**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**
**ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

**МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ**
**Кафедра програмної інженерії**

# КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА СЕРВІСУ АВТОМАТИЧНОГО ПЕРЕКЛАДУ ТЕКСТУ З ВИКОРИСТАННЯМ JAVASCRIPT ТА AMAZON TRANSLATE»**

Виконав: студент 4 курсу, групи 6.1210-1пі
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
Р.Ю. Санталов
(ініціали та прізвище)
Керівник доцент кафедри програмної інженерії, PhD, Столярова А.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент доцент кафедри фундаментальної та прикладної математики, доцент, к.ф.-м.н. Панасенко Є.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2024

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

**ЗАТВЕРДЖУЮ**
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2023 р.

# З А В Д А Н Н Я

## НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Санталову Руслану Юрійовичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка сервісу автоматичного перекладу тексту з використанням JavaScript та Amazon Translate

керівник роботи Столярова Анастасія Валеріївна, PhD
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 21 » грудня 2023 року № 2180-с

2. Строк подання студентом роботи 03.06.2024 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Проєктування та тестування сервісу автоматичного перекладу тексту.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | Завдання видав | Завдання прийняв |
| | | | |
| | | | |
| | | | |

7. Дата видачі завдання 25.12.2023 р.

## К А Л Е Н Д А Р Н И Й  П Л А Н

| № | Назва етапів кваліфікаційної роботи | Строк виконання етапів роботи | Примітка |
|---|---|---|---|
| 1. | Розробка плану роботи. | 10.01.2024 | |
| | | | |
| 2. | Збір вихідних даних. | 24.01.2024 | |
| | | | |
| 3. | Обробка методичних та теоретичних | 26.02.2024 | |
| | джерел. | | |
| | | | |
| 4. | Розробка першого та другого розділу. | 18.04.2024 | |
| | | | |
| 5. | Розробка третього розділу. | 20.05.2024 | |
| | | | |
| 6. | Оформлення та нормоконтроль | 27.05.2024 | |
| | кваліфікаційної роботи бакалавра. | | |
| | | | |
| 7. | Захист кваліфікаційної роботи. | 18.06.2024 | |

Студент _____ Р.Ю. Санталов
(підпис) (ініціали та прізвище)

Керівник роботи _____ А.В. Столярова
(підпис) (ініціали та прізвище)

## Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

# РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка сервісу автоматичного перекладу тексту з використанням JavaScript та Amazon Translate»: 48 с., 19 рис., 1 табл., 13 джерел, 5 додатків.

АВТОМАТИЧНИЙ ПЕРЕКЛАД, ЗАСТОСУНОК, СЕРВІС, AMAZON TRANSALTE, JAVASCRIPT.

Об’єкт дослідження – процес розробки сервісу автоматичного перекладу тексту.

Предмет дослідження – дослідження JavaScript та Amazon translate.

Мета роботи: розробити сервіс для автоматизованого перекладу тексту.

Метод дослідження – метод збору та аналізу вимог до програмного забезпечення, методи проєктування, конструювання та тестування програмного забезпечення.

При розробці сервісу був проведений аналіз предметної області, обраний та спроєктований застосунок за допомогою мови програмування JavaScript та сервісу машинного перекладу Amazon Translate. Інтегрованим середовищем розробки було обрано WebStorm, а візуалізація програми була розроблена за допомогою фреймворку Electron. Сфера застосування застосунку – області, де потрібна робота з текстом на іноземній мові.

В результаті роботи отримано програмне забезпечення для автоматичного перекладу тексту.

# SUMMARY

Bachelor’s qualifying paper «Development the Service of Automatic Translation Text Using JavaScript and Amazon Translate»: 48 pages, 19 figures, 1 table, 13 references, 5 supplements.

AMAZON TRANSLATE, APPLICATION, AUTOMATIC TRANSLATION, JAVASCRIPT, SERVICE.

The object of the study is the process of developing an automatic text translation service.

The subject of the research is the study of JavaScript and Amazon Translate.

The aim of the study is development of a service for automatic text translation using JavaScript and Amazon Translate.

The methods of research are method of collecting and analyzing software requirements, software design methods, construction, and testing of software.

During the development of the service, an analysis of the subject area was conducted, and an application was selected and designed using the JavaScript programming language and the Amazon Translate machine translation service. WebStorm was chosen as the integrated development environment, and the application's visualization was developed using the Electron framework. The application is intended for use in areas where working with text in a foreign language is required.

As a result of the work, software for automatic text translation was obtained.

**ЗМІСТ**

# ВСТУП

Через стрімкий розвиток глобалізації, технологій та міжнародних комунікацій активно повстає питання необхідності в освоєнні нових мов, особливо якщо це стосується перекладу ділової документації або наукових статей з престижних журналів. Саме тому машинний переклад тексту представляє особливе значення.

Основними застосунками для перекладу тексту на персональних комп'ютерах є Tooltip Translator, Client for Google Translate, QtTranslate, Dictionarist, але в цих застосунках наявні обмеження стосовно способу вводу та отримання інформації, а також вибір іншого сервісу для перекладу інформації. Саме через це вбачається доцільним розробка застосунку для автоматичного перекладу тексту з використанням Amazon Translate, як альтернативного сервісу для перекладу.

Автоматичний переклад тексту застосовують у різних сферах:

- міжнародний бізнес: переклад угод та ділових паперів на інші мови дозволяє пришвидшити процес перемовин та підписання документів;
- туризм: переклад інформації про туристичні пам'ятки, меню в ресторанах, інформації в готелі дозволяє туристам, що не знають офіційної мови країни в якій вони перебувають, насолоджуватися відпочинком та дізнаватися нову інформацію про цю країну;
- освіта: переклад наукових статей, дипломів на інші мови студентами для публікації або захисту не лише в університеті в якому вони навчаються, але і публікуватися в міжнародних наукових журналах;
- інтернет-контент: переклад іноземних новинних сайтів, можливість перекладу онлайн чатів, соціальних мереж дозволяє позбутися мовного бар'єру при спілкуванні.

Предметом дослідження є застосування JavaScript та Amazon Translate.

Об'єктом дослідження є процес розробки сервісу автоматичного

перекладу тексту.

Мета роботи: розробити сервіс для автоматизованого перекладу тексту. Для досягнення поставленої мети необхідно вирішити наступні завдання:

– проаналізувати сучасний стан проблематики у сфері автоматичного перекладу тексту;
– визначити інструменти для програмної реалізації автоматичного перекладу тексту;
– реалізувати та протестувати застосунок для автоматичного перекладу тексту.

Кваліфікаційна робота складається зі вступу, трьох розділів, висновків, переліку посилань та п’яти додатків. Загальний обсяг роботи становить 48 сторінок. Додатки викладено на 15 сторінках. У переліку посилань 13 використаних джерел.

# 1 ОПИС ПРАКТИЧНОГО ЗАВДАННЯ

Машинний переклад – автоматичний переклад з вихідної мови на цільову за допомогою комп'ютерних програм або двигунів машинного перекладу без участі людини [1]. Основною перевагою машинного перекладу є швидкий і простий процес перекладу тексту.

В межах даного розділу, буде наведено найпоширеніші інструменти для автоматичного перекладу тексту, а саме мови програмування (Python, C++, JavaScript), сервіси машинного перекладу (Google Cloud Translation AI, Amazon Translate, Crowdin) та фреймворки (Electron, React Native, Flutter).

## 1.1 Вимоги до програмної реалізації

Застосунок для автоматичного перекладу тексту повинен мати в собі наступний функціонал:

- вибір як мови перекладу, так і його автоматичне визначення застосунком;
- автоматичний переклад тексту при його введенні в визначене поле застосунку;
- розпізнавання тексту з зображення та його автоматичний переклад;
- запис, відтворення у вигляді тексту голосу людини та його автоматичний переклад на іншу мову;

## 1.2 Вибір мови програмування

Серед сучасних мов програмування розглянемо найпопулярніші.

***Python*** – мова програмування, яка широко використовується при

розробці застосунків та навчанні штучного інтелекту [2].

Перевагами Python є:

– велика стандартна бібліотека, що надає можливість використовувати розробникам використовувати вже наявний код без необхідності писати його з нуля;
– інтеграція з іншими мовами програмування, такими як Java, C, C++;
– можна використовувати на різних операційних системах;
– легкість у читанні, завдяки простому синтаксису.

***C++*** – мова програмування загального призначення, що підтримує такі парадигми, як об'єктно-орієнтованість, узагальненість та процедурність програмування.

Серед переваг використання C++ можна виділити наступні [3]:

– управління пам'яттю;
– нульова абстракція;
– детерміноване руйнування;
– можливість використання функції друга.

***JavaScript*** – високорівнева скриптова мова програмування, що широко використовується для створення інтерактивних застосунків та сайтів [4]. JavaScript надає такі можливості, як:

– додавання анімації і спливаючих повідомлень;
– валідація форм;
– оновлення контенту та моментальне реагування на дії розробника.

Перевагами застосування Javascript є:

– інтерпретованість – виконання коду в реальному часі без необхідності щоразу компілювати код ;
– динамічна типізація – тип змінних присвоюється автоматично під час виконання коду;
– велика кількість бібліотек та фреймворків;
– кросплатформеність – виконання коду на різних платформах та пристроях;

- автоматичне збирання сміття – оптимізація пам'яті, шляхом видалення об'єктів що не використовуються;
- імперативність, функціональність та основи об’єктно – орієнтованого програмування – підтримка основної парадигми програмування дозволяє створювати складні структури, описувати кроки до виконання та застосовувати функціональні концепції;
- асинхронність – використання механізмів, що дозволяють опрацьовувати операції, що не потребують негайного завершення без блокування інших операцій;
- відкритий код – доступність для використання без необхідності купувати ліцензію.

Зважаючи на переваги кожної мови програмування, було вирішено застосувати JavaScript, оскільки ця мова програмування дозволяє використовувати різні фреймворки та бібліотеки при створенні застосунків, а також наявну можливість виконувати код в реальному часі динамічну типізацію.

### 1.3 Вибір сервісу машинного перекладу

Зважаючи на стрімкий розвиток технологій, з’явилася велика кількість сервісів для машинного перекладу. Розглянемо деякі з них.

***Google Cloud Translation AI*** – служба перекладу на основі штучного інтелекту. Ця служба є більш гнучкою та контрольованою аніж Google Translate в процесах налаштування перекладів відповідно до потреб користувача. Сама служба дозволяє переводити вебсайти, застосунки, аудіо- та відеофайли.

Translation API та Translation Hub є службами перекладу, тоді як AutoML Translation інтегрується з обома для підвищення точності за допомогою спеціальних моделей перекладу.

API перекладу допомагає програмно перекладати в реальному часі. Також він працює з іншими API для складніших випадків використання. Translation Hub пропонує вебінтерфейс без коду для організації складних робочих процесів перекладу з контролем і безпекою корпоративного рівня [5].

***Crowdin*** – програмне забезпечення на основі штучного інтелекту для локалізації застосунків.

Перевагами використання Crowdin є:

– наявність пам’яті перекладу;

– підтримка великої кількості форматів файлів;

– інтеграції API за допомогою CMS;

– наявність можливості отримувати аналітику та звіти.

Схема застосування локалізації на платформі Crowdin наведено на рисунку 1.1 [6].


Рисунок 1.1 – Процес локалізації за допомогою платформи Crowdin

***Amazon Translate*** – сервіс нейронного машинного перекладу, що забезпечує швидкий, високоякісний, доступний за вартістю переклад з користувацькими налаштуваннями з однієї мови на іншу.

Працює це наступним чином: користувач завантажує текст або текстовий файл в сервіс Amazon Translate. Amazon Translate своєю чергою використовуючи технології машинного навчання та більш ніж 5500 комбінацій перекладу перекладає текст або текстовий файл на потрібну користувачу мову (рис. 1.2) [7]. Сервіс при цьому використовує такі функції, як:

– переклад документів;
– маскування ненормативної лексики;
– додержання формальності;
– автоматичне джерело знаходження мови.

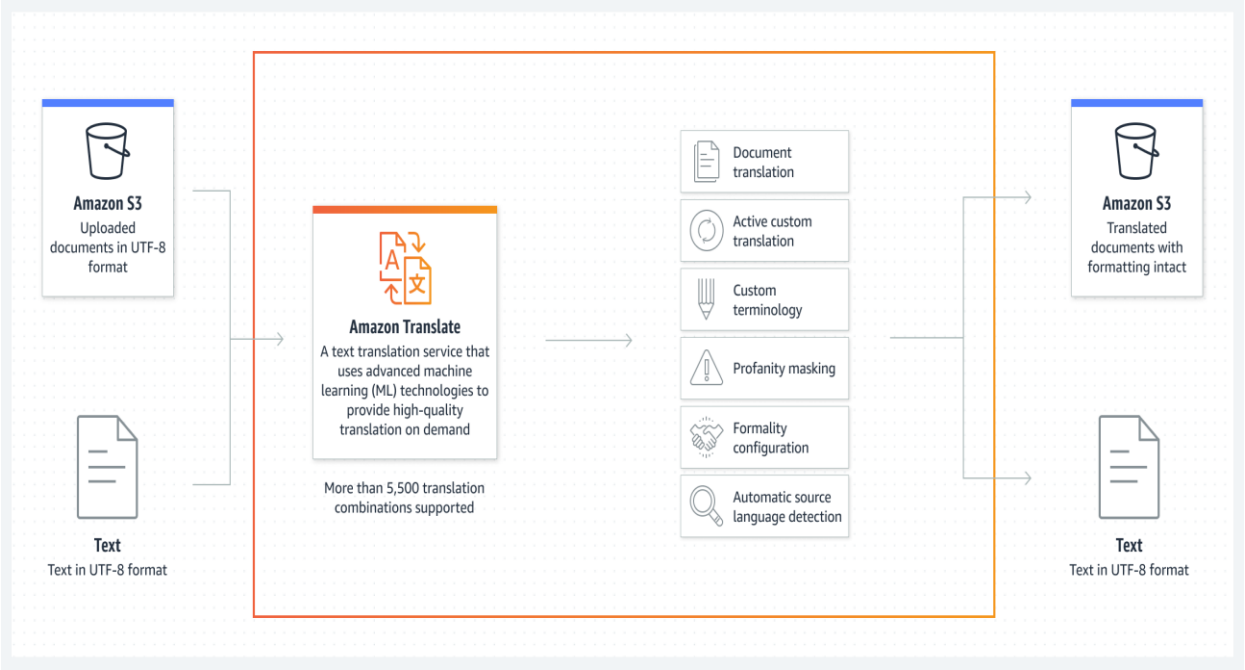


Рисунок 1.2 – Схема роботи Amazon Translate

Одним з важливих факторів, що відіграють при виборі сервісів машинного перекладу, є вартість використання. Amazon Translate пропонує як безкоштовне використання, але з обмеженням 2 мільйони переведених символів на місяць, так і платне в діапазоні від 15 до 60 доларів в залежності від типу перекладу. Google AI також має в своєму функціоналі безкоштовний тип перекладу з обмеженням в 500 тисяч переведених символів, так і платне в

діапазоні від 20 до 80 доларів в залежності від потреб користувача. Crowdin, на відміну від попередніх сервісів, розповсюджується за місячною підпискою в діапазоні від 50 до 450 доларів. Тому, зважаючи на переваги використання Amazon Translate, зокрема в наданій кількості безкоштовних символів для перекладу, було вирішено використовувати саме його.

## 1.4 Вибір фреймворку

Для реалізації нашого застосунку розглянемо найпопулярніші сучасні фреймворки.

***React Native*** – фреймворк інтерфейсу, що використовується для розробки програм на різних платформах. Серед переваг можна виділити [8]:

- прискорена розробка застосунків завдяки готовим компонентам та великій екосистемі пакетів JavaScript;
- легкість в опанування фреймворку;
- можливість використання коду на різних платформах.

***Flutter*** – платформа з відкритим кодом, призначена для створення користувацького інтерфейсу застосунків для різних платформ з використанням однієї бази даних коду.

Перевагами Flutter є:

- застосування вбудованої мови програмування Dart, що забезпечує швидку роботу та високу продуктивність;
- використання графічної бібліотеки Google Skia для відображення інтерфейсу, що забезпечує однаковий вигляд застосунків на різних платформах;
- можливість бачити зміни в реальному часі в роботі застосунку при написанні написання коду.

***Electron*** – фреймворк від GitHub, що дозволяє розробляти різні графічні застосунки для операційних систем за допомогою вебтехнологій. Перевагами

його використання є [9]:

– створення застосунків на різних операційних системах з використанням єдиної кодової бази;
– використання HTML, CSS, JavaScript, що полегшує розробку застосунків;
– використання Chronium для рендеру користувацького інтерфейсу.

Зважаючи на вибір мови програмування та сервісу перекладу, найкращим варіантом буде використання саме фреймворку Electron.

## 1.5 Висновки до розділу 1

Для реалізації визначеного функціоналу застосунку найефективнішим з усіх наведених варіантів буде використання JavaScript у якості мови програмування, Amazon Translate, як сервісу машинного перекладу та фреймворку Electron для відображення графічного застосунку. На це є декілька причин:

– використання JavaScript у поєднанні з Electron дозволяє використовувати знайомі технології для створення десктопних застосунків (це значно спрощує розробку та підтримку, оскільки розробники можуть використовувати один мовний стек для всіх платформ);
– інтеграція Amazon Translate з Electron є простою завдяки документації та доступним SDK (це дозволяє швидко впроваджувати функціональність машинного перекладу без великих витрат часу та зусиль);
– застосування саме JavaScript, що дозволяє виконувати код в режимі реального часу без необхідності щоразу компілювати код і наявність динамічної типізації та великої кількісті бібліотек, значно зменшує витрати часу на розробку застосунку.

# 2 ПРОГРАМНА РЕАЛІЗАЦІЯ ПРАКТИЧНОГО ЗАВДАННЯ

В межах даного розділу буде наведено програмну реалізацію фронтенду та бекенду з описом їх функцій, зображення застосунку та створення його виконуваного файлу.

## 2.1 Програмна реалізація фронтенду

Для програмної реалізації фронтенду було створено такі файли як index.html та styles.css [10].

Index.html створює інтерфейс для роботи застосунку, зокрема:

– doctype, html, head, body – структурні елементи цього документу, що містять в собі метаінформацію та основний вміст сторінки;
– link rel="stylesheet" href="styles.css" – під'єднує файл css для стилізації сторінки;
– source Language, Targeted Language – відображує випадаючий список для вибору вихідної мови, встановлює мітку для нього та створює список, що буде динамічно заповнюватись за допомогою JavaScript (див. рис. 2.1);
– text Input – відображує поле введення тексту та за допомогою кнопки перекласти текст;
– image Input – при натисканні кнопки, відображає поле для завантаження зображення та його подальший переклад;
– voice input – за допомогою кнопок здійснюється початок та зупинка голосового розпізнавання тексту з його подальшим виведенням на екран та перекладом на необхідну мову.

Styles.css є невід’ємною частиною розробки фронтенду, оскільки це є набором характеристик форматування, що визначаються в каскадній таблиці

стилів і визначають спосіб відображення самих елементів html [11]. Основними елементами цього файлу є:

– body – тіло документа, де розташовуються інші елементи;
– textarea – виділене місце для друку слів з їх подальшим перекладом на іншу мову;
– input – завантаження файлу для його обробки та переведення на іншу мову.

```
<div>
  <label for="sourceLanguage">Source Language:</label>
  <select id="sourceLanguage">
    <!-- Options will be dynamically populated by JavaScript -->
  </select>
</div>
<div>
  <label for="targetLanguage">Target Language:</label>
  <select id="targetLanguage">
    <!-- Options will be dynamically populated by JavaScript -->
  </select>
</div>
```

Рисунок 2.1 – Створення випадаючого списку доступних мов для перекладу

За результатами створення таких файлів та наповнення їх програмним кодом, було реалізовано фронтенд застосунку (див. рис. 2.2). На ньому можна побачити загальний функціонал, а саме:

– Source Language – вхідна мова;
– Target Language – бажана мова перекладу;
– Text Input – ввід тексту для перекладу
– Image Input – завантаження зображення для перекладу;
– Voice Input – переклад голосового тексту;

– Translated Text – відображення тексту, що був перекладений на бажану мову.

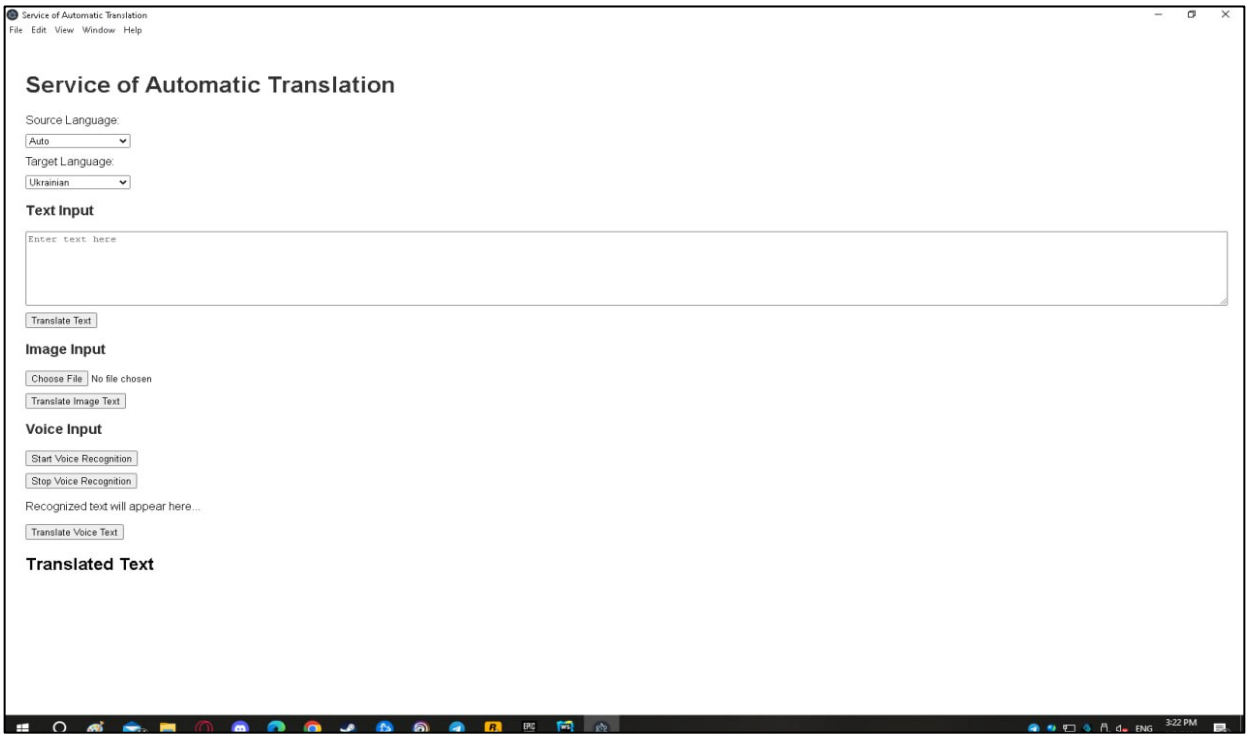


Рисунок 2.2 – Фронтенд застосунку

## 2.2 Програмна реалізація бекенду

Для реалізації бекенду застосунку було створено такі файли як main.js, preload.js, renderer.js.

Preload.js використовує фреймворк Electron для створення мосту між основним процесом та процесом відображення вебсторінки в питаннях роботи із зображеннями, текстом, а також голосовим розпізнаванням та обробляти різні події:

– processImage – використовується для запуску обробки зображення шляхом відправлення основному процесу відповідного повідомлення;
– onImageText та onImageTextError – викликається, коли користувач завантажує файл та реєструє зворотній виклик для отримання тексту;

при невдалій спробі відображується повідомлення про помилку (рис. 2.3);

– translateText, onTranslateText – використовується для перекладу тексту, шляхом відправки та отримання процесу повідомлення тексту між вихідною та бажаними мовами (рис. 2.4);
– startVoiceRecognition, onVoiceRecognition – використовується для розпізнавання тексту шляхом відправки процесу повідомлення з ключем підписки, регіону сервісу та вихідною мовою;
– saveImage, onImageSaved, onImageError – використовується для збереження зображення та його подальшого опрацювання.

```
onImageText: (callback) => ipcRenderer.on('image-text', (event, text) =>
callback(text)),
  onImageTextError: (callback) => ipcRenderer.on('image-text-error', (event,
error) => callback(error)),
```

Рисунок 2.3 – Реєстрація та подальше опрацюваннятексту зображення

```
translateText: (text, sourceLanguage, targetLanguage) =>
ipcRenderer.send('translate-text', { text, sourceLanguage, targetLanguage }),
  onTranslatedText: (callback) => ipcRenderer.on('translated-text', (event,
translatedText) => callback(translatedText)),
  onTranslateError: (callback) => ipcRenderer.on('translate-error', (event, error)
=> callback(error)),
```

Рисунок 2.4 – Переклад тексту, шляхом відправки та отримання повідомлення

Файл renderer.js взаємодіє з іншими файлами та надає функціональність до інтерфейсу застосунку. Сам включає в себе заповнення селекторів мов, обробку текстового, голосового введення інформації, а також можливість переводити текст з зображення на іншу мову, зокрема:

- селектор мов заповнює вихідну та бажану мови за допомогою об’єкта language: за замовченням там присутнє автоматичне розпізнавання тексту з перекладом його на англійську (див. рис. 2.5);
- переклад тексту здійснюється за допомогою надсилання інформації до Amazon та отримання зворотного виклику;
- переклад тексту з зображення відбувається шляхом обробки відповідних подій, шляхом надсилання запитів через фреймворк та отримання відповідей з подальшим перекладом (див. рис. 2.6);
- розпізнавання голосу з подальшим текстовою транскрипцією та можливістю перекладу на іншу мову здійснюється за допомогою функції startVoiceRecognition(), де обробник інформації отримує текст з подальшим відправленням для перекладу (див. рис. 2.7).

```
document.addEventListener('DOMContentLoaded', () => {
  const sourceLanguageSelect = document.getElementById('sourceLanguage');
  const targetLanguageSelect = document.getElementById('targetLanguage');

  for (const [code, name] of Object.entries(languages)) {
    const sourceOption = document.createElement('option');
    sourceOption.value = code;
    sourceOption.textContent = name;
    sourceLanguageSelect.appendChild(sourceOption);

    if (code !== 'auto') {
      const targetOption = document.createElement('option');
      targetOption.value = code;
      targetOption.textContent = name;
      targetLanguageSelect.appendChild(targetOption);
    }
  }

  sourceLanguageSelect.value = 'auto';
  targetLanguageSelect.value = 'uk';

  console.log('Source Language Options:', sourceLanguageSelect.innerHTML);
  console.log('Target Language Options:', targetLanguageSelect.innerHTML);

});
```

Рисунок 2.5 – Програмна реалізація функції вибору мови

```
function translateImage() {
  const imageInput = document.getElementById('imageInput');
  if (imageInput.files.length === 0) {
    alert('Please select an image file.');
    return;
  }
  const file = imageInput.files[0];
  const fileReader = new FileReader();
  fileReader.onload = function(event) {
    const arrayBuffer = event.target.result;
    window.electron.saveImage(arrayBuffer, file.name);
  };
  fileReader.readAsArrayBuffer(file);
}
window.electron.onImageSaved((filePath) => {
  window.electron.processImage(filePath);
});
window.electron.onImageText((text) => {
  document.getElementById('textInput').value = text;
  translateText(); // Переклад отриманого тексту
});
window.electron.onImageTextError((error) => {
  console.error('Image Text Extraction Error:', error);
  alert('Failed to extract text from the image.');
});
```

Рисунок 2.6 – Програмна реалізація функції перекладу тексту із зображення

```
function startVoiceRecognition() {
  const subscriptionKey = 'Enter your subscription Key';
  const serviceRegion = 'enter your region';
  const sourceLanguage = document.getElementById('sourceLanguage').value;
  window.electron.startVoiceRecognition(subscriptionKey, serviceRegion, sourceLanguage);
}
window.electron.onVoiceRecognizedText((voiceText) => {
  document.getElementById('voiceText').innerText = voiceText;
  document.getElementById('textInput').value = voiceText;
  translateText(); // Переклад розпізнаного тексту
});
function stopVoiceRecognition() {
  console.log(' stopVoiceRecognition is successful..');
}
function translateVoiceText() {
  const sourceLanguage = document.getElementById('sourceLanguage').value;
  const targetLanguage = document.getElementById('targetLanguage').value;
  const voiceText = document.getElementById('voiceText').innerText;
  window.electron.translateText(voiceText, sourceLanguage, targetLanguage);
}
```

Рисунок 2.7 – Програмна реалізація функції розпізнавання голосу

У самому застосунку це буде виглядати наступним чином – користувач, за бажанням, обирає вхідну мову (рис. 2.8), бажану мову перекладу (рис. 2.9) та вводить текст у відповідне поле. Після натискання на кнопку «Translate Text» застосунок переводить текст та виводить результат (див. рис. 2.10).

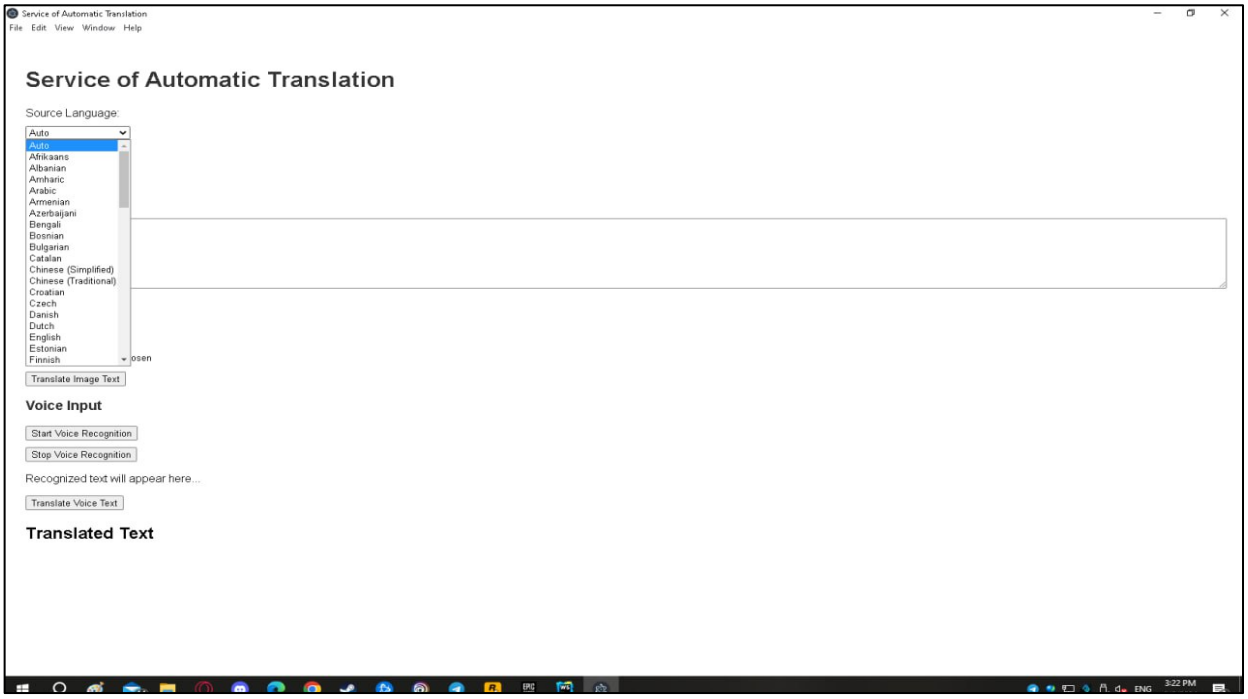


Рисунок 2.8 – Обрання вхідної мови

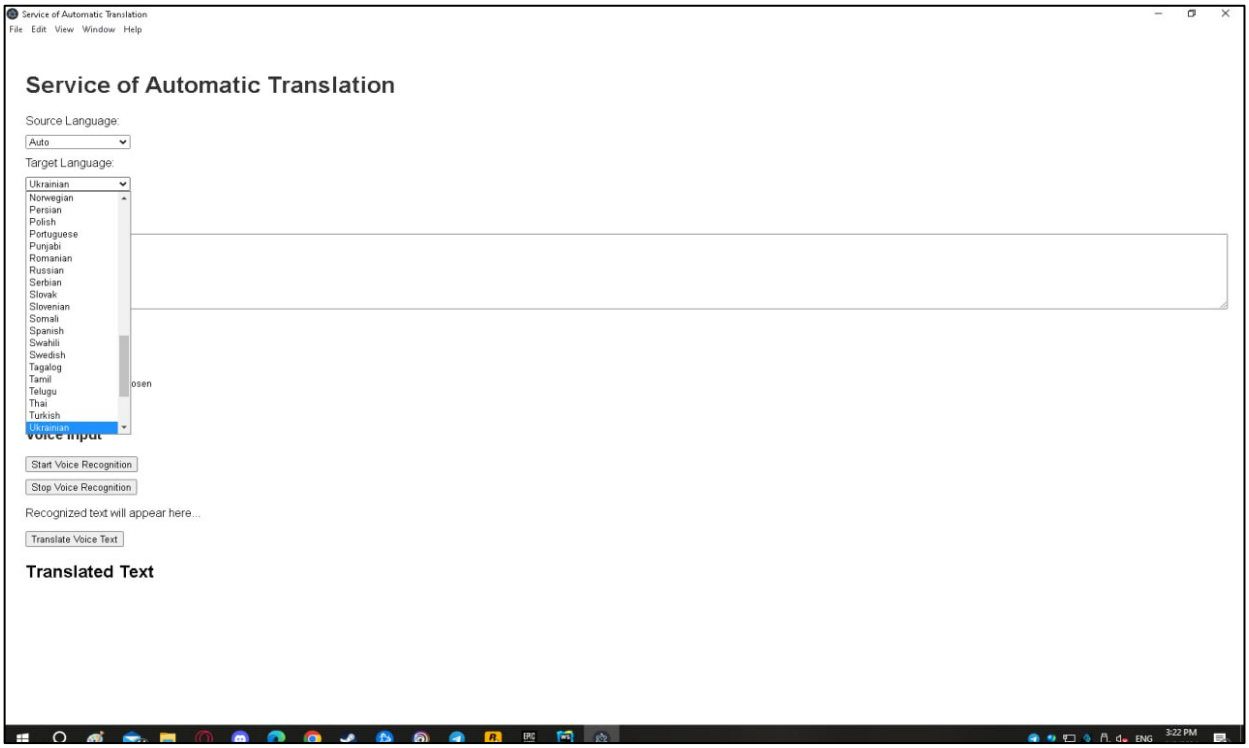


Рисунок 2.9 – Обрання бажаної мови перекладу

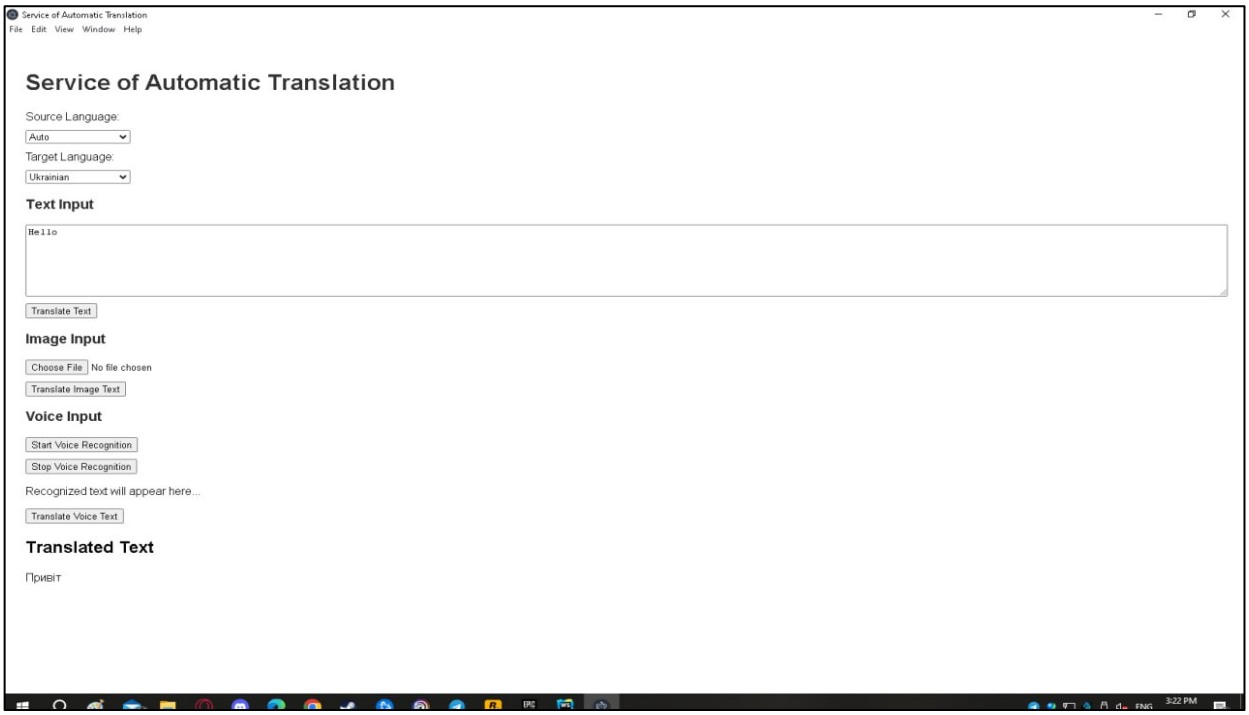


Рисунок 2.10 – Переклад тексту

Для перекладу тексту з зображення, користувачу потрібно завантажити файл в застосунок та натиснути на кнопку «Translate Image Text». За результатами роботи, застосунок розпізнає текст та зробить автоматичний переклад на бажану для користувача мову (див. рис. 2.11).

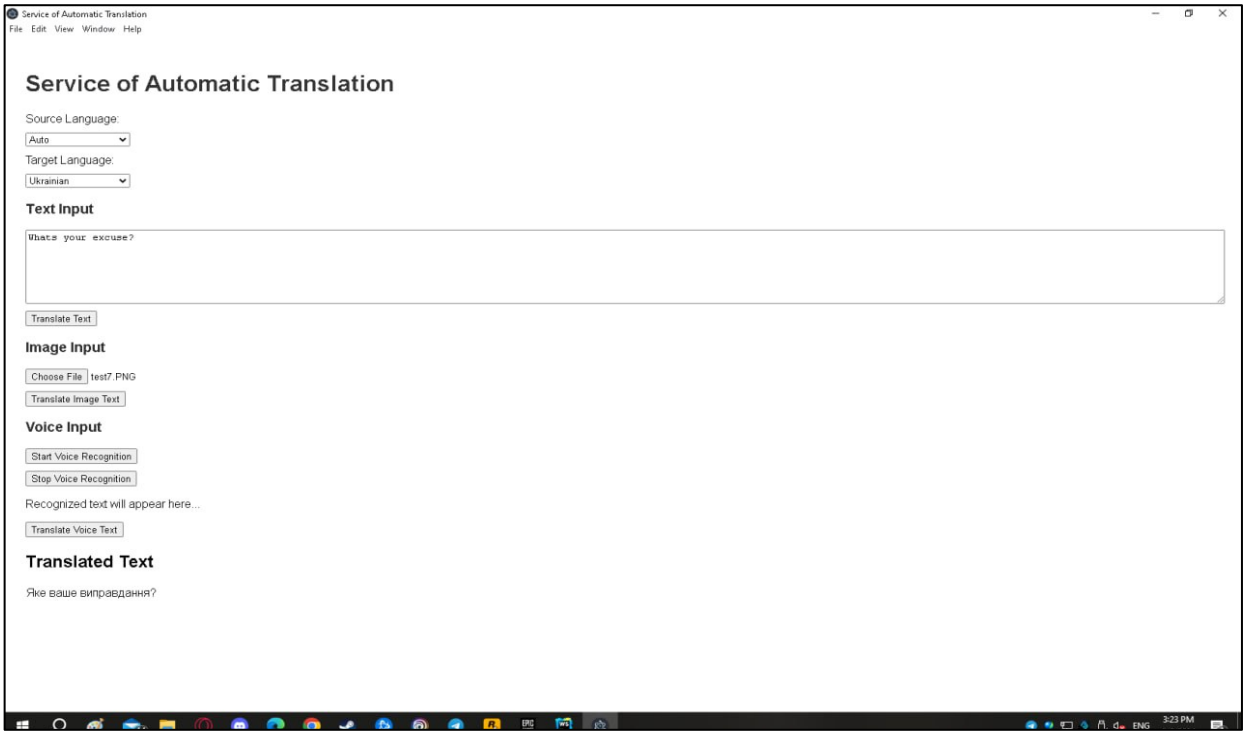


Рисунок 2.11 – Автоматичний переклад тексту з зображення

Файл main.js призначений для координації інших файлів, запуску застосунку за допомогою фреймворку Electron та інтегрує можливості оптичного розпізнавання символів (OCR) для перекладу тексту. В самому файлі присутні наступні можливості:

– інтеграція з Amazon для подальшої роботи з текстом, зображеннями та голосом (рис. 2.12);
– створення вікна із застосунком;
– обробка зображень (рис. 2.13);
– переклад тексту та його розпізнавання (рис. 2.14);
– розпізнавання голосу (рис. 2.15);
– збереження зображень (рис. 2.16).

```
AWS.config.update({
   accessKeyId: 'Enter your KeyID',
   secretAccessKey: 'Enter AccessKey',
   region: 'Enter region'
});
const translate = new AWS.Translate({ region: 'eu-central-1' });
```

Рисунок 2.12 – Програмна реалізація інтеграції з Amazon

```
ipcMain.on('process-image', (event, filePath) => {
   Tesseract.recognize(filePath, 'eng', {
      logger: m => console.log(m)
   }).then(({ data: { text } }) => {
      event.reply('image-text', text);
   }).catch(err => {

      console.error('Error:', err);

      event.reply('image-text-error', err.message);
   });
});
```

Рисунок 2.13 – Програмна реалізація обробки зображень

```
ipcMain.on('translate-text', (event, { text, sourceLanguage, targetLanguage }) => {
   const params = {
      Text: text,
      SourceLanguageCode: sourceLanguage,
      TargetLanguageCode: targetLanguage
   };

   translate.translateText(params, (err, data) => {
      if (err) {
         console.log(err, err.stack);
         event.reply('translate-error', err.message);
      } else {
         event.reply('translated-text', data.TranslatedText);
      }
   });
});
```

Рисунок 2.14 – Програмна реалізація функції перекладу та розпізнавання тексту

```
ipcMain.on('start-voice-recognition', async (event, { subscriptionKey,
serviceRegion, sourceLanguage }) => {
   const speechConfig =
speechsdk.SpeechConfig.fromSubscription(subscriptionKey, serviceRegion);
   speechConfig.speechRecognitionLanguage = sourceLanguage !== 'auto' ?
sourceLanguage : 'en-US';

   const audioConfig = speechsdk.AudioConfig.fromDefaultMicrophoneInput();
   const recognizer = new speechsdk.SpeechRecognizer(speechConfig,
audioConfig);

   recognizer.recognizeOnceAsync(result => {
      const voiceText = result.text;
      event.reply('voice-recognized-text', voiceText);
   });
});
```

Рисунок 2.15 – Програмна реалізація функції розпізнавання тексту

```
ipcMain.on('save-image', (event, fileData, fileName) => {
    const tempPath = path.join(app.getPath('temp'), fileName);
    fs.writeFile(tempPath, Buffer.from(fileData), (err) => {
        if (err) {
            console.error('Error saving file:', err);
            event.reply('image-save-error', err.message);
        } else {
            event.reply('image-saved', tempPath);
        }
    });
});
```

Рисунок 2.16 – Програмна реалізація функції збереження зображення

## 2.3 Створення виконуваного файлу та ярлику застосунку

Фреймворк Electron надає можливість пакувати та створювати виконувані файли та ярлики застосунків. Для цього потрібно встановити Electron Packager за допомогою команди npm install electron-packager --save-dev та запустити скрипт для створення виконуваного файлу командою npm run package-win. Для оптимізації цього процесу можна також запустити процес створення пакету, шляхом створення інсталятора для застосунку командою npm run dist. Результатом виконання цих команд стане створення повноцінної програми з виконуваним файлом (рис. 2.17).

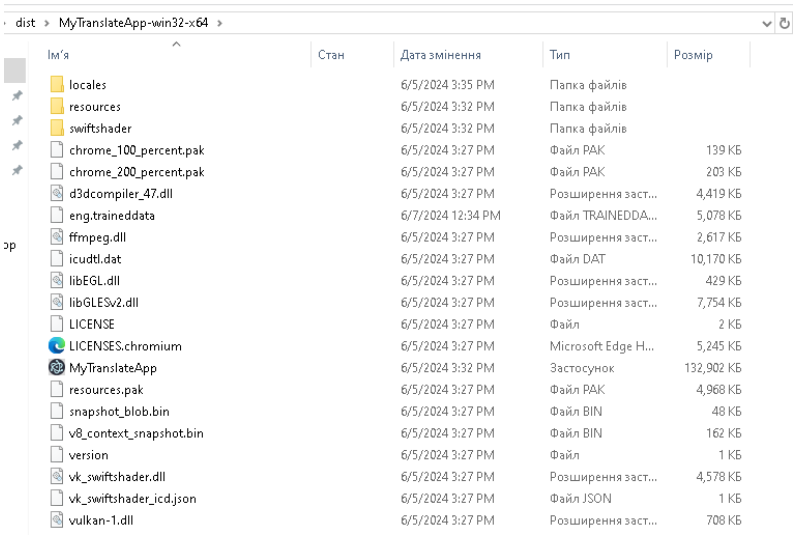


Рисунок 2.17 – Створення виконуваного файлу застосунку

## 2.4 Висновки до розділу 2

В межах даного розділу, відповідно до вимог, було створено застосунок із застосуванням фреймовку Electron, де була реалізована інтеграція з Amazon Translate, а також функції автоматичного розпізнавання мови його перекладу та розпізнавання тексту з зображення з його подальшим перекладом. Також було наведено зображення застосунку та створено його виконуваний файл.

# 3 ТЕСТУВАННЯ ТА НАЛАГОДЖЕННЯ

Тестування – процес перевірки програмного продукту з метою виявлення дефектів, помилок та недоліків. Розрізняють ручне та автоматизоване тестування.

В межах даного розділу, буде створений план тестування, а саме чекліст у вигляді таблиці та буде проведено тестування програмного застосунку за ним.

## 3.1 Вимоги до плану тестування та вибір способу тестування

Основними вимогами до тестування застосунку, відповідно до сайту QaTestLab [12] є:

- юзабіліті тестування – дослідження для визначення зручності використання ПЗ;
- тестування ефективності – перевірка необхідних обсягів коду і ресурсів QA, що використовуються програмою для виконання окремої функції;
- тестування продуктивності – перевірка швидкості роботи ПЗ або його окремих функцій;
- тестування швидкості відновлення – проводиться з метою визначення швидкості відновлення системи у разі софтверного креша (падіння програмного забезпечення) або помилки «заліза».

Для реалізації такого тестування доцільніше використовувати ручне тестування замість автоматизованого, оскільки це є найкращим способом протестувати зручність використання застосунку.

## 3.2 Створення чекліста та тестування застосунка

Чекліст – контрольний список дій, який допомагає систематизувати роботу, уникнути помилок і відстежувати процес [13]. Відповідно до поставлених вимог приклад чеклісту наведено в таблиці 3.1.

Таблиця 3.1 – Чекліст для тестування застосунка

| | **Результат тестування** | | |
|---|---|---|---|
| **Назва тесту** | **Успіх** | **Провал** | **Пропущено** |
| Застосунок відкривається | + | | |
| В застосунку відображається коректна назва | + | | |
| Можна змінювати вихідну та бажані мови | + | | |
| При перекладі тексту з однієї мови на іншу текст з’являється внизу застосунка | + | | |
| Є можливість автоматичного розуміння тексту та його перекладу на бажану мову | + | | |
| Є відображення помилки при спробі переведення пустого поля | + | | |
| Є можливість завантажувати зображення для перекладу | + | | |
| При завантаженні файлів не типу png або jpeg застосунок видає помилку | + | | |
| Швидкий запуск та переклад тексту | + | | |
| Можливість голосового перекладу тексту | + | | |
| Швидкий запуск застосунку після раптового закриття | + | | |

## 3.2 Висновки до розділу 3

В межах даного розділу було створено чекліст за яким було протестовано застосунок для автоматичного перекладу тексту. За результатами тестування застосунок пройшов усі тести.

