МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

СЛОБОДЯНЮК СЕРГІЙ СЕРГІЙОВИЧ

Допускається до захисту:
в.о. завідувача кафедри
інформаційних технологій
канд. техн. наук, доцент
_____ О. В. Зелінська
«___»_____2025 р.

РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ГОЛОСОВОГО ПЕРЕКЛАДУ В РЕЖИМІ РЕАЛЬНОГО ЧАСУ («VOICE TRANSLATOR»)

Спеціальність 122 Комп’ютерні науки

Кваліфікаційна (бакалаврська) робота

Керівник:
О. В. Зелінська, доцент кафедри
інформаційних технологій,
к.т.н., доцент
_____

Оцінка: _____ / _____ / _____
(бали/за шкалою ЄКТС/за національною шкалою)
Голова ЕК:_____

Вінниця – 2025

АНОТАЦІЯ

Слободянюк С. С. Розробка програмного забезпечення для голосового перекладу в режимі реального часу («Voice Translator»). Спеціальність 122 «Комп'ютерні науки», освітня програма «Комп'ютерні науки». Донецький національний університет імені Василя Стуса, Вінниця. 2025.

У кваліфікаційній (бакалаврській) роботі досліджено та проаналізовано системи автоматичного перекладу, синтез мовлення в реальному часі, використання цих технологій для розробки голосового перекладача. За допомогою таких технологій як: Java, JavaFx, Vosk, LiberTranslate, Docker, RHVoice, бібліотек Java, було розроблено програму, яка отримувала аудіо потік від одного з співрозмовників, розпізнавала мовлення, перекладала та озвучувала переклад другому співрозмовнику, який запустив цю програму.

Ключові слова: розробка, перекладач, синтез мовлення, розпізнавання мовлення, голосовий чат, Java.

57 ст., 14 рис., 7 табл., 26 джерел.

ABSTRACTS

Slobodianiuk S. Development of software for real-time voice translation ('Voice Translator'). Speciality 122 'Computer Science', educational programme 'Computer Science'. Vasyl' Stus Donetsk National University, Vinnytsia, 2025.

In the qualification (bachelor's) thesis, the author researched and analysed automatic translation systems, real-time speech synthesis, and the use of these technologies for the development of a voice translator. With the help of such technologies as: Java, JavaFx, Vosk, LiberTranslate, Docker, RHVoice, and the Java built-in library, a program was developed that received an audio stream from one of the interlocutors, recognised speech, translated it, and voiced the translation to the second interlocutor who launched the program.

Keywords: development, translator, speech synthesis, speech recognition, voice chat, Java.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ASR – Automatic Speech Recognition.

MT – Machine Translate.

TTS – Text-to-Speech.

AWS – Amazon Web Service.

HIPAA – Health Insurance Portability and Accountability Act.

БД – База даних.

DB – Data Base.

МП – Машинний переклад.

UI – User Interface.

CLI – Command-Line Interface.

JVM – Java Virtual Machine.

PCM – Pulse-Code Modulation.

GUI – Graphical User Interface.

ПЗ – Програмне Забезпечення.

ВСТУП

Актуальність теми дослідження.

Глобалізація, розвиток міжнародних зв’язків, зростання популярності онлайн-спілкування та віддаленої роботи загострюють проблему мовного бар’єру. Особливо цю проблему відчутно у голосових чатах (геймінг-платформах, віддалені зустрічі, тощо), де незнання іншої мови, робить спілкування та розуміння важким. Хоча існують сервіси для перекладання та озвучення перекладу, але вони не створюють легкі умови для спілкування та зазвичай мають обмежений, безкоштовний функціонал. Отже, саме тому, пропонується програмне забезпечення, яке буде підключатися до голосового чату, розпізнавати аудіопотік, перекладати та озвучувати його, і все це в режимі реального часу. Такий підхід, має полегшити спілкування в різних голосових чатах між іноземцями.

Метою дослідження є розробити програму голосового перекладача, який у режимі реального часу, забезпечує розпізнавання мовлення, автоматичний переклад та озвучення.

Завдання дослідження.

1. Ознайомитися та узагальнити сучасні технології розпізнавання мовлення, перекладу та синтезу мовлення.
2. Проаналізувати обрані інструменти для програмного забезпечення (Vosk, LibreTranslate, RHVoice) та обґрунтувати їх вибір.
3. Розробити клієнт-серверну систему голосового чату для передачі аудіопотоку між клієнтами.
4. Реалізувати програмні модулі.
5. Застосувати програму на практиці, виявивши недоліки.
6. Обґрунтувати можливість подальшого вдосконалення системи.

Об’єктом дослідження є процес автоматичного голосового перекладу в умовах онлайн-спілкування через аудіочат.

Предметом дослідження є компоненти та алгоритми, що забезпечують перетворення аудіосигналу англійською мовою в український аудіовідгук: ASR-модуль, MT-модуль, TTS-модуль, а також механізм їх інтеграції в єдину систему.

Результати роботи доповідались на Всеукраїнській конференції: Актуальні проблеми гуманітарних, технічних і природничих наук. Також, на III Всеукраїнської науково-практичної конференції: Комп’ютерні технології обробки даних.

РОЗДІЛ 1
ОГЛЯД ТЕХНОЛОГІЙ ГОЛОСОВОГО ПЕРЕКЛАДУ

1.1 Технології розпізнавання мовлення

Автоматичне розпізнання мовлення – це досить складна технологія, яку доволі важко розробити, адже в світі існує велика кількість мов з різними діалектами та акцентами, що робить розпізнавання мови, важким процесом.

ASR застосовує методи обробки природної мови та машинного перекладу для власної розробки. Використовуючи в програмному забезпечені різні механізми для вивчення мови, а розробники забезпечують точність розпізнавання мовлення. Основні етапи включають: збір звуку, попередню обробку, виділення характеристик, акустичне моделювання, моделювання мови та декодування [10].

До етапів ASR відносяться запис звуку, спочатку мікрофон вловлює мовлення користувача, після чого перетворює аудіо в електричний сигнал. Попередня обробка аудіо, електричний сигнал оцифровується та обробляється на зменшення шуму для покращення якості. Витяг функції, цифрове аудіо проаналізовується, щоб виділити висоту тону, спектральні коефіцієнти та інші акустичні характеристики. Акустичне моделювання, проаналізовані характеристики порівнюються з попередньо навченими моделями для відображення аудіо на окремі звуки мови. Моделювання мови, розпізнані звуки складаються в слова та фрази за допомогою попередньо встановлених мовних пакетів на основі контексту. Декодування, декодується найбільш ймовірна послідовність слів, яка збігається з вхідним аудіо варіантом [7].

Прикладами використання ASR-технологій є віртуальні помічники та розумні пристрої, ASR є основним компонентом віртуальних помічників, наприклад Siri. Це дозволяє керувати різними пристроями, сервісами, розумним домом, використовуючи лише голос. Найпопулярніші такі продукти: Google Assistant, який розроблений у 2016 році, є одним з найкращих помічником для

спілкування в чаті. Його точність близько 95% англійською мовою. Siri – помічник від Apple, який доступний у понад 30 країнах та володіє 21 мовами. Siri є першою системою, що призвела фурор використання технології розпізнавання мови. Amazon Alexa – також, доволі популярний помічник, яким користуються більше 100 мільйонів людей у світі. Автомобільна та транспортна, ASR, що використовується в автомобільних інформаційно-розважальних системах. Ці технології дозволяють водіям легко керувати різними функціями за допомогою голосу, наприклад: перемикання музики, навігатор, клімат-контроль [5]. Охорона здоров'я, ASR дає можливість лікарям легше вести нотатки, записи, документацію, що набагато спрощує процес. ASR часто використовується в різних кол-центрах для взаємодії з клієнтами, що підвищує продуктивність та покращує результати. ASR допомагає вивчати мову для людей, надаючи правильну вимову або різних мовних навичок, що робить процес вивчення легшим. Технологія ASR допомагає людям з проблемами зі слухом, чути те, що відбувається навколо, що дає таким людям почуватися як всі.

До переваг ASR можна віднести ефективність, технології ASR дозволяють ефективно вводити дані, використовуючи лише мовлення, доступність, полегшує використання пристроїв для людей з обмеженими можливостями, економічна ефективність, ASR зменшує потребу в ручній транскрипції, використовуючи голосові вводи, що економить час та ресурси.

Недоліками ASR є акценти та діалект, різні акценти та діалекти можуть не правильно оброблятися технологією, що може сказатися на отриманих результатах, фоновий шум, посторонні звуки можуть заважати правильно розпізнати вхідні дані, омофони, слова, які звучать однаково, можуть завести в оману технології ASR, що призведе до не правильних результатів.

Порівняння найкращих рішень для розпізнавання мовлення.

Transkriptor – провідний інструмент розпізнавання мовлення. Цей інструмент є одним із найточніших на ринку, що пропонує ефективність виконання та зрозумілий інтерфейс. Transkriptor може легко підключатися до різних зустрічей та робити транскрибацію їх. Унікальність цього інструменту в

тому, що в цій системі є вбудований помічник Tor на базі штучного інтелекту, що якісно перетворює дані на змістовний ресурс. Tor дуже якісно аналізує вхідні дані, розуміючі ключові моменти та може надати короткий опис отриманих даних.

Основні характеристики:

✓ Висока точність (близько 99%): сервіс забезпечує надійну транскрибацію.

✓ Розширена мовна підтримка: сервіс підтримує понад 100 мов.

✓ Швидкий час виконання: Transkriptor швидко обробляє вхідні дані, що дає можливість заощадити час.

✓ Помічник на базі штучного інтелекту: вбудований ШІ Tor дає можливість отримувати звіти по транскрибації та можливість спілкування з ним [3].

Google Speech-to-Text – потужний сервіс для розпізнавання мовлення, доступний в Google Cloud. Ця технологія розроблена в основному для програмістів, щоб ті використовували його у своїх програмах. Google Speech-to-Text дуже добре справляється з потоковою транскрипцією в режимі реального часу, що дозоляє впроваджувати у голосові інтерфейси.

