань користувача, зокрема мов перекладу, типу перекладача, інтервалу оновлення та параметрів overlay-вікна. Під час запуску програма зчитує ці дані, а після зміни налаштувань оновлює файл. Якщо

файл відсутній, то при запуску, додаток буде використовувати стандартні налаштування і при виході з програми, або застосуванні нових параметрів, налаштування будуть записані.

Зовнішня частина системи представлена сервісами перекладу, до яких програма звертається через мережу. У межах роботи передбачена взаємодія з такими сервісами, як **LibreTranslator** та **GoogleTranslator**. Програма передає розпізнаний текст через HTTP-запит і отримує перекладений результат у відповідь.

З урахуванням такої схеми розгортання можна визначити основні вимоги до апаратного та програмного забезпечення. Для стабільної роботи додатку рекомендовано використовувати комп’ютер із процесором не нижче **Intel Core i3** або аналогічним. Для комфортної роботи, особливо при частому оновленні перекладу, бажано використовувати процесор рівня **Intel Core i5** або вище.

Мінімальний обсяг оперативної пам’яті становить **4 ГБ**. При меншому обсязі пам’яті можливі затримки під час OCR-обробки або одночасної роботи з іншими програмами. Також важливим фактором є стабільне інтернет-з’єднання, оскільки переклад виконується через зовнішні сервіси машинного перекладу.

Програмний додаток орієнтований на роботу в операційній системі **Windows 11**. Це пов’язано з використанням технології WPF, платформи .NET та можливостей Windows для роботи з вікнами, overlay-вікном і гарячими клавішами.

Для коректної роботи системи необхідні такі компоненти:

- операційна система Windows 11;
- платформа .NET;
- файли та бібліотеки, необхідні для роботи OCR;
- основний виконуваний файл OnScreenTranslator.exe.

У процесі тестування було встановлено, що на комп’ютері із середніми характеристиками система працює стабільно та не створює значного навантаження на ресурси. Найбільше навантаження виникає під час OCR-обробки та надсилання запитів до сервісів перекладу.

4.3.2 Склад інсталяційного пакету. Оскільки програмний додаток має запускатися на комп'ютері користувача, важливо також визначити склад інсталяційного пакету. Він повинен містити всі основні файли, які потрібні для запуску програми, роботи OCR-модуля та коректного відображення інтерфейсу.

Склад інсталяційного пакету пов'язаний із діаграмою розгортання, оскільки саме ці компоненти розміщуються на пристрої користувача та забезпечують роботу системи. Основним файлом пакету є OnScreenTranslator.exe, який запускає програму та керує всіма основними процесами. Для роботи OCR використовуються oneocr.dll та oneocr.onemodel. Також до складу пакету можуть входити додаткові ресурси інтерфейсу, зокрема файли локалізації. Склад інсталяційного пакету наведено в таблиці 4.1.

Таблиця 4.1

Склад інсталяційного пакету

№	Компонент	Призначення
1	OnScreenTranslator.exe	Основний виконуваний файл програмного додатку
2	oneocr.dll	Бібліотека для роботи OCR-модуля
3	oneocr.onemodel	Модель, що використовується для розпізнавання тексту
4	Файли локалізації	Ресурси для відображення інтерфейсу різними мовами
5	Додаткові бібліотеки .Net	Файли, необхідні для запуску та роботи програми

Після встановлення інсталяційного пакету користувач повинен мати можливість запустити програму без ручного налаштування структури файлів. Усі необхідні компоненти мають розміщуватися в одній директорії або в підпапках, до яких програма звертається під час запуску.

ВИСНОВКИ

У результаті виконання роботи було створено програмний додаток, який перекладає текст з екрану без ручного копіювання. Основна увага приділялась тому, щоб користувач міг швидко отримати переклад із будь-якої вибраної області екрану.

На першому етапі було розглянуто предметну область і наявні рішення. Це дозволило зрозуміти, які можливості вже реалізовані в подібних програмах і чого їм не вистачає. На основі цього було спроєктовано структуру додатку, побудовано UML-діаграми, логічну модель даних, діаграми компонентів і пакетів, а також описано середовище розгортання системи.

Програмний додаток реалізовано на мові програмування C# з використанням фреймворку WPF. У системі реалізовано вибір області екрану, захоплення зображення, розпізнавання тексту за допомогою OCR, переклад через зовнішні сервіси та відображення результату в overlay-вікні. Для збереження налаштувань використовується JSON-файл.

Проведене тестування показало, що додаток виконує основні функції та може використовуватись для перекладу тексту з екрану під час навчання, роботи з документацією, перегляду відео або використання програм без локалізації. У майбутньому додаток можна розвивати далі: додати інші OCR-модулі, підключити нові сервіси перекладу та розширити налаштування overlay-вікна.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. RegisterHotKey function – [Електронний ресурс] – режим доступу: https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-registerhotkey (дата звернення: 20.04.2026)

2. RSTGameTranslation – [Електронний ресурс] – режим доступу: https://github.com/thanhkeke97/RSTGameTranslation (дата звернення: 20.04.2026)

3. GTranslate – [Електронний ресурс] – режим доступу: https://github.com/d4n3436/GTranslate (дата звернення: 20.04.2026)

4. LibreTranslate Documentation – [Електронний ресурс] – режим доступу: https://docs.libretranslate.com/ (дата звернення: 20.04.2026)

5. Google Cloud Translation API – [Електронний ресурс] – режим доступу: https://docs.cloud.google.com/translate/docs/reference/rest (дата звернення: 22.04.2026)

6. Using the Fluent Theme in WPF with PowerShell and .Net 9 – [Електронний ресурс] – режим доступу: https://smsagent.blog/2024/11/19/using-the-fluent-theme-in-wpf-with-powershell-and-net-9/ (дата звернення: 26.04.2026)

7. Introducing Fluent 2 – [Електронний ресурс] – режим доступу: https://developer.microsoft.com/en-us/fluentui (дата звернення: 26.04.2026)

8. Make HTTP requests with the HttpClient class – [Електронний ресурс] – режим доступу: https://learn.microsoft.com/en-us/dotnet/fundamentals/networking/http/httpclient (дата звернення: 26.04.2026)

9. REST API Calls with HttpClient in C# – [Електронний ресурс] – режим доступу: https://www.tutorialspoint.com/csharp/csharp_rest_api_calls_with_httpclient.htm (дата звернення: 26.04.2026)

10. OCR sample – [Електронний ресурс] – режим доступу: https://learn.microsoft.com/en-us/samples/microsoft/windows-universal-samples/ocr/ (дата звернення: 20.04.2026)

11. 7 Proven Ways to Improve OCR Accuracy – [Електронний ресурс] – режим доступу: https://www.smartslipdate.com/blog/improve-ocr-accuracy-tips (дата звернення: 30.04.2026)

12. Dispatcher.Invoke Method – [Електронний ресурс] – режим доступу: https://learn.microsoft.com/en-us/dotnet/api/system.windows.threading.dispatcher.invoke?view=windowsdesktop-10.0 (дата звернення: 25.04.2026)

13. CancellationTokenSource.Cancel Method – [Електронний ресурс] – режим доступу: https://learn.microsoft.com/en-us/dotnet/api/system.threading.cancellationtokensource.cancel?view=net-10.0 (дата звернення: 25.04.2026)

14. WPF Threading Model – [Електронний ресурс] – режим доступу: https://learn.microsoft.com/en-us/dotnet/desktop/wpf/advanced/threading-model (дата звернення: 15.04.2026)

15. Translumo – [Електронний ресурс] – режим доступу: https://github.com/ramjke/Translumo (дата звернення: 20.04.2026)

16. Realtime Screen OCR Translator – [Електронний ресурс] – режим доступу: https://store.steampowered.com/app/3923160/Realtime_Screen_OCR_Translator/ (дата звернення: 20.04.2026)

17. JsonSerializer.Serialize Method – [Електронний ресурс] – режим доступу: https://learn.microsoft.com/en-us/dotnet/api/system.text.json.jsonserializer.serialize?view=net-10.0 (дата звернення: 16.04.2026)

18. LiveScreenTranslator – [Електронний ресурс] – режим доступу: https://github.com/ryanp3343/LiveScreenTranslator (дата звернення: 20.04.2026)

19. Windows Presentation Foundation documentation – [Електронний ресурс] – режим доступу: https://learn.microsoft.com/en-us/dotnet/desktop/wpf/ (дата звернення: 26.04.2026)

20. XAML overview (WPF) – [Електронний ресурс] – режим доступу: https://learn.microsoft.com/en-us/dotnet/desktop/wpf/xaml/ (дата звернення: 26.04.2026)

21. Graphics.CopyFromScreen Method – [Електронний ресурс] – режим доступу: https://learn.microsoft.com/en-us/dotnet/api/system.drawing.graphics.copyfromscreen?view=windowsdesktop-10.0 (дата звернення: 25.04.2026)

22. Serialize and deserialize JSON using C# – [Електронний ресурс] – режим доступу: https://learn.microsoft.com/en-us/dotnet/standard/serialization/system-text-json/overview (дата звернення: 16.04.2026)

23. Task cancellation – [Електронний ресурс] – режим доступу: https://learn.microsoft.com/en-us/dotnet/standard/parallel-programming/task-cancellation (дата звернення: 25.04.2026)

24. dotnet/wpf: Windows Presentation Foundation – [Електронний ресурс] – режим доступу: https://github.com/dotnet/wpf (дата звернення: 26.04.2026)

25. WPF Samples – [Електронний ресурс] – режим доступу: https://learn.microsoft.com/en-us/dotnet/desktop/wpf/wpf-samples (дата звернення: 26.04.2026)