# ВИСНОВКИ

Кваліфікаційна робота була спрямована на розробку сервісу автоматичного перекладу тексту із застосуванням мови програмування JavaScript та сервісу автоматичного перекладу тексту Amazon Translate. Для досягнення поставленої мети було проаналізовано сучасний стан проблематики у сфері автоматичного перекладу тексту, визначено інструменти для програмної реалізації автоматичного перекладу тексту, реалізовано та протестовано застосунок.

В першому розділі було здійснено вибір мови програмування, сервісу автоматичного перекладу тексту та фреймворку, а також висунуті вимоги до інструментів досягнення мети. За результатами порівняння різних інструментів, було обрано мову програмування JavaScript, сервіс автоматичного перекладу тексту Amazon Translate та фреймворк Electron.

В другому розділі було реалізовано програмна частина коду, зокрема створено бекенд та фронтенд застосунку, а також створені функції для розпізнавання, автоматичного перекладу тексту різними способами ( текстовий, голосовий, а також за допомогою розпізнавання зображення).

В третьому розділі були висунуті вимоги до тестування застосунку, за результатами яких створено чекліст для тестування застосунку.

При розробці застосунку було виявлено проблему з впровадженням функціоналу автоматичного перекладу тексту за допомогою голосового розпізнавання тексту через те, що Google припинило підтримку такого способу у Chromium, який використовується у фреймворці Electron. Вирішенням цієї проблеми стало застосування Microsoft Azure та його інтеграція у застосунок. На етапі тестування наявних проблем виявлено не було і застосунок успішно пройшов усі тестові кейси.

Для покращення роботи застосунку, в майбутньому можна додати історію перекладу тексту, а також можливість зберігати переведений текст у документ формату Word та зробити підтримку форматів Word, Pdf, тощо.

# ПЕРЕЛІК ПОСИЛАНЬ


1. Система автоматизації перекладу – це не машинний переклад. URL: https://www.itaf.biz/uk/systema-avtomatyzacii-perekladu-ce-ne-mashynnyj-pereklad/ (дата звернення: 20.03.2024).
2. Підручник з Python: вебсайт. URL: Підручник з Python — Python 3.12.3 documentation (дата звернення: 22.03.2024).
3. Переваги мови C++, яка і досі залишається актуальною. URL: https://dou.ua/forums/topic/42888/ (дата звернення: 23.03.2024).
4. Що таке JavaScript і для чого він потрібен. URL: https://goit.global/ua/articles/shcho-take-javascript-i-dlia-choho-vin-potriben/ (дата звернення: 24.03.2024).
5. Перекладайте документи, аудіо та відео в реальному часі за допомогою Google AI. URL: https://cloud.google.com/translate#how-it-works (дата звернення: 28.03.2024).
6. Що таке Crowdin та навіщо про нього знати нашим клієнтам. URL: https://mk-translations.ua/ua/blog/shho-take-crowdin-ta-navishho-pro-nogo-znati-nashim-kliyentam/ (дата звернення: 30.03.2024).
7. Amazon Translate Вільний і точний машинний переклад. URL: https://aws.amazon.com/translate/?nc1=h_ls (дата звернення: 31.03.2024).
8. Плюси та мінуси React Native. URL: https://www.netguru.com/blog/react-native-pros-and-cons (дата звернення: 03.04.2024).
9. Плюси та мінуси розробки Electron Js – чи варто ним користуватися? URL: https://medium.com/@emmaw4430/pros-and-cons-of-electron-js-development-should-you-use-it-407f67dae310 (дата звернення: 06.04.2024).
10. w3school: вебсайт. URL: https://www.w3schools.com/tags/tag_body.asp (дата звернення: 15.04.2024).
11. Основи CSS. URL: https://www.expression-web-tutorials.com/css-basics.html (дата звернення: 20.04.2024).

12. Огляд видів тестування. URL: https://training.qatestlab.com/blog/technical-articles/review-the-types-of-testing/ (дата звернення: 05.05.2024).
13. Що таке чек-лист? Призначення, застосування та 10 правил створення ефективного чек-листа. URL: https://web-promo.ua/ua/blog/chto-takoe-chek-list-naznachenie-primenenie-i-10-pravil-sozdaniya-effektivnogo-chek-lista/ (дата звернення: 10.05.2024).

# ДОДАТОК А

## Програмна реалізація інтерфейсу

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Service of Automatic Translation</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
<h1>Service of Automatic Translation</h1>

<div>
    <label for="sourceLanguage">Source Language:</label>
    <select id="sourceLanguage">
        <!-- Options will be dynamically populated by JavaScript -->
    </select>
</div>
<div>
    <label for="targetLanguage">Target Language:</label>
    <select id="targetLanguage">
        <!-- Options will be dynamically populated by JavaScript -->
    </select>
</div>

<div>
    <h3>Text Input</h3>
    <textarea id="textInput" placeholder="Enter text here"></textarea>
```

```
    <button onclick="translateText()">Translate Text</button>
</div>

<div>
    <h3>Image Input</h3>
    <input type="file" id="imageInput" accept="image/*">
    <button onclick="translateImage()">Translate Image Text</button>
</div>

<div>
    <h3>Voice Input</h3>
    <button onclick="startVoiceRecognition()">Start Voice Recognition</button>
    <button onclick="stopVoiceRecognition()">Stop Voice Recognition</button>
    <p id="voiceText">Recognized text will appear here...</p>
    <button onclick="translateVoiceText()">Translate Voice Text</button>
</div>

<h2>Translated Text</h2>
<p id="translatedText"></p>

<script src="renderer.js"></script>
</body>
</html>
```

# ДОДАТОК Б

## Програмна реалізація стилізації інтерфейсу

```
body {
    font-family: Arial, sans-serif;
    padding: 20px;
}

h1, h3 {
    color: #333;
}

label, textarea, input, button, select {
    display: block;
    margin: 10px 0;
}

textarea {
    width: 100%;
    height: 100px;
}

input[type="file"] {
    margin: 10px 0;
}
```

# ДОДАТОК В

## Програмна реалізація мосту між main.js та renderer.js

```
const { contextBridge, ipcRenderer } = require('electron');

contextBridge.exposeInMainWorld('electron', {
  processImage: (filePath) => ipcRenderer.send('process-image', filePath),
  onImageText: (callback) => ipcRenderer.on('image-text', (event, text) => callback(text)),
  onImageTextError: (callback) => ipcRenderer.on('image-text-error', (event, error) => callback(error)),

  translateText: (text, sourceLanguage, targetLanguage) => ipcRenderer.send('translate-text', { text, sourceLanguage, targetLanguage }),
  onTranslatedText: (callback) => ipcRenderer.on('translated-text', (event, translatedText) => callback(translatedText)),
  onTranslateError: (callback) => ipcRenderer.on('translate-error', (event, error) => callback(error)),

  startVoiceRecognition: (subscriptionKey, serviceRegion, sourceLanguage) => ipcRenderer.send('start-voice-recognition', { subscriptionKey, serviceRegion, sourceLanguage }),
  stopVoiceRecognition: (subscriptionKey, serviceRegion, sourceLanguage) => ipcRenderer.send('stop-voice-recognition'),
  onVoiceRecognizedText: (callback) => ipcRenderer.on('voice-recognized-text', (event, voiceText) => callback(voiceText)),

  saveImage: (fileData, fileName) => ipcRenderer.send('save-image', fileData, fileName),
```

```
  onImageSaved: (callback) => ipcRenderer.on('image-saved', (event, filePath) =>
callback(filePath)),
  onImageSaveError: (callback) => ipcRenderer.on('image-save-error', (event, error)
=> callback(error))
});
```

# ДОДАТОК Г

## Програмна реалізація основних функцій застосунку

```
const languages = {
  'auto': 'Auto',
  'af': 'Afrikaans',
  'sq': 'Albanian',
  'am': 'Amharic',
  'ar': 'Arabic',
  'hy': 'Armenian',
  'az': 'Azerbaijani',
  'bn': 'Bengali',
  'bs': 'Bosnian',
  'bg': 'Bulgarian',
  'ca': 'Catalan',
  'zh': 'Chinese (Simplified)',
  'zh-TW': 'Chinese (Traditional)',
  'hr': 'Croatian',
  'cs': 'Czech',
  'da': 'Danish',
  'nl': 'Dutch',
  'en': 'English',
  'et': 'Estonian',
  'fi': 'Finnish',
  'fr': 'French',
  'de': 'German',
  'el': 'Greek',
  'gu': 'Gujarati',
  'ht': 'Haitian Creole',
```

'ha': 'Hausa',

'he': 'Hebrew',

'hi': 'Hindi',

'hu': 'Hungarian',

'is': 'Icelandic',

'id': 'Indonesian',

'ga': 'Irish',

'it': 'Italian',

'ja': 'Japanese',

'kn': 'Kannada',

'kk': 'Kazakh',

'ko': 'Korean',

'lv': 'Latvian',

'lt': 'Lithuanian',

'mk': 'Macedonian',

'ms': 'Malay',

'ml': 'Malayalam',

'mt': 'Maltese',

'mn': 'Mongolian',

'no': 'Norwegian',

'fa': 'Persian',

'pl': 'Polish',

'pt': 'Portuguese',

'pa': 'Punjabi',

'ro': 'Romanian',

'ru': 'Russian',

'sr': 'Serbian',

'sk': 'Slovak',

'sl': 'Slovenian',

'so': 'Somali',

```
  'es': 'Spanish',
  'sw': 'Swahili',
  'sv': 'Swedish',
  'tl': 'Tagalog',
  'ta': 'Tamil',
  'te': 'Telugu',
  'th': 'Thai',
  'tr': 'Turkish',
  'uk': 'Ukrainian',
  'ur': 'Urdu',
  'uz': 'Uzbek',
  'vi': 'Vietnamese',
  'cy': 'Welsh',
  'xh': 'Xhosa',
  'yi': 'Yiddish',
  'yo': 'Yoruba',
  'zu': 'Zulu'
};


document.addEventListener('DOMContentLoaded', () => {
  const sourceLanguageSelect = document.getElementById('sourceLanguage');
  const targetLanguageSelect = document.getElementById('targetLanguage');

  for (const [code, name] of Object.entries(languages)) {
    const sourceOption = document.createElement('option');
    sourceOption.value = code;
    sourceOption.textContent = name;
    sourceLanguageSelect.appendChild(sourceOption);
```

```
    if (code !== 'auto') {
      const targetOption = document.createElement('option');
      targetOption.value = code;
      targetOption.textContent = name;
      targetLanguageSelect.appendChild(targetOption);
    }
  }


  sourceLanguageSelect.value = 'auto';
  targetLanguageSelect.value = 'uk';


  console.log('Source Language Options:', sourceLanguageSelect.innerHTML);
  console.log('Target Language Options:', targetLanguageSelect.innerHTML);
});

function translateText() {
  const sourceLanguage = document.getElementById('sourceLanguage').value;
  const targetLanguage = document.getElementById('targetLanguage').value;
  const text = document.getElementById('textInput').value;

  window.electron.translateText(text, sourceLanguage, targetLanguage);
}

window.electron.onTranslatedText((translatedText) => {
  document.getElementById('translatedText').innerText = translatedText;
});

window.electron.onTranslateError((error) => {
```

```
  console.error('Translation Error:', error);
  alert('Failed to translate text.');
});

function translateImage() {
  const imageInput = document.getElementById('imageInput');
  if (imageInput.files.length === 0) {
    alert('Please select an image file.');
    return;
  }

  const file = imageInput.files[0];
  const fileReader = new FileReader();

  fileReader.onload = function(event) {
    const arrayBuffer = event.target.result;
    window.electron.saveImage(arrayBuffer, file.name);
  };

  fileReader.readAsArrayBuffer(file);
}

window.electron.onImageSaved((filePath) => {
  window.electron.processImage(filePath);
});

window.electron.onImageText((text) => {
  document.getElementById('textInput').value = text;
  translateText(); // Translate extracted text
});
```

```
window.electron.onImageTextError((error) => {
  console.error('Image Text Extraction Error:', error);
  alert('Failed to extract text from the image.');
});
function startVoiceRecognition() {
  const subscriptionKey = '083e42fe395640ad8f0f0d21d3344391';
  const serviceRegion = 'northeurope';
  const sourceLanguage = document.getElementById('sourceLanguage').value;
  window.electron.startVoiceRecognition(subscriptionKey, serviceRegion,
sourceLanguage);
}
window.electron.onVoiceRecognizedText((voiceText) => {
  document.getElementById('voiceText').innerText = voiceText;
  document.getElementById('textInput').value = voiceText;
  translateText(); // Translate text after receiving voice input
});
function stopVoiceRecognition() {
  window.electron.stopVoiceRecognition();
}
function translateVoiceText() {
  const sourceLanguage = document.getElementById('sourceLanguage').value;
  const targetLanguage = document.getElementById('targetLanguage').value;
  const voiceText = document.getElementById('voiceText').innerText;

  window.electron.translateText(voiceText, sourceLanguage, targetLanguage);
}
```

# ДОДАТОК Д

## Програмна реалізація створення вікна застосунку

```
const { app, BrowserWindow, ipcMain } = require('electron');
const path = require('path');
const Tesseract = require('tesseract.js');
const AWS = require('aws-sdk');
const speechsdk = require('microsoft-cognitiveservices-speech-sdk');
const fs = require('fs');

// AWS Translate Configuration
AWS.config.update({
    accessKeyId: 'Enter KeyID',
    secretAccessKey: ' Enter AccessKey',
    region: 'Enter Region'
});
const translate = new AWS.Translate({ region: 'Enter Region' });

function createWindow() {
    const mainWindow = new BrowserWindow({
        width: 800,
        height: 600,
        webPreferences: {
            preload: path.join(__dirname, 'preload.js'),
            contextIsolation: true,
            enableRemoteModule: false,
            nodeIntegration: false
        }
    });
```

```
    mainWindow.loadFile('index.html');
}

app.on('ready', createWindow);

app.on('window-all-closed', () => {
    if (process.platform !== 'darwin') {
        app.quit();
    }
});

app.on('activate', () => {
    if (BrowserWindow.getAllWindows().length === 0) {
        createWindow();
    }
});

ipcMain.on('process-image', (event, filePath) => {
    Tesseract.recognize(filePath, 'eng', {
        logger: m => console.log(m)
    }).then(({ data: { text } }) => {
        event.reply('image-text', text);
    }).catch(err => {
        console.error('Error:', err);
        event.reply('image-text-error', err.message);
    });
});

ipcMain.on('translate-text', (event, { text, sourceLanguage, targetLanguage }) => {
    const params = {
```

```
        Text: text,
        SourceLanguageCode: sourceLanguage,
        TargetLanguageCode: targetLanguage
    };

    translate.translateText(params, (err, data) => {
        if (err) {
            console.log(err, err.stack);
            event.reply('translate-error', err.message);
        } else {
            event.reply('translated-text', data.TranslatedText);
        }
    });
});

let recognizer = null;

ipcMain.on('start-voice-recognition', async (event, { subscriptionKey, serviceRegion, sourceLanguage }) => {
    const speechConfig = speechsdk.SpeechConfig.fromSubscription(subscriptionKey, serviceRegion);
    speechConfig.speechRecognitionLanguage = sourceLanguage !== 'auto' ? sourceLanguage : 'en-US';

    const audioConfig = speechsdk.AudioConfig.fromDefaultMicrophoneInput();
    recognizer = new speechsdk.SpeechRecognizer(speechConfig, audioConfig);

    recognizer.recognizeOnceAsync(result => {
        const voiceText = result.text;
        event.reply('voice-recognized-text', voiceText);
```

```
    });
});

ipcMain.on('stop-voice-recognition', async (event) => {
    recognizer.stopContinuousRecognitionAsync();
});

ipcMain.on('save-image', (event, fileData, fileName) => {
    const tempPath = path.join(app.getPath('temp'), fileName);
    fs.writeFile(tempPath, Buffer.from(fileData), (err) => {
        if (err) {
            console.error('Error saving file:', err);
            event.reply('image-save-error', err.message);
        } else {
            event.reply('image-saved', tempPath);
        }
    });
});
```