Основні характеристики:

✓ Підвищена точність для живого аудіо: ефективно розпізнає мовлення в режимі реально часу, дуже добре працює зі спонтанністю та перериванням мовлення.

✓ Найкраща базова модель: Speech-to-Text була визнана провідною базовою моделлю для різних програм з розпізнавання мовлення, що пропонує розробникам надійний варіант для їхніх програм.

Amazon Transcribe – це ASR-сервіс, який пропонує Amazon Web Services. Цей сервіс також розроблений для програмістів, що хочуть використати цю технологію в своїх проєктах. Але AWS ще й надає інструменти та консолі, які дозволяють використовувати технологію як plug-and-play.

Основні характеристики:

✓ Аналітика дзвінків: розроблені інструменти, які спеціально створені для аналізу дзвінків, дозволяють проводити аналіз настроїв та визначення ключових фраз.

✓ Медична транскрипція: транскрипція, яка відповідає вимогам HIPAA, що використовується в медичних цілях та забезпечує повну конфіденційність даних [9].

Microsoft Azure Speech – цей сервіс схожий з попереднім, але є частиною Microsoft, що дає змогу легко інтегруватися в Microsoft Office 365, Dynamics 365, Teams. Цим сервісом зазвичай користуються організації, які інвестують або використовують інші продукти Microsoft. Цю технологію також можуть використовувати розробники для інтеграції в свій проєкт.

Основні характеристик:

✓ Уніфікована служба мовлення: поєднує в собі як перетворення мовлення на текст, так і текст на мовлення.

✓ Настроювані моделі: точність налаштування мовних моделей під конкретні галузі.

Speechmatics – провідний постачальник високоточних технологій розпізнавання мовлення. Сервіс пропонує API для програмістів та готові надати рішення для бізнесу, які спеціалізуються на розпізнаванні складних аудіо умов. Speechmatics має більш гнучке використання API, ніж вище згадані хмарні технології, що дає розробникам більше свободи для інтеграції в проєкт. Також, потрібно розуміти, що для інтеграції цієї технології, потрібні знання в програмуванні, адже сервіс не працює за принципом «підключи і працюй», але гнучкість і контроль – це те, заради чого варто докласти зусиль.

Основні характеристики:

✓ Глобальне мовне покриття: велика підтримка багатьох мов та акцентів, що робить розпізнавання легшим та кращим.

✓ Висока точність: Speechmatics забезпечує високу точність розпізнавання, навіть при наявності шуму.

Отже, було ознайомлено з основними етапами ASR, розглянуто сфери використання та найкращі рішення для розпізнавання мовлення на ринку.

1.2 Аналіз систем автоматичного перекладу

Системи автоматичного перекладу або CAT-програми, це програмне забезпечення, що автоматизує та прискорює процес перекладу тексту. Використовуючи комп'ютерні алгоритми і технології машинного навчання, програми перекладають тексти з однієї мови на іншу.

До основних функцій автоматичного перекладу можемо віднести машинний переклад, програма самостійно перекладає тексти. Системи керування перекладами, допомагають організувати та керувати процесом перекладу, зберігаючи попередні переклади та термінологію. Translation Memory (TM), БД, які містять перекладені фрагменти тексту, що в подальшому допомагає уникнути дублювання та покращує якість перекладу. Автоматичний аналіз, програми, які проаналізовують тексти на повтори та інші особливості, що можуть впливати на переклад. Переклад термінології, допомагають якісно та чітко перекладати терміни, щоб зберігалася точність та узгодженість [12].

Машинний переклад (МП) – простими словами, коли без участі людини, комп'ютерна програма бере сама участі в перекладі тексту. МП виконує просту заміну слів вхідного тексту на слова вихідного тексту [11].

Корпусні методи дають можливість проводити складніші переклади. Вони працюють краще з фонетичною топологією, ідіомами та з визначенням нетипових конструкцій.

До типів машинного перекладу можна віднести статистичний МП, цей метод працює на основі статичних моделей, які збираються на основі вивчення великої купи різного контенту – оригіналу та перекладу. Статистичний МП визначає відповідності між словами вихідної мови та словами цільової мови. Перевагами якого є автоматизоване навчання на великих корпусах; швидкий старт за наявності даних, а недоліками – обмежена здатність до глобального

контексту; часті помилки узгодження слів та стилістичної єдності. МП на основі правил, ця модель проводить граматичний аналіз вхідного тексту і потім будує переклад з урахуванням граматичних правил. Результат такого перекладу треба редагувати. Перевагами є керування процесом перекладу на кожному рівні; передбачуваня якості при вичерпному охоплені правил. Недоліками є висока трудомісткість розробки та підтримки правил для кожної пари мов; погано масштабується на нові домени та стилістику. Гібридний МП: цей метод є сумішшю попередніх двох методів, тобто у цій моделі використовується пам'ять перекладів і граматичні правила. Але все одно цей метод потребує редагування збоку професіоналів. Так, як це гібрид двох попередніх моделей, то ця модель має ті самі переваги та недоліки. Нейронний МП: ця модель використовує підхід штучного інтелекту для перекладу тексту. Перевагою є те, що модель надає єдину систему для роботи з вхідними та вихідними текстами [2]. До переваг можна віднести плавність, узгодженість та покращена якість перекладу, навіть для складних речень; здатність до «тонкого налаштування» на спеціалізованих доменних даних; можливість мультимовних моделей (одна велика модель обслуговує багато пар мов). Недоліками є високі обчислювальні ресурси для тренування та розгортання; ризик генерації неправдивого або контекстуального невідповідного тексту.

Можна виділити основні переваги МП, швидкість, комп'ютерні програми перекладають тексти швидше, адже людина-перекладач, хоч і точно перекладає, але не зрівняється по швидкості з комп'ютером; вартість, якщо обрати та навчити машину під обрані вимоги, то МП забезпечить економний та ефективний результат [1].

Отже, в цьому розділі було оглянуто основні функції систем автоматичного перекладу, детальніше розглянуто машинний переклад та його типи.

1.3 Синтез мовлення в реальному часі

Синтез мовлення (Text-to-Speech, TTS) є завершальним етапом у системі голосового перекладачу, який отримує перекладений текст та відправляє його назад у звуковому сигналі. У контексті реального часу, головними вимогами до TTS є мінімальна латентність, природність та зрозумілість голосу.

Синтез мовлення відбувається в три етапи: текст у слова, слова у фонеми, фонеми у звуки [6]. Текст у слова, процес синтезу мовлення розпочинається з попередньої обробки або нормалізації тексту, яка допомагає зменшити неоднозначність, обираючи найдоречніший варіант озвучування. Цей етап включає аналіз і очищення тексту, що дозволяє системі точніше його «прочитати». Особливої уваги потребують такі елементи, як числа, дати, час, скорочення, абревіатури та спеціальні символи – їх потрібно правильно інтерпретувати. Для цього використовують методи статистичного аналізу або нейронні мережі, які допомагають визначити найімовірніші звучання. Омографи – слова з однаковим написанням, але різним значенням – також потребують обробки перед озвученням. Наприклад, фраза «Я продаю машину» може бути неправильно інтерпретована, оскільки слово «продаю» може мати інше звучання та значення. Завдяки контексту система може зробити правильний вибір. Технології розпізнавання мовлення дозволяють перетворювати усне мовлення на текст навіть тоді, коли воно містить складні або неоднозначні слова. Слова у фонеми: після розпізнавання слів, синтезатор мовлення формує звукові сигнали, що відповідають цим словам. Для цього кожна система потребує великого словника з переліком слів та вказівками щодо їхньої правильної вимови. Зокрема, необхідно мати список фонем – найменших одиниць звуку, з яких складаються слова. Це особливо важливо, оскільки в англійській мові є лише 26 літер, але понад 40 фонем. У теорії процес виглядає таким чином, що комп’ютер знаходить слово в словнику, отримує відповідну послідовність фонем і озвучує її. Але на практиці це завдання значно складніше через мовні нюанси та винятки. Альтернативний підхід полягає у поділі написаного слова на графеми (буквені

одиниці) та застосуванні правил, за якими кожна з них перетворюється на відповідну фонему. Такий метод дозволяє обійтися без повного словника, використовуючи мовні закономірності для генерації звучання. Фонеми у звуки, після чого, як комп'ютер перетворив текст у список фонем, виникає питання, як визначити ключові фонеми, які необхідно озвучити під час синтезу мовлення різними мовами?

Існує три основні підходи для вирішення цього питання, перший підхід це використання записів живої мови, де реальні люди вимовляють слова, що потім аналізуються на фонемному рівні. Другий підхід полягає у генерації фонем комп'ютером на основі базових акустичних характеристик звуків, таких як частота, амплітуда та тривалість. Третій, найсучасніший метод, це моделювання фізіологічного механізму людського голосу в реальному часі. Він забезпечує природніше звучання завдяки застосуванню високоточних алгоритмів синтезу.

До компонентів сучасного TTS-пайплайна можемо віднести лінгвістична обробка тексту, нормалізація тексту, розгортання чисел, абревіатур, спеціальних символів у словах, токенізація та маркування частин мови необхідна для правильного розставляння наголосів і пауз. Акустичне моделювання, побудова мови з попередньо записаних фрагментів, низька гнучкість, великі розміри ресурсів. Параметричні методи (HMM-базовані), синтез на основі параметрів вокодера, природність нижча за сучасні нейромережеві рішення. Нейронні методи, Seq2Seq-модель із механізмом уваги, що генерує спектрограми; FastSpeecg, безуважні архітектури для значно зниженої латентності. Генерація хвильового сигналу (Vocoder), WaveNet (висока якість, але велика затримка), Parallel WaveGAN, MelGAN, HiFi-GAN (баланс якості та швидкості); Поточний тренд – поєднання FastSpeech 2 + HiFi-GAN для наднизької затримки та високої природності.

Реалізація синтезу мовлення в режимі реального часу включає використання стрімінгових TTS-архітектур, які поступово генерують і відтворюють аудіо по фреймах без очікування повного тексту, а також оптимізацію інферінгу за допомогою квантування моделі (наприклад, до INT8),

використання GPU/TPU або DSP-чіпів на пристроях. Важливою складовою є буферизація та керування паузами, що передбачає динамічне коригування розміру буфера для мінімізації спотворень і розривів. У цьому контексті застосовуються як open-source рішення, зокрема Coqui TTS із підтримкою FastSpeech та HiFi-GAN, так і фреймворк NVIDIA NeMo, який пропонує попередньо навчені моделі для низьколатентного стрімінгового TTS. Крім того, хмарні сервіси на кшталт Google Cloud TTS і Azure Speech дозволяють швидко розгорнути систему з широким вибором голосів, хоча їх ефективність залежить від стабільності мережевого з'єднання.

Окрім уже згаданих нейронних нейронних підходів, у світі TTS застосовують і інші технології, кожна з яких має свої сильні та слабкі сторони. Конкатенативний синтез (Unit-selection TTS), система складає вихідне мовлення із заздалегідь записаних фрагментів, вибираючи найбільш підходящі юніти на базі пошуку у великому «бібліотечному» копусі. Дуже природний тембр, мінімальні спотворення в матеріалі. Потрібний великий об'єм записаних даних; Складність у покритті всіх комбінацій звуків; Високі вимоги до пошукових алгоритмів та індексації. Параметричний синтез (HMM-базований, parametric TTS), на основі статистичних моделей (HMM) генерує параметри, а потім відтворює їх через вокодер. Компактні моделі, легко адаптуються на голосові дані невеликого обсягу. Але голос звучить «штучно» або «металево» через обмежену пропускну здатність вокодера, низька природність у порівнянні з конкатенативними або сучасними нейронними методами. Нейронні вокодери (WaveNet-подібні та їх оптимізації), автокорельована модель, що генерує звукову хвилю по семплу: висока якість, але дуже велика затримка. Parallel WaveGAN, MelGAN, WaveGlow, генеративні моделі на основі GAN чи flow-архітектур: значно швидші, але іноді менш природні. Flow-and-diffusion-базовані моделі, Flow-моделі (WaveGlow, Flowtron) перетворюють простий розподіл у складний звуковий сигнал за детерміністичними кроками: забезпечують високу швидкість та відтворюють деталізований спектр. Diffusion-TTS (DiffWave, Grad-TTS) починають із випадкового шуму і поступово очищають його до звукового

сигналу: відносно новий підхід, що демонструє дуже високу якість і гнучкість, але потребує кількох ітерацій очищення. Експресивний та емоційний TTS, навчаються не лише текст-аудіо, а додають теги для емоцій, стилю, інтенсивності. Style tokens/Global style embeddings, Vector-записи, зо задають voice style під час синтезу. Zero-shot voice cloning, моделі, які можуть клонувати голос нового диктора за кількома секундами зразків. Голосова конверсія (Voice Conversion, VC), не змінює текст, але трансформує один голос у голос іншого диктора, зберігаючи зміст. Використовується в діалогових системах для уніфікації тембру або персони. До методів можем віднести автоенкодери (VAE-based), GAN-підходи (StarGAN-VC), flow-моделі (Flow-VC). Edge та мобільні рішення, спрощені архітектури для запуску на мікроконтролерах або мобільних CPU, з дуже низьким обсягом пам’яті та обчислень [13].

У цьому розділі детально описано етап синтезу мовлення в реальному часі: від лінгвістичної нормалізації тексту до генерації звукового сигналу через нейронні вокодери. Розглянуто різні архітектури, а також стратегії оптимізації для низької латентності та високої якості.

1.4 Етапи реалізації проєкту

Успішна реалізація голосового перекладача потребує поетапного підходу – від збору вимог до оптимізації готового рішення. Спершу аналізуються функціональні (мовні пари, точність ASR, формат інтерфейсу) та нефункціональні вимоги (швидкодія, автономність, ресурсоємність). Далі здійснюється вибір компонентів: ASR (наприклад, Vosk), машинного перекладу (трансформерні моделі з прийнятною затримкою), TTS-системи та платформи для інтеграції в голосовий чат. Архітектура проєкту будується на модульній основі з чітким поділом на ASR, MT, TTS, комунікаційний шар та UI, визначаються API і схема обміну даними. На етапі реалізації створюється прототип із наскрізною обробкою: від розпізнавання мовлення до синтезованого перекладу. Тестування охоплює функціональні, регресійні, навантажувальні та

юзабіліті-аспекти. Завершальний етап включає оптимізацію інферінгу (квантування, багатопоточність, прискорення), покращення точності моделей та удосконалення UI/UX для зручності користувача. Етапи реалізації проєкту описують повний життєвий цикл розробки голосового перекладача: від збору вимог і вибору технологічних компонентів до проєктування, прототипування, тестування, оптимізації, розгортання та підтримки [39, 40].

Такий поетапний підхід гарантує контроль якості на кожному кроці та дозволяє вчасно виявляти та усувати ризики, забезпечуючи своєчасне виконання завдання.

РОЗДІЛ 2
ІНСТРУМЕНТИ ТА ТЕХНОЛОГІЇ ДЛЯ РЕАЛІЗАЦІЇ СИСТЕМИ

2.1 Інструмент для розпізнавання мовлення

У межах цього підпункту, детально буде розглянути одну з ключових бібліотек для розпізнавання мовлення – Vosk, її архітектуру, основні можливості та спосіб інтеграції з Java-проєктом.

Vosk – це open-source бібліотека ASR, побудована на базі Kaldi. Поширюється під ліцензією Apache 2.0, що дозволяє її використовувати в комерційних та дослідницьких цілях без додаткових обмежень. На відміну від багатьох сервісів, Vosk виконує розпізнавання локально, що забезпечує мінімальну затримку та незалежність від інтернет-з’єднання. Існує понад 20 готових мовних моделей різного розміру та точності: від легких (~40 МБ) до великих (>1 ГБ).

Vosk підтримує стрімінговий режим, текст поступає частинами ще до завершення вимови речення, що критично для застосунків із низькою латентністю [16].

Розглянемо особливості технології Vosk. (табл. 2.1).

Таблиця 2.1 Ключові особливості

Функція	Опис
Streaming API	Повертає часткові та остаточні транскрипції, дозволяє обробляти аудіопотік «на льоту».
Speaker Diarization	Можливість виділення кількох мовців у записі (через зовнішні плагіни).
Keyword Spotting	Підтримка пошуку вхідних ключових слів із визначенням таймінгів (timestamps).
Вбудований шумозаглушувач	Автоматичне зниження фонового шуму за допомогою spectral subtraction.
Низькі вимоги до ресурсів	Може працювати на Raspberry Pi, мобільних пристроях та вбудованих системах.
API для різних мов програмування	Окрім Java, є обгортки для Python, C#, JavaScript, Go та інших.

Для підключення до Java потрібно підключити залежності, які відповідають за Vosk, у файл конфігурації наданий момент це в pom.xml (Maven) [15].

```xml
<dependency>
  <groupId>com.alphacephei</groupId>
  <artifactId>vosk</artifactId>
  <version>0.3.48</version>
</dependency>
```

Рисунок 2.1 – Підключення залежностей

Далі потрібно завантажити мовну модель та інтегрувати її в програмний код.

```java
import org.vosk.Model;
import org.vosk.Recognizer;
// …
Model model = new Model("models/vosk-model-uk");
Recognizer recognizer = new Recognizer(model, 16000.0f);
```

Рисунок 2.2 – Інтеграція мовної моделі

Потім добавляємо обробку аудіопотоку.

```java
InputStream ais = /* потік PCM-аудіо, 16 kHz, 16 bit, mono */;
byte[] buffer = new byte[4096];
int nbytes;
while ((nbytes = ais.read(buffer)) >= 0) {
    if (recognizer.acceptWaveForm(buffer, nbytes)) {
        System.out.println(recognizer.getResult());
    } else {
        System.out.println(recognizer.getPartialResult());
    }
}
System.out.println(recognizer.getFinalResult());
```

Рисунок 2.3 – Обробка аудіопотоку

Хоча Vosk є основним вибором для ASR-модуля, у ролі альтернатив можна розглядати Kaldi (C++) гнучка, розширювана платформа, однак складніше у

налаштувані. Deepspeech (Mozilla), відкритий TensorFlow-проєкт з Python/C++ API, але з вищими вимогами до ресурсів. Commercial APIs (Google, Azure, IBM), висока точність і підтримка шумозаглушення, але потребують стабільного інтернет-з'єднання та мають витрати за запити.

Таким чином, Vosk поєднує в собі баланс між простотою інтеграції, можливостями офлайн-роботи та ресурсною економністю. Саме ці властивості роблять його оптимальним вибором для Java-реалізації голосового перекладача, де критичною є як точність розпізнавання, так і мінімальна затримка. Також, варто пам'ятати, що Vosk працює на певному аудіоформаті, отже потрібно бути обачним та дотримуватися цього.

2.2 Інструменти для перекладу тексту

У якості основного рушія машинного перекладу в цьому проєкті обрана система LiberTransalte – відкритий, самостійно розгортуваний сервіс із REST-API, що базується на сучасних нейромережевих моделях. Далі детальніше розглянемо архітектуру, можливості та приклади інтеграції.

LiberTranslate поширюється під ліцензією GNU Affero GPL v3, що дозволяє безкоштовно встановлювати й змінювати код. Можна розгорнути його локально або на сервері, гарантуючи конфіденційність та автономність без сторонніх хмарних сервісів. За основу взято передтреновані трансформери з OPUS-MT та Argos Translate. Підтримуються понад 50 мов. Простий HTTP-інтерфейс для запитів перекладу, що робить інтеграцію з будь-якою мовою програмування максимально зручною [19].

Розглянемо особливості LiberTranslate, (табл. 2.2).

Таблиця 2.2 Ключові особливості LiberTranslate

Функція	Опис
Самостійне розгортання	Можливість встановити сервер на локальному хості або в приватному дата-центрі.

Продовження таблиці 2.2 Ключові особливості LiberTranslate

Функція	Опис
Шифрування трафіку	Підтримка HTTPS для безпечної передачі запитів і відповідей.
Підтримка custom-моделей	Можна підмінити або донавчити власні моделі Marian/Argos для специфічного домену.
Паралельні запити	HTTP keep-alive та підтримка багатьох одночасних клієнтів.
Додаткові сервіси	Детектування мови (/detect), переклад файлів, статус сервера, статистика використання.

До архітектури рішень можемо віднести Frontend-шаблони, Node.js/Express або мікрофреймворк у Java/Python для маршрутизації й обробки клієнтських запитів. LibreTranslate-сервер, Docker-контейнер або локальна інсталяція Python-пакету. Моделі перекладу, Marian OPUS-MT, завантажені у вигляді .onnx або PyTorch-моделей (за потреби можна конвертувати для кращої продуктивності). Кешування результатів, Redis або in-memory кеш для прискорення часто повторюваних запитів.

До переваг можемо віднести повна автономія – жодних витрат за хмару й обмежень API; гнучкість у налаштуванні власних моделей і фільтрації контенту; підтримка робочих навантажень через кешування й маштабування по контейнерах.

До обмежень відносяться відносно вища затримка порівняно з великими хмарними провайдерами при слабкій інфраструктурі; необхідність власного моніторингу й оновлення моделей; менший вибір голосових моделей для доменів із вузькою спеціалізацією [18].

Таким чином, LiberTranslate є оптимальним рішенням для приватного хостингу мікросервіс перекладу. Завдяки простому REST-інтерфейсу та

підтримці сучасних трансформених моделей, він поєднує в собі зручність інтеграції, контроль над даними й достатній рівень якості перекладу.

2.3 Інструмент для синтезу мовлення

У цьому підпункті буде розглянуто механізм синтезу мовлення – RHVoice, його властивості, архітектуру та спосіб інтеграції в проєкт.

RHVoice – безкоштовна система TTS під ліцензією MIT, розроблена для багатомовного синтезу з відкритими голосовими движками. Підтримує низку єфропейських мов, зокрема українську, англійську, іспанську та інші.

RHVoice поєднує два основні компоненти, лінгвістичний фротенд – нормалізація тексту, розбиття на речення, токенізація, встановлення наголосів та інто-наційних пауз. Вокодер – HMM-базований генератор параметрів + LPC-вокодер для генерації звукових хвиль [20].

Розглянемо таблиця ключових можливостей RHVoice, (табл. 2.3).

Таблиця 2.3 Ключові можливості RHVoice

Функція	Опис
Локальний синтез	Повністю офлайн, не потребує інтернет-з’єднання, що підвищує безпеку даних і швидкість обробки.
Конфігуровані голоси	Можливість вибору між кількома вбудованими голосами або підключення власного голосового пакета.
Налаштування інтонації	Зміна висоти тону, швидкості мовлення та гучності через параметри командного рядка чи API.
Мінімальні залежності	Написана на C++, легко компілюється на Linux, Windows, embedded-платформах.
API та CLI	Підтримка викликів через бібліотеку (libRHVoice) та командний рядок (RHVoice-client).

Налаштування та оптимізація синтезу мовлення включає регулювання швидкості за допомогою прапорця -s у CLI (наприклад, -s 1.2 для збільшення темпу на 20%), зміну тону/висоти голосу через параметр -a, кешування результатів шляхом збереження WAV-файлів для типових фраз з метою уникнення повторного синтезу, а також реалізацію паралельної обробки через запуск кількох CLI-запитів у окремих потоках для одночасного виконання [21].

До переваг можемо віднести абсолютний офлайн, висока швидкість на локальному обладнанні; гнучкість у налаштуванні параметрів голосу без перекомпіляції; легкість впровадження в десктоп- та сервісні додатки.

До недоліків можемо віднести природність голосу нижча за сучасні нейронні TTS (FastSpeech 2 + HiFi-GAN); менший вибір емоційних або стилістичних варіантів інтонації; відносно грубі артефакти при швидкому темпі мовлення або нестандартному тексті.

З огляду на вищезазначене, RHVoice стає оптимальним вибором для озвучення тексту в системі голосового перекладача, де критичною є автономність та мінімальна затримка, а також можливість тонкої регулювання параметрів синтезу під потреби кінцевого користувача.

2.4 Вибір мови програмування

У процесі розробки системи голосового перекладу важливим кроком стало вибір мови програмування, яка дозволяє забезпечити ефективну інтеграцію основних компонентів системи – розпізнавання мовлення, машинного перекладу та синтезу голосу. Для реалізації проєкту була обрана Java, яка надає широкий спектр інструментів для створення багатоплатформних застосунків, зокрема з підтримкою графічного інтерфейсу та взаємодією з зовнішніми бібліотеками і сервісами.

Java-програми можуть виконуватись на будь-якій операційній системі, де встановлена JVM (Java Virtual Machine): Windows, Linux, macOS. Це дає

можливість запускати програму як на сервері, так і на клієнтському ПК без значних змін у коді.

Оскільки система голосового перекладу має виконувати кілька задач паралельно (наприклад, прослуховування мови, обробку тексту, переклад і синтез), Java пропонує зручний та безпечний механізм багатопотоковості через Thread, ExecutorService, CompletableFuture тощо [22].

Java має велику кількість бібліотек і фреймворків, що дозволяють працювати з HTTP-запитами, файлами, звуком, графічним інтерфейсом, потоками тощо. Наприклад: Unirest або HttpClient – для роботи з REST API (LibreTranslate), javax.sound.sampled – для обробки аудіо, Swing або JavaFX – для створення графічного інтерфейсу користувача, ProcessBuilder – для виклику зовнішніх програм (Vosk, RHVoice) [28].

Проєкти на Java легко розширюються, додаючи нові модулі, API, UI-компоненти. У разі потреби систему можна адаптувати до мобільної версії (через Android) або веб-інтерфейсу (через Spring Boot + API) [23].

Розглянемо взаємодію з основними компонентами, (табл. 2.4).

Таблиця 2.4 Взаємодія з основними компонентами

Компонент	Спосіб взаємодії в Java
Vosk (розпізнавання)	Через Java-біндінг vosk-api.jar, який викликає моделі C++
LibreTranslate (переклад)	Через HTTP-запити до локального або віддаленого REST API
RHVoice (синтез)	Через CLI або виклики JNI / JNA до нативної бібліотеки

Це дозволяє об'єднати всі етапи в один послідовний або паралельний процес у межах одного застосунку, не потребуючи складних зв'язків між мовами програмування [27].

Також розглянемо і альтернативні варіанти вибору мови програмування, їх переваги та недоліки, (табл. 2.5) [35].

Таблиця 2.5 Альтернативні варіанти

Мова	Переваги	Недоліки
Python	Простота, підтримка ML, багато бібліотек	Менш ефективна для великих GUI/десктоп-проєктів
C++	Висока швидкість	Складність інтеграції з API, GUI, багатопоточність
JavaScript	Веб-інтерфейси, кросплатформені фреймворки	Не зручно для офлайн-систем розпізнавання/синтезу
C#	Зручна розробка GUI (WinForms/WPF)	Обмежена кросплатформеність, особливо в Linux

Отже, вибір Java як основної мови для реалізації системи голосового перекладача обґрунтований її універсальністю, потужною підтримкою багатопоточності, простотою інтеграції з нативними бібліотеками та API, а також широкими можливостями для розробки як локальних, так і кросплатформених застосунків. Java забезпечує баланс між продуктивністю, стабільністю та зручністю для розробника, що особливо важливо при роботі з мультимедійними компонентами в режимі реального часу [26].

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Створення клієнт-серверної частини голосового чату

У цьому підпункті детально розглядається реалізація базової клієнт-серверної архітектури для двосторонньої передачі аудіо у реальному часі з використанням Java Sockets та Java Sound API. Голосовий чат складається з двох програм, VoiceChatServer – сервер, який встановлює з'єднання з двома клієнтами й ретранслює аудіопотік між ними та VoiceChatClient – клієнт, який захоплює звук з мікрофона, відправляє його на сервер та відтворює вхідний потік звуку з сервера [36].

Архітектура системи передбачає ініціалізацію сервера, який слухає порт (55555) і очікує на підключення двох клієнтів; після встановлення з'єднань запускаються два потоки для релейної передачі аудіо: кожен читає дані з одного клієнта й відразу пересилає іншому. Клієнти підключаються до сервера за вказаною IP-адресою, ініціалізуючи мікрофон і динаміки в однаковому аудіоформаті. Передача аудіо здійснюється двобічно: клієнт читає PCM-дані з мікрофона (через TargetDataLine) і надсилає їх у потоці на Socket.getOutputStream(), сервер негайно пересилає ці байти другому клієнту, який відтворює їх через динаміки. Зворотний потік даних відбувається аналогічно. Сервер виконує лише релейну функцію, без обробки або буферизації, що забезпечує мінімальну затримку [31,32].

Потрібно підготувати правильний формат аудіо, так як Vosk приймає не усі формати:

- ✓ Формат: WAV (Linear PCM)
- ✓ Кількість каналів: 1 (моно)
- ✓ Частота дискретизації: 16000 Гц (16 кГц) — оптимально
- ✓ (можна й інші частоти, як 8000 або 44100 Гц, але модель може працювати гірше або повільніше)

✓ Бітова глибина: 16 біт

16-бітна моно-запис зменшує обсяг передаваних даних та навантаження на мережу.

Реалізація серверу починається зі створення ServerSocket, після чого запускаються relay-потоки для кожного з клієнтів, (рис. 3.1).

```
ServerSocket serverSocket = new ServerSocket(port);
Socket clientA = serverSocket.accept();
Socket clientB = serverSocket.accept();

relay(clientA.getInputStream(), clientB.getOutputStream(), "A→B");
relay(clientB.getInputStream(), clientA.getOutputStream(), "B→A");
```

Рисунок 3.1– Початок реалізації серверу

Кожен потік читає з вихідного потоку InputStream in і пише у відповідний OutputStream out, використовуючи буфер 4096 байт

Ось, які переваги такого підходу, мінімальна логіка на сервері, що спрощує масштабування та мала латентність завдяки негайній пересилці байтів.

Після реалізація серверу, потрібно створити клієнта, який буде під'єднуватися до серверу. Спочатку створюється змінна socket, яка буде приймати два параметри для підключення. Потім ініціалізуємо мікрофон та динаміки, для передачі та отримання аудіо. Також, потрібно налаштувати, щоб у фоновому потоці передавався звук з мікрофону. Вкінці, створюється метод для отримання та відтворення аудіо, яке надійшло від одного з клієнтів, (див. Додаток А «Реалізація Клієнта») [29].

Також, обробляємо помилки, щоб логувати помилки без блокування основного потоку програми.

Реалізована клієнт-серверна архітектура забезпечує простий і водночас ефективний канал для двосторонньої аудіопередачі в режимі реального часу, що допоможе в тестуванні програми-перекладача. Використання стандартних Java Socket та Java Sound API робить рішення кросплатформеним та легким для інтеграції з подальшими модулями розпізнавання, перекладу та синтезу мовлення [30].

3.2 Використання Vosk для розпізнавання мовлення

У цьому підпункті описано, як на базі Vosk-моделі реалізовано сервіс розпізнавання мовлення – SpeechService та як завдяки абстракції AudioSource забезпечується гнучкість у підключенні різних джерел аудіо (мікрофон, мережевий потік тощо).

Клас `SpeechService` відповідає за архітектуру сервісу розпізнавання мовлення, інкапсулюючи взаємодію з Vosk API. Він забезпечує завантаження мовної моделі, ініціалізацію об'єкта `Recognizer` та обробку PCM-буферів аудіо. Основні методи включають `start(AudioSource source)`, який приймає будь-яке аудіо-джерело, створює `Model` та `Recognizer`; `recognize()`, який зчитує наступний фрейм звуку з `AudioSource`, подає його до `Recognizer` і повертає текст, якщо розпізнано завершений сегмент; та `stop()`, що зупиняє аудіо-джерело і звільняє всі задіяні ресурси [33,34].

Інтерфейс AudioSource абстрагує спосіб отримання PCM-даних, (рис. 3.2).

```
public interface AudioSource {
    int read(byte[] buffer);
    void stop();
}
```

Рисунок 3.2 – Інтерфейс AudioSource

Це дозволяє підміняти різні реалізації – мікрофон, файл, мережевий потік.

Клас VoiceChatAudioSource бере на вхід InputStream (від сокета клієнта) і реалізує читання у буфер до завершення потоку.

Створюється метод start(), що буде ініціалізувати надходжений аудіопотік, який передається за допомогою розробленого інтерфейсу, (рис. 3.3).

```
public void start(AudioSource source) {
    this.audioSource = source;
    try {
        Model model = new Model("model");
        recognizer = new Recognizer(model, 16000);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Рисунок 3.3 – Метод start()

Model – завантажує файли акустичної та мовної моделі із теки model/.

Recognizer – приймає модель і частоту дискретизації, готується до обробки PCM-фреймів.

Далі, створюється метод recognize(), що буде обробляти та розпізнавати наджоджене аудіо, (рис.3.4).

```
public String recognize() {
    byte[] buf = new byte[4096];
    try {
        int len = audioSource.read(buf);
        if (len > 0 && recognizer.acceptWaveForm(buf, len)) {
            return new org.json.JSONObject(recognizer.getResult())
                    .optString("text", "");
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    return "";
}
```

Рисунок 3.4 – Метод recognize()

Описаний метод відповідає за обробку одного фрейма аудіо з метою розпізнавання мовлення. Він виконує такі кроки: зчитує до 4096 байт PCM-даних із `audioSource`; передає їх у `Recognizer.acceptWaveForm(...)`, який повертає `true`, якщо завершено сегмент мовлення і можна отримати фінальний результат; у такому випадку метод `getResult()` повертає JSON-рядок із розпізнаним текстом, деталями розбивки на слова (`result`) та, за потреби, інформацією про мовця (`spk`). Для отримання тексту з JSON використовується бібліотека

`org.json`, яка дозволяє витягнути поле `"text"` як фінальний розпізнаний результат.

Останній метод – це stop(), який закриває вхідний стрім і сигналізує сервісу про припинення обробки.

Завдяки SpeechService і інтерфейсу AudioSource, інтеграція Vosk у систему реалізована максимально гнучко й модульно: легко змінити джерело аудіо, конфігурувати параметри моделі або додати обробку «проміжних» результатів. Така структура спрощує тестування, розширення функціоналу та подальшу оптимізацію розпізнавання [28].

3.3 Використання LiberTransalte для перекладу тексту

У цьому підпункті описано реалізацію класу TranslationService, який взаємодіє з локально розгорнутим сервером LibreTranslate через REST API для перекладу тексту з англійської на українську.

TranslationService — це модуль машинного перекладу, який відповідає за прийом вхідного тексту англійською мовою та повернення його українського перекладу, займаючи центральне місце у відповідному етапі обробки голосового потоку. Взаємодія з сервером перекладу здійснюється через HTTP POST-запит з JSON-повідомленням, що забезпечує зручну та стандартизовану передачу даних. Основні переваги такої архітектури — повна ізоляція від внутрішньої реалізації стороннього перекладача, що спрощує заміну сервісу або розширення системи для підтримки нових мов без змін у логіці клієнтської частини.

Спочатку потрібно ініціалізувати з’єднання з локальним сервером перекладача, який запущений за допомогою Docker(Desktop), (рис. 3.5).

```
URL url = new URL("http://localhost:5000/translate");
HttpURLConnection conn = (HttpURLConnection) url.openConnection();
conn.setRequestMethod("POST");
conn.setRequestProperty("Content-Type", "application/json; utf-8");
conn.setDoOutput(true);
```

Рисунок 3.5 – Ініціалізація HTTP-з’єднання

✓ URL, за замовчуванням localhost:5000/translate, якщо сервер LibreTranslate запущено в Docker.

✓ Метод POST, для передачі тіла запиту з параметрами перекладу.

✓ Заголовки, Content-Type: application/json: utf-8 – повідомляє сервер про формат і кодування запиту.

✓ conn.setDoOutput(true), необхідно для запису тіла запиту.

Далі потрібно прописати тіло формування та відправки запиту, (рис. 3.6).

```
JSONObject payload = new JSONObject()
        .put("q", text)
        .put("source", "en")
        .put("target", "uk")
        .put("format", "text");
try (OutputStream os = conn.getOutputStream()) {
    os.write(payload.toString().getBytes(StandardCharsets.UTF_8));
```

Рисунок 3.6 – Формування та відправка запиту

Параметри запиту:

✓ q – текст для перекладу.

✓ source – код вихідної мови (en).

✓ target – код цільової мови (uk).

✓ format – формат тексту (text або html).

Запис у потік: байти JSON записуються в conn.getOutputStream [17].

Далі прописуємо, щоб програма інтерпретувала повідомлення, яке відправив сервер, це має бути або перекладений текст або порожня множина, де InputStream читає відповідь сервера, StringBuilder акумулює всі рядки відповіді, (див. Додаток А «Реалізація MTService») [16].

Можливі доопрацювання модуля `TranslationService` включають кілька напрямів підвищення надійності та ефективності: налаштування таймаутів для запобігання зависанню при недоступності сервера; реалізацію механізму повторних спроб із обмеженням кількості і паузами між запитами; підтримку паралельної обробки перекладів через `ExecutorService`; розширення

функціональності методу `translate(...)` за рахунок параметрів `source` і `target` для динамічного вибору мовної пари; а також кешування нещодавно перекладених фраз у структурі типу `Map<String, String>` для зменшення кількості запитів до зовнішнього сервера й підвищення продуктивності.

Клас TranslationService забезпечує простий та зрозумілий інтерфейс для машинного перекладу через LibreTranslate. Завдяки добре спроектованому HTTP-клієнту й обробці JSON, сервіс легко інтегрується в загальну архітектуру голосового перекладача, гарантуючи автономну роботу без звернень до зовнішніх хмарних API.

3.4 Використання TTSService для озвучення перекладеного тексту

У цьому підпункті розглянемо клас TtsService, що відповідає за синтез мовлення в системі, та опишемо, як із його допомогою відбувається озвучення тексту за допомогою вбудованого Windows Speech API через PowerShell.

Метою класу є забезпечення простого інтерфейсу speak(String, text) для перетворення рядка в мовлення без прямої роботи з низькорівневими TTS-бібліотеками [25].

Було обрано System.Speech.Synthesis – стандартна .NET-бібліотека для TTS, викликана через PowerShell, що дає можливість запускати синтез без залучення зовнішніх компонентів.

До переваг можемо віднести відсутність потреби встановлювати додаткове ПЗ – достатньо наявність Windows із .NET Framework; Гнучка конфігурація голосу, швидкості та гнучкості шляхом передачі параметрів у скрипт PowerShell[24].

TTSService складається з одного методу, який відповідає за озвучення отриманого перекладу. Метод запускає PowerShell.exe, потім через Add-Type -AssemblyName System.Speech, підключається збірка .NET System.Speech, створюється екземпляр SpeechSynthesizer, за допомогою SelectVoice(VOICE) вибирається голос, який був завантажений додатково. Також встановлюються

параметри Rate та Volume на дефолті значення, після чого викликається $synth.Speak(…) для озвучення в консолі, (див. Додаток Б «Реалізація TTSService).

Тепер розглянемо переваги та обмеження такого підходу, (табл. 3.1).

Таблиця 3.1 Переваги та обмеження підходу

Переваги	Обмеження
Використання штатних .NET-компонентів	Працює лише на Windows із встановленим PowerShell
Не потребує інсталяції додаткового ПЗ	Висока латентність через запуск зовнішнього процесу
Гнучке налаштування голосу, швидкості, гучності	Обмежений набір голосів, залежні від того, що встановлено в ОС

Оптимізація озвучення може включати кілька покращень, такі як асинхронний запуск озвучення в окремому потоці замість блокуючого waitFor(), щоб не затримувати роботу UI чи інших модулів; винесення виклику Add-Type у попередньо ініціалізований скрипт для уникнення повторного завантаження збірки при кожному виклику; використання альтернативних голосових движків, таких як SAPI5, із зазначенням конкретного голосу в константі VOICE; а також кешування озвучених WAV-файлів за ключем тексту, що дозволяє при повторному зверненні відтворювати вже згенероване аудіо через Java Sound API без повторного синтезу, що значно економить ресурси.

Отже, клас TtsService надає простий механізм озвучення перекладеного тексту, спираючись на вбудовані можливості Windows. Завдяки PowerShell-скрипту вдається швидко інтегрувати TTS без зовнішніх залежностей, хоча підхід обмежений платформою та потребує оптимізації латентності для більш плавного користувацького досвіду.

3.5 Тестування програмного забезпечення

Спочатку потрібно запустити сервер голосового чату та підключити клієнтів (рис. 3.7).

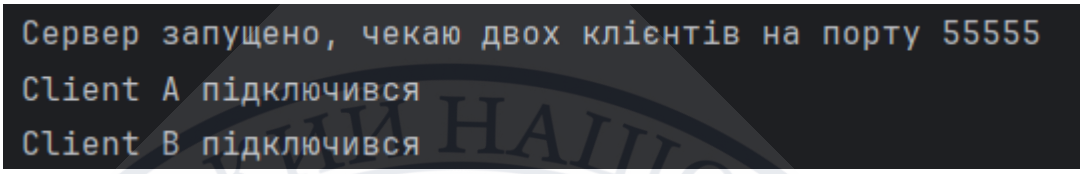

Рисунок 3.7 – Запуск серверу та підключення клієнтів

Після чого запускаємо програму-перекладач (рис. 3.8).

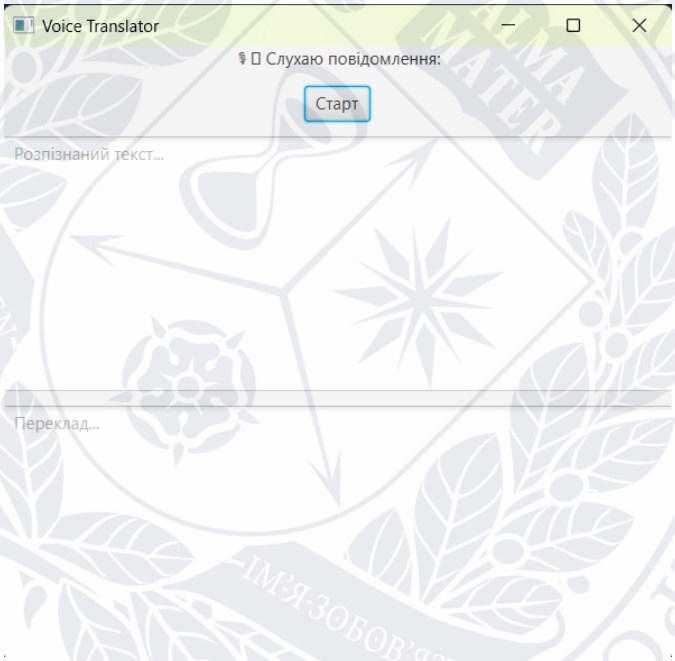

Рисунок 3.8 – Графічний інтерфейс перекладача

Поки, що інтерфейс виглядить так, його в майбутньому можна допрацьовувати, але основні функції програма виконує. Інтерфейс складається з кнопки «Старт», яка під’єднує програму до серверу, та двох полів, в першому показується розпізнаний текст, в другому – перекладений текст [37, 38].

Тепер протестуємо розпізнавання та переклад тексту на пару англійських фраз, (рис. 3.9).

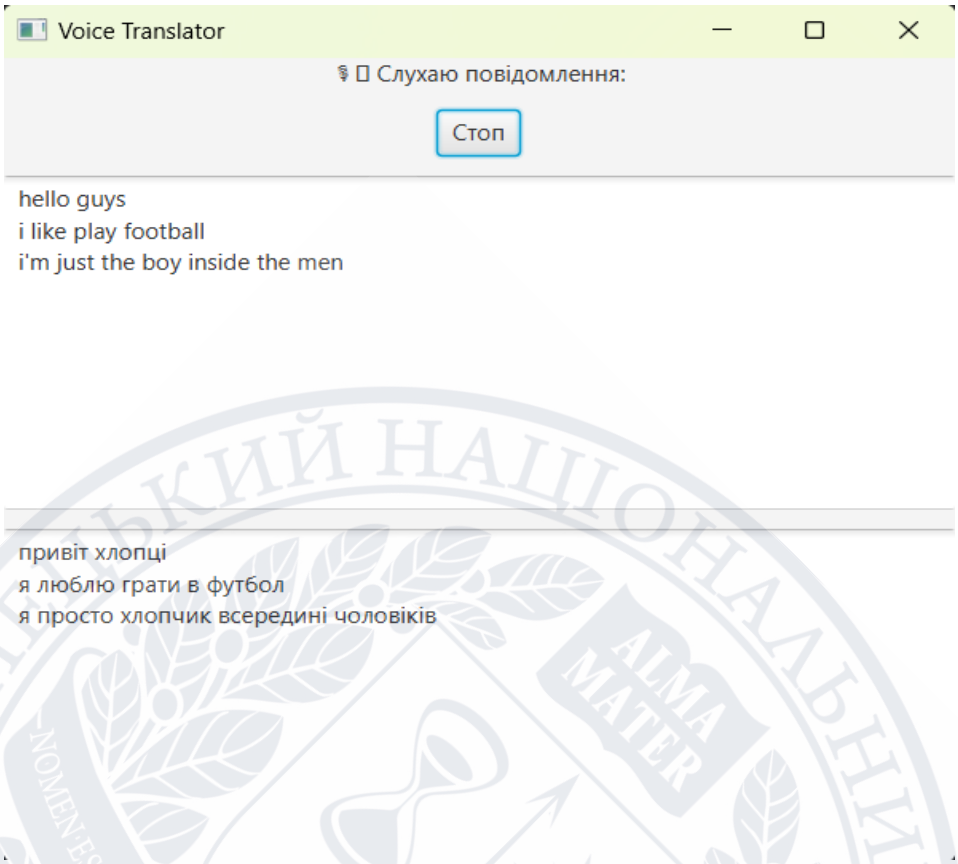

Рисунок 3.9 – Приклад роботи перекладача

Під час тестування програми, було помітно деякі неточності в перекладі, наприклад слово «Hello», що означає «Привіт», програма перекладає по іншому, (рис. 3.10).

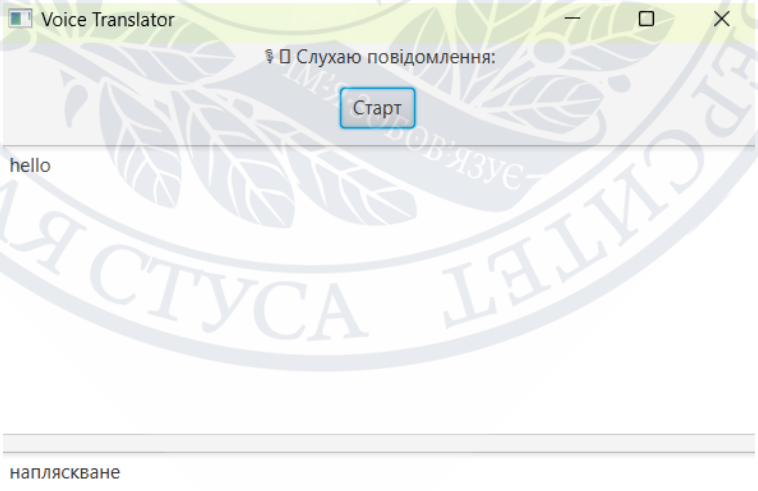

Рисунок 3.10 – Неточність перекладу слова «Hello»

Хоча на Рисунку 3.9, видно що в зв’язці двох слів «Hello guys», переклад є правильним.

Також, якщо взяти речення «What is your name?», що означає «Як тебе звати?», то переклад також неточний, (рис. 3.11).

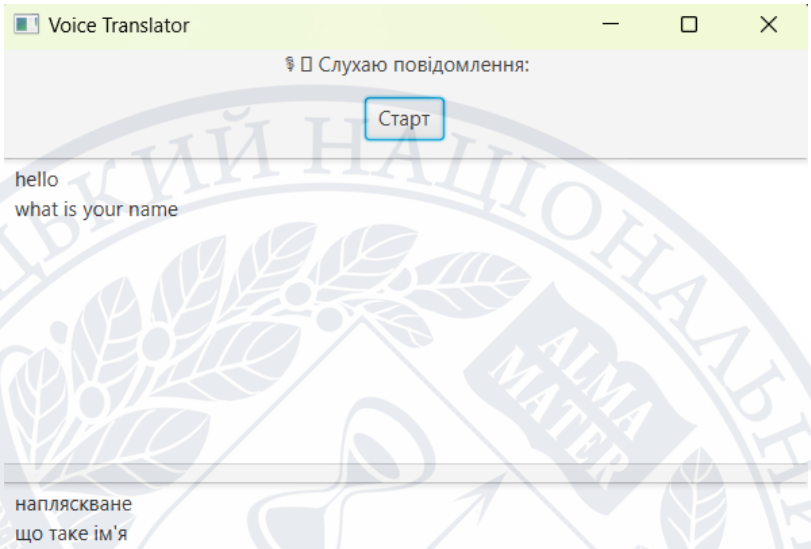

Рисунок 3.5 – Неточності перекладу речення

Також, проведемо тест на швидкість розпізнавання, перекладу та озвучення. Для цього створимо тестовий код, який буде заміряти час по кожному ключовому елементі і показувати загальний час, (табл. 3.2).

Таблиця 3.2 Тестування на швидкість роботи ключових елементів

№	Розпізнано фразу	Розпізнавання (мс)	Переклад (мс)	Загальний час (мс)
1	hello guys	8	110	2707
2	where are you from	0	73	1670

Продовження таблиці 3.2 Тестування на швидкість роботи ключових елементів

№	Розпізнано фразу	Розпізнавання (мс)	Переклад (мс)	Загальний час (мс)
3	can you help me please	0	74	3098
4	i like play football	0	85	2448

Отже, з одного боку, програма працює не погано, вона добре розпізнає мовлення, яке надходить від іншого користувача, перекладає та озвучує переклад, але є певні нюанси, такі як: деякі неточності в перекладі, доволі велика затримка озвучення, так як програма довго перекладає отриманий текст, що може сказатися на не зручності.

Цю програму можна вдосконалювати, так як зараз це просто початок. Потрібно вибрати інший сервіс для перекладання, який точніше дає переклад, наприклад Google Translate API, що набагато кращий, ніж LiberTranslate, але плюси LiberTranslate в тому, що він безкоштовний ніж його аналоги. Також потрібно зменшити затримку роботи програми, щоб вона була більш швидкою при обробці запиту. Ще можна добавити базу даних, яка буде вміщувати в собі моделі мов для розпізнавання, мовні пакети озвучення на різних мовах (чоловічі та жіночі голоси), також можна щоб фрази, які були перекладені, зберігалися в базі даних, щоб при запиті на таку саму фразу, не йшов час на обробку, а відразу брався аудіозапис з БД. Також, потрібно добавити можливість вибору мови, з якої має бути переклад і на яку перекладати, ще можна добавити список доступних голосів синтезу мовлення повідомлення.

ВИСНОВКИ

Проаналізовано методи розпізнавання мовлення (HMM-GMM, CNN/RNN/Transformer, end-to-end), машинного перекладу (RBMT, SMT, NMT) та синтезу мовлення (конкатенативний, параметричний, нейронні моделі). Визначено їхні сильні та слабкі сторони щодо latency, точності й ресурсних вимог.

Обґрунтовано використання Vosk для офлайн-ASR (низька затримка, простота інтеграції), LibreTranslate для машинного перекладу через REST API (гнучкість самостійного хостингу, сучасні трансформерні моделі) і RHVoice (або Windows System.Speech) для TTS (офлайн-режим, мінімальні залежності).

Створено модульну клієнт-серверну структуру: VoiceChatClient ↔ VoiceChatServer для передачі PCM-потоку, ASR-модуль (SpeechService + AudioSource), MT-модуль (TranslationService + LibreTranslate) та TTS-модуль (TtsService). Такий поділ забезпечує легку заміну або оновлення окремих компонентів.

Реалізовано:

- ✓ клієнт-серверну частину для двобічної аудіорелейної передачі з Java Sound API;
- ✓ сервіс розпізнавання мови на базі Vosk із абстракцією AudioSource для різних джерел звуку;
- ✓ модуль перекладу через HTTP-клієнт до LibreTranslate;
- ✓ клас озвучення перекладу PowerShell-скриптом (System.Speech) та альтернативно RHVoice.

Проведено тести програми-перекладача на точність розпізнавання та перклад. Після цих тестів, можна сказати, що програма добре розпізнає мовлення, навіть якщо на фоні є інші звуки, але переклад розпізнаного тексту, в деяких моментах є не точним. Також є велика затримка між отриманим потоком та озвученим перекладом, що потрібно ще допрацьовувати.

Розроблений прототип може застосовуватися в освітніх платформах, ігрових голосових чатах та корпоративних аудіоконференціях. Для зниження latency доцільно впровадити квантування моделей, багатопотокову обробку та передінфраструктурні прискорювачі (GPU/DSP).

Отже, виконані завдання дозволили створити інтегровану систему голосового перекладу, яка поєднує автономність, модульність і достатню якість сервісів ASR, MT і TTS для практичного використання.

СПИСОК ВИКОРИСТАНИХ ПОСИЛАНЬ

1. Анадея А. Який сервіс автоматичного перекладу найточніший. *Linguise*. URL: https://surl.li/ensibn (дата звернення 17.05.2025).
2. Цвіркун Ю. Види машинного перекладу. *Профпереклад*. URL: https://profpereklad.ua/vidi-mashinnogo-perekladu/ (дата звернення 17.05.2025).
3. Фіалковська Д. Повний посібник з розпізнавання мовлення. *Transkriptor*. URL: https://transkriptor.com/uk/розпізнавання-мовлення/ (дата звернення 17.05.2025).
4. Що таке ASR (автоматичне розпізнавання мовлення): усе, що потрібно знати початківцю (у 2024 році). *Shaip*. URL: https://surli.cc/vttoav (дата звернення 17.05.2025).
5. Як працює синтез мовлення?. *Speaktor*. URL: https://speaktor.com/uk/мовлення-синтез/ (дата звернення: 17.05.2025).
6. Що таке синтез мовлення (TTS) і як він працює з ШІ?. *EITCA*. URL: https://surl.li/trdrsj (дата звернення 18.05.2025).
7. Jurafsky D., Martin J. H. Speech and Language Processing : навч. Посіб. 2025. Т. 3 : Speech and Language Processing. 599 с. URL: https://web.stanford.edu/~jurafsky/slp3/ed3book.pdf (дата звернення 18.05.2025).
8. Google. Cloud Translation documentation. *Google Cloud*. URL: https://cloud.google.com/translate/docs (date of access 17.05.2025).
9. Këpuska V. B., Bohouta G. Comparing Speech Recognition Systems (Microsoft API, Google API and CMU Sphinx). : навч. Посіб. Indianapolis, IN, USA : ResearchGate, 2017. 6 с. URL: https://doi.org/10.1109/FIE.2017.8190668 (дата звернення 17.05.2025).
10. Wav2vec 2.0: a framework for self-supervised learning of speech representations / Baevski A. Та ін. 2020. С. 5–8. URL: https://doi.org/10.48550/arXiv.2006.11477 (дата звернення 17.05.2025).
11. Bahdanau D., Cho K., Bengio Y. Neural machine translation by jointly learning to align and translate. *ICLR 2015*, м. San Diego, CA, 7 трав. 2015 р. 2015. С. 4–11. URL: https://doi.org/10.48550/arXiv.1409.0473 (дата звернення 17.05.2025).
12. Neubig G. Neural machine translation and sequence-to-sequence models: a tutorial. Pittsburgh, 2017. 65 с. URL: https://doi.org/10.48550/arXiv.1703.01619 (дата звернення 18.05.2025).
13. Natural TTS synthesis by conditioning wavenet on mel spectrogram predictions / Ron J. Weiss та ін. 18 квіт. 2018 р. California, 2018. С. 1–5. URL: https://doi.org/10.48550/arXiv.1712.05884 (дата звернення 18.05.2025).
14. FastSpeech: fast, robust and controllable text to speech / Yi Ren та ін. м. Vancouver, 8 груд. 2019 р. Zhejiang, 2019. С. 4–10. URL: https://doi.org/10.48550/arXiv.1905.09263 (дата звернення 18.05.2025).

15.Albion N. GitHub – alphacep/vosk-api: Offline speech recognition API for Android, iOS, Raspberry Pi and servers with Python, Java, C# and Node. *GitHub*. URL: https://github.com/alphacep/vosk-api (дата звернення 19.05.2025).

16.VOSK offline speech recognition API. *Alpacephei*. URL: https://alphacephei.com/vosk/ (дата звернення 19.05.2025).

17.LibreTranslate – free and open source machine translation API. *LibreTranslate*. URL: https://libretranslate.com/ (дата звернення 19.05.2025).

18.Manuela S. GitHub – libretranslate/libretranslate: free and open source machine translation API. Self-hosted, offline capable and easy to setup. *GitHub*. URL: http://github.com/LibreTranslate/LibreTranslate (дата звернення: 05.06.2025).

19.LibreTranslate проти Google Translate: детальне порівняння характеристик. *MachineTranslation.com*. URL: https://surl.li/vklzhn (дата звернення 19.05.2025).

20.RHVoice. *RHVoice.org*. URL: https://rhvoice.org/ (дата звернення 19.05.2025).

21.Zvonimir Stanečić. GitHub – RHVoice/RHVoice: a free and open source speech synthesizer for English and other languages. *GitHub*. URL: https://github.com/RHVoice/RHVoice (дата звернення 19.05.2025).

22.Oracle. The Java™ tutorials. *Oracle*. URL: https://docs.oracle.com/javase/tutorial/ (дата звернення 19.05.2025).

23.Oracle. Java Sound Technology. *Oracle*. URL: https://surli.cc/qobrsu (дата звернення 19.05.2025).

24.Microsoft. Microsoft Speech Platform – Runtime Languages. *Microsoft*. URL: https://www.microsoft.com/en-us/download/details.aspx?id=27224 (дата звернення 19.05.2025).

25.Microsoft. Microsoft Speech API (SAPI) 5.4. *Microsoft Learn: Build skills that open doors in your career*. URL: https://surl.li/lghern (дата звернення 19.05.2025).

26.Java. *Oracle*. URL: https://www.java.com/ (дата звернення 19.05.2025).

27. Java: матеріали для самопідготовки. *EPAM | Campus*. URL: https://campus.epam.ua/ua/blog/280 (дата звернення 19.05.2025).

28. The Java™ Tutorials. *Oracle*. URL: https://docs.oracle.com/javase/tutorial/ (дата звернення 19.05.2025).

29. Бібліотека тьюторіалів. *MKYoung*. URL: https://mkyong.com/ (дата звернення 19.05.2025).

30.Martin Fowler. A website on building software effectively. *martinfowler.com*. URL: https://martinfowler.com/ (дата звернення 19.05.2025).

31.Клієнт-серверна архітектура. *JavaRush*. URL: https://surl.li/jezbao (дата звернення 19.05.2025).

32.Zmerzlyi I. Клієнт-серверна архітектура та ролі серверів. *Medium*. URL: https://surl.lu/wjzmwz (дата звернення 19.05.2025).

33. Scott Selikoff, Jeanne Boyarsky. OCP Oracle Certified Professional Java SE 17 Developer Study Guide. Sybex, 2022. 1056 с.

34.Java Sound API. *Oracle*. URL: https://surl.li/nrcsrt (дата звернення 19.05.2025).

35.B. Andrus. Python проти Java: Яку мову програмування обрати?. *DreamHost Blog*. URL: https://surl.lu/rsnttx (дата звернення 19.05.2025).

36.JetBrains. IntelliJ IDEA – the IDE for Pro Java and Kotlin Development. *JetBrains*. URL: https://www.jetbrains.com/idea/ (дата звернення 19.05.2025).

37.JavaFX. *JavaFX*. URL: https://openjfx.io/ (дата звернення 19.05.2025).

38.Констянтин. Введення в Java FX. *JavaRush*. URL: https://surl.li/dcofsu (дата звернення 19.05.2025).

39.Олена. Методології розробки програмного забезпечення. *Wezoom*. URL: https://surl.lu/sqneti (дата звернення 19.05.2025).

40.Стадії розробки ПЗ. *QALight*. URL: https://qalight.ua/baza-znaniy/stadiyi-tsiklu-rozrobki-pz/ (дата звернення 19.05.2025).

ДОДАТКИ

ДОДАТОК А

Реалізація MTService

```
import org.json.JSONObject;

import java.io.*;
import java.net.HttpURLConnection;
import java.net.URL;
import java.nio.charset.StandardCharsets;

public class TranslationService {
    public String translate(String text) {
        try {
            URL url = new
URL("http://localhost:5000/translate");
            HttpURLConnection conn = (HttpURLConnection)
url.openConnection();
            conn.setRequestMethod("POST");
            conn.setRequestProperty("Content-Type",
"application/json; utf-8");
            conn.setDoOutput(true);

            JSONObject payload = new JSONObject()
                    .put("q", text)
                    .put("source", "en")
                    .put("target", "uk")
                    .put("format", "text");
            try (OutputStream os = conn.getOutputStream()) {

os.write(payload.toString().getBytes(StandardCharsets.UTF_8));
            }

            try (BufferedReader br = new BufferedReader(
                    new InputStreamReader(conn.getInputStream(),
StandardCharsets.UTF_8))) {
                StringBuilder sb = new StringBuilder();
                String line;
                while ((line = br.readLine()) != null)
sb.append(line);
                return new
JSONObject(sb.toString()).optString("translatedText", "");
            }
        } catch (Exception e) {
            return "❌ Помилка перекладу: " + e.getMessage();
        }
    }
}
```

ДОДАТОК Б

Реалізація TTSService

```
package org.example.service;

public class TtsService {
    private static final String VOICE = "Natalia";

    public void speak(String text) {
        try {
            String escaped = text.replace("'", "''");
            String ps = String.join(" ",
                    "powershell.exe",
                    "-NoProfile", "-ExecutionPolicy", "Bypass",
                    "-Command",
                    "\"Add-Type -AssemblyName System.Speech; " +
                            "$synth = New-Object
System.Speech.Synthesis.SpeechSynthesizer; " +
                            "$synth.SelectVoice('" + VOICE +
"'); " +
                            "$synth.Rate = 0; $synth.Volume =
100; " +
                            "$synth.Speak('" + escaped + "');\""
            );
            ProcessBuilder pb = new ProcessBuilder("cmd.exe",
"/c", ps);
            pb.redirectErrorStream(true);
            Process p = pb.start();
            p.waitFor();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

ДОДАТОК В

Реалізація SpeechService

```
package org.example.service;

import org.vosk.Model;
import org.vosk.Recognizer;

public class SpeechService {
    private Recognizer recognizer;
    private AudioSource audioSource;

    public void start(AudioSource source) {
        this.audioSource = source;
        try {
            Model model = new Model("model");
            recognizer = new Recognizer(model, 16000);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public String recognize() {
        byte[] buf = new byte[4096];
        try {
            int len = audioSource.read(buf);
            if (len > 0 && recognizer.acceptWaveForm(buf, len))
{
                return new
org.json.JSONObject(recognizer.getResult())
                        .optString("text", "");
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return "";
    }

    public void stop() {
        if (audioSource != null) {
            audioSource.stop();
        }
    }
}
```

ДОДАТОК Г

Реалізація VoiceChatServer

```
package org.example.voicechat;

import java.io.InputStream;
import java.io.OutputStream;
import java.net.ServerSocket;
import java.net.Socket;

public class VoiceChatServer {
    public static void main(String[] args) throws Exception {
        int port = 55555; // порт для голосового чату
        System.out.println("Сервер запущено, чекаю двох клієнтів
на порту " + port);

        try (ServerSocket serverSocket = new ServerSocket(port))
{
            // Приймаємо дві коннекції
            Socket clientA = serverSocket.accept();
            System.out.println("Client A підключився");
            Socket clientB = serverSocket.accept();
            System.out.println("Client B підключився");

            // Потоки a ↔ b
            relay(clientA.getInputStream(),
clientB.getOutputStream(), "A→B");
            relay(clientB.getInputStream(),
clientA.getOutputStream(), "B→A");
        }
    }

    private static void relay(InputStream in, OutputStream out,
String tag) {
        new Thread(() -> {
            byte[] buf = new byte[4096];
            try {
                int len;
                while ((len = in.read(buf)) != -1) {
                    out.write(buf, 0, len);
                }
            } catch (Exception e) {
                System.err.println("[" + tag + "] Relay error: "
+ e.getMessage());
            }
        }).start();
    }
}
```

ДОДАТОК Д

Реалізація VoiceChatClient

```
package org.example;

import javax.sound.sampled.*;
import java.io.*;
import java.net.*;

public class VoiceChatClient {
    private static final String SERVER_ADDRESS = "localhost";
    private static final int SERVER_PORT = 50005;

    public static void main(String[] args) throws Exception {
        Socket socket = new Socket(SERVER_ADDRESS, SERVER_PORT);
        System.out.println("Підключено до сервера");

        AudioFormat format = new AudioFormat(44100.0f, 16, 1,
true, false);
        TargetDataLine microphone =
AudioSystem.getTargetDataLine(format);
        microphone.open(format);
        microphone.start();

        SourceDataLine speakers =
AudioSystem.getSourceDataLine(format);
        speakers.open(format);
        speakers.start();

        // Передача звуку серверу
        new Thread(() -> {
            try (OutputStream out = socket.getOutputStream()) {
                byte[] buffer = new byte[1024];
                while (true) {
                    int count = microphone.read(buffer, 0,
buffer.length);
                    out.write(buffer, 0, count);
                    out.flush();
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }).start();

        // Відтворення звуку від іншого клієнта
        try (InputStream in = socket.getInputStream()) {
            byte[] buffer = new byte[1024];
            int count;
            while ((count = in.read(buffer)) > 0) {
                speakers.write(buffer, 0, count);
            }
```

ДОДАТОК Е

Реалізація MainController

```
package org.example.controller;

import javafx.application.Platform;
import javafx.fxml.FXML;
import javafx.scene.control.Button;
import javafx.scene.control.TextArea;
import org.example.service.AudioSource;
import org.example.service.SpeechService;
import org.example.service.TranslationService;
import org.example.service.TtsService;
import org.example.voicechat.VoiceChatAudioSource;

import java.io.InputStream;
import java.net.Socket;

public class MainController {
    private Socket chatSocket;
    private AudioSource chatSource;

    @FXML private Button startBtn;
    @FXML private TextArea recognizedText, translatedText;

    private final SpeechService speechService = new
SpeechService();
    private final TranslationService translationService = new
TranslationService();
    private final TtsService ttsService = new TtsService();

    private volatile boolean running = false;

    @FXML
    public void initialize() {
        startBtn.setOnAction(e -> {
            if (!running) {
                running = true;
                startBtn.setText("Стоп");

                try {
                    // 1) Підключаємося до голосового чату
                    chatSocket = new Socket("localhost", 55555);
                    InputStream chatIn =
chatSocket.getInputStream();

                    // 2) Створюємо адаптер і запускаємо
SpeechService
                    chatSource = new
VoiceChatAudioSource(chatIn);
                    speechService.start(chatSource);
```

```
                    // 3) Запускаємо цикл розпізнавання
                    new
Thread(this::runRecognitionLoop).start();
                } catch (Exception ex) {
                    ex.printStackTrace();
                    running = false;
                    startBtn.setText("Старт");
                }
            } else {
                running = false;
                startBtn.setText("Старт");
                speechService.stop();
                try {
                    if (chatSource != null) chatSource.stop();
                    if (chatSocket != null) chatSocket.close();
                } catch (Exception ex) {
                    ex.printStackTrace();
                }
            }
        });
    }

    private void runRecognitionLoop() {
        while (running) {
            String rec = speechService.recognize();
            if (!rec.isEmpty()) {
                Platform.runLater(() ->
recognizedText.appendText(rec + "\n"));
                String tr = translationService.translate(rec);
                Platform.runLater(() ->
translatedText.appendText(tr + "\n"));
                ttsService.speak(tr);
            }
        }
    }
}
```