ДОДАТКИ

ДОДАТОК А

Код алгоритму процесу перекладу

```
private void StartTranslationLoop()
{
    // check if translation is running
    if (_isTranslationRunning)
        return;

    // if translation is not running set lock
    _isTranslationRunning = true;

    // cancellation token to cancel translation using button
    _translationCts = new CancellationTokenSource();
    var token = _translationCts.Token;

    SettingsManager settings = SettingsManager.GetInstance();

    _translationService = settings.GetTranslationService(out string apikey);

    // languages that will be used in translator
    string source = ComBoxSourceLang.SelectedValue.ToString();
    string target = ComBoxTargetLang.SelectedValue.ToString();

    int translationInterval = settings.GetTranslationInterval() * 1000;

    if (_overlayWindow != null)
    {
        _overlayWindow.Dispatcher.Invoke(() =>
            _overlayWindow?.TxtTranslationStatus.Text =
Strings.TranslationStatusOn
        );
    }

    bool isBlockMode = settings.GetTranslationOcrMode().Equals("block");
    bool isOcrMode = settings.GetTranslationWorkMode().Equals("ocr");
    bool isOneTime = settings.GetTranslationUpdateMode().Equals("onetime");
    bool isOneTimeWithWaiting =
settings.GetTranslationUpdateMode().Equals("onetimewithwaiting");

    if (isOneTimeWithWaiting) translationInterval = int.MaxValue;

    Task.Run(async () =>
    {
        try
        {
            while (!token.IsCancellationRequested)
            {
                // if overlay is hidden do not spend resources to translate
anything
```

```
                if (_overlayWindow == null)
                {
                    await Task.Delay(400);
                    continue;
                }

                Bitmap image;

                // check if overlay lays over text area we want to translate
                bool? intersects = _overlayWindow?.Dispatcher.Invoke(() =>
_overlayWindow?.IntersectsScreenArea(_selectedScreenArea.Value)
                );

                if (intersects == true)
                {
                    // hide overlay to not capture it
                    _overlayWindow?.Dispatcher.Invoke(() =>
_overlayWindow?.Hide());
                    image =
ScreenCaptureService.GetImage(_selectedScreenArea.Value);
                    _overlayWindow?.Dispatcher.Invoke(() =>
_overlayWindow?.Show());
                }
                else
                {
                    image =
ScreenCaptureService.GetImage(_selectedScreenArea.Value);
                }

                // getting text from image
                string text;
                if (isBlockMode)
                    text = _ocrService.GetTextFromImage(image).Replace("\r\n",
" ");
                else
                    text = _ocrService.GetTextFromImage(image);
                image.Dispose();

                // check if text has to be translated
                if (!string.IsNullOrEmpty(text) && (!_previousText.Equals(text)
||
                    !_previousSourceLang.Equals(source) ||
!_previousTargetLang.Equals(target)))
                {
                    // update previous data
                    _previousText = text;
                    _previousSourceLang = source;
                    if (isOcrMode)
                        _previousTargetLang = target = source;
                    else
                        _previousTargetLang = target;

                    // translate text if source and target languages are
different
```

```
                    string translated = source == target ? text : await
_translationService.TranslateAsync(
                        text, source, target, apikey
                    );
                    _previousTranslatedText = translated;

                    // update text in overlay
                    Dispatcher.Invoke(() =>
                    {
                        _overlayWindow?.TxtOverlay.Visibility =
Visibility.Visible;
                        _overlayWindow?.TxtOverlay.Text = translated;
                    });

                    if (isOneTime)
                        return;
                }

                await Task.Delay(translationInterval, token);
            }
        }
        catch (OperationCanceledException)
        {
            // exit translation loop if stop button was pressed
            return;
        }
        catch (Exception ex)
        {
            Dispatcher.Invoke(() =>
            {
                MessageBox.Show(ex.Message, "Translation error",
MessageBoxButton.OK, MessageBoxImage.Error);
            });
            return;
        }
        finally
        {
            // release lock
            _isTranslationRunning = false;
            Dispatcher.Invoke(() => TlgBtnStartStopTranslation.IsChecked =
false);
            _overlayWindow?.Dispatcher.Invoke(() =>
                _overlayWindow?.TxtTranslationStatus.Text =
Strings.TranslationStatusOff
            );
        }
    }, token);
     }
```

ДОДАТОК Б

Діаграма активності процесу перекладу

ДОДАТОК В

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

Декларація про академічну доброчесність та використання ШІ

Я, Закомірний Максим Сергійович, здобувач(ка) НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

1. Моя бакалаврська кваліфікаційна робота на тему «Програмний додаток перекладу тексту з екрану в реальному часі» є результатом моєї особистої праці.
2. Усі запозичення з текстів інших авторів мають відповідні посилання згідно з чинними стандартами.
3. Щодо використання інструментів ШІ зазначаю, що (обрати необхідне):
    - [ ] інструменти генеративного ШІ не використовувалися.
    - [X] інструменти ШІ (ChatGPT, Gemini) були використані для:
        - [ ] перекладу іншомовних джерел;
        - [X] стилістичного редагування та перевірки граматики;
        - [X] допомоги у написанні/відлагодженні програмного коду;
        - [ ] інше: _____.

Я розумію, що надання недостовірної інформації може призвести до скасування результатів захисту та відрахування з числа здобувачів НУБіП України.

«___»_____2026 р. _____ **Максим ЗАКОМІРНИЙ**

(підпис)