Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра автоматизації та інтелектуальних інформаційних технологій

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«**Автоматизована система розпізнавання виголошених слів методами машинного навчання**»

Виконав: студент 2 курсу, групи 3АКІТ-22м,
спеціальність 151– «Автоматизація та комп'ютерно-інтегровані технології».
_____ Антон БІЛОУС

Керівник к.т.н., доцент кафедри АІІТ
_____ Гармаш ВОЛОДИМИР
«__4__» грудня 2023 р.

Опонент: к.т.н., доцент кафедри КН
_____ Олексій СІЛАГІН
«__8__» грудня 2023 р.

Допущено до захисту
Завідувач кафедри АІІТ

_____ д.т.н., проф. Олег БІСІКАЛО
«__11__» грудня 2023 р.

Вінниця ВНТУ – 2023 року

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра автоматизації та інтелектуальних інформаційних технологій
Рівень вищої освіти другий (магістерський)
Галузь знань 15 – Автоматизація та приладобудування
(шифр і назва)
Спеціальність 151 – Автоматизація та комп'ютерно-інтегровані технології
(шифр і назва спеціальності)
Освітня програма Інформаційні системи та інтернет речей
(назва освітньо-професійної програми)

ЗАТВЕРДЖУЮ
Завідувач кафедри АІІТ
д.т.н., проф. Олег БІСІКАЛО.
20 09 2023 року

ЗАВДАННЯ

НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Білоусу Антону Олександровичу
(прізвище, ім'я, по батькові)

1. Тема роботи: «Автоматизована система розпізнавання виголошених слів методами машинного навчання»
Керівник роботи Гармаш В.В. к.т.н., доцент кафедри АІІТ
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджене протоколом №24 засідання кафедри АІІТ від «18» 09 2023р.
2.Термін подання студентом роботи 05.12 2023 р.
3. Вихідні дані до роботи:
3.1. Середній час розпізнавання голосової команди: 2,5 с.
3.2. Максимальний час розпізнавання голосової команди: 3,2 с.
3.3. Середній час запуску програми: 6 с.

4. Зміст текстової частини
Вступ. Проектні рішення з розробки системи розпізнавання мовлення. Детальний огляд технології синтезу мовлення та розпізнавання мови. Математичне забезпечення для порівняння аудіофайлів. Опис архітектури програмного забезпечення для розпізнавання мовлення. Аналіз реалізації програмного забезпечення. Реалізація та тестування прогамного забезпечення. Висновки. Список використаних джерел. Додатки.

5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових кре
Схема структурна варіантів використання; Модель зі збереженням стану; Стр
шаблону MVC; Схема структурна класів моделей; Архітектура створеної си
Обробка голосової команди користувача.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада Консультанта	Підпис, дата	
		завдання видав	виконав прийняв
Спеціальна частина	к.т.н., доцент кафедри АПТ Володимир ГАРМАШ	20.09.2023	05.12.

7. Дата видачі завдання 20.09.2023р.

КАЛЕНДАРНИЙ ПЛАН

№	Назва та зміст етапу	Термін виконання		Примітки
		початок	закінчення	
1.	Вибір, узгодження та затвердження теми МКР	20.09.2023	27.09.2023	вик.
2.	Дослідження предметної області	28.10.2023	05.11.2023	вик.
3.	Обгрунтуванян методів реалізації та контролю	06.11.2023	17.11.2023	вик.
4.	Розробка структури та функціональної схеми	18.11.2023	24.11.2023	вик.
5.	Розробка програмного забезпечення	25.11.2023	30.11.2023	вик.
6.	Тестування та перевірка припущень	01.12.2023	06.12.2023	вик.
7.	Оформлення матеріалів до захисту МКР	07.12.2023	12.12.2023	вик.

Студент _____ Антон БІЛОУС
(підпис) (прізвище та ініціали)

Керівник роботи _____ Володимир ГАРМАШ
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

УДК 004.67

Білоус А.О. Автоматизована система розпізнавання виголошених слів методами машинного навчан. Магістерська кваліфікаційна робота зі спеціальності 151 – Автоматизація та комп’ютерно-інтегровані технології, освітня програма – Інформаційні системи і Інтернет речей. Вінниця: ВНТУ, 2023. 106 с.

На укр.мові. Бібліогр.: 27 назв; рис.:25; табл.: 2.

У даній магістерській кваліфікаційній роботі було розроблено автоматизовану систему розпознавання виголошених слів за допомогою машинного навчання. В роботі проводиться огляд існуючих алготитмів та підходів до розпізназання виголошених слів. Досліджено математичне забезпечення для вирішення поставленої задачі. Розроблено архітектуру програмного продукту для розпізнавання виголошених слі.

Ключові слова: нейронні мережі, машинне навчання, мовлення

ABSTRACT

Bilous A. O. Automated system for recognizing spoken words using machine learning methods. Master's thesis on specialty 151 - Automation and computer-integrated technologies, educational program - Information systems and internet of things. Vinnytsia: VNTU, 2023. 106 p.

In the Ukrainian language. Bibliography: 27 titles; fig.: 25; tabl.: 2.

In this master's thesis, an automated system for recognizing spoken words using machine learning was developed. The paper reviews existing algorithms and approaches to recognizing spoken words. Mathematical support for solving the given problem was studied. The architecture of a software product for recognizing spoken words has been developed.

Keywords: neural networks, machine learning, speech

ЗМІСТ

ВСТУП

Постійний розвиток інформаційних технологій призвів до збільшення кількості існуючих апаратних і програмних засобів, які використовуються в повсякденному житті. Це дає можливість впроваджувати сучасні технології в будь-яку сферу життя суспільства та приносити цим користь. Це призводить до того, різні групи людей можуть покращити своє життя за допомогою сучасних технологій, наприклад за допомогою новітніх систем і алгоритмів розпізнавання голосу є можливість створювати системи розпізнавання голосових команд.

У даній магістерській роботі розглядається питання автоматизації розпізнавання виголошених слів на прикладі налаштування вимови при вивченні іншомовних слів.

Вивчаючи іноземну мову, необхідно розвивати чотири види мовлення: читання, письмо, аудіювання та говоріння. Якщо перші три навички можна розвинути самостійно або за допомогою спеціальних програм, то говорити буде складніше, оскільки перевірити правильність вимови може вчитель іноземної мови або носій мови. Але найефективнішим способом поповнення словникового запасу є комплексний підхід до вивчення слів, що включає всі перераховані вище види мовленнєвої діяльності.

Є багато варіантів тренування вимови. Перший варіант - прослухати і повторити. Такий варіант не дуже ефективний, так як людині складно почути свою вимову з боку, а тим більше оцінити його. Другий спосіб – курси іноземних мов. Цей варіант ефективний, але досить дорогий. Третій спосіб – вивчення мови за допомогою спеціалізованих додатків для спілкування з носіями мови. Третій варіант дуже ефективний, але має ряд недоліків. По-перше, важко знайти людей, які справді зацікавлені в допомозі з вимовою, оскільки такі додатки в основному створені для того, щоб люди просто спілкувалися на різні теми, не думаючи про вимову. По-друге, деякі люди не бажають спілкуватися з незнайомцями, що є суттєвою перешкодою для вивчення мови на основі досвіду таких програм.

Тому сьогодні має сенс вирішувати проблеми вимови за допомогою інтелектуальних систем, які можуть аналізувати вимову користувачів.

Мета дослідження — дослідити та покращити процес розпізнавання вимовлених слів шляхом розробки програмної архітектури, яка порівнюватиме два аудіофайли. Перший аудіофайл зчитується з мікрофона користувача, а другий аудіофайл генерується системою за допомогою технології перетворення тексту в мову.

Для досягнення цієї мети необхідно виконати наступні завдання:

- Проаналізувати існуючі рішення проблеми та існуючі аналогічні рішення;
- Розробити програмну архітектури вимови;
- Зчитувати звуки з мікрофона користувача і зберігати їх в аудіофайли;
- Використати технологію перетворення тексту в мовлення для створення аудіо файлів із тексту;
- Створити тексту з аудіофайлів за допомогою технології перетворення мови в текст;
- Реалізувати алгоритм попередньої обробки та синхронізації аудіофайлів;
- Реалізувати алгоритм, який порівнює два аудіофайли для розпізнавання вимови;
- Проаналізувати результати роботи системи.

Об'єкт дослідження. Процес порівняння аудіофайлів для визначення правильної вимови.

Предмет дослідження – алгоритми та методи розпізнавання мови, методи порівняння аудіофайлів.

Отримані результати мають наукову новизну. Найважливішими науковими результатами магістерської роботи є:

- Запропоновані архітектурні рішення для побудови інтелектуальних систем розпізнавання мовлення та генерації вимови;

– Розроблена перша система вимови, яка, на відміну від аналогічних систем, не залежить від кількості та набору слів і фраз, а також є точнішою, оскільки порівняння здійснюються безпосередньо на аудіо файлах.

Практичне значення отриманих результатів. Розроблена система є додатком для вивчення вимови іншомовних слів.

1 ПРОЕКТНІ РІШЕННЯ З РОЗРОБКИ СИСТЕМИ РОЗПІЗНАВАННЯ МОВЛЕННЯ

1.1 Опис бізнес процесів

1.1.1 Опис процесу діяльності

Автоматизація націлена на процес розпізнавання мовлення, створення аудіофайлів із тексту та подальшу обробку аудіофайлів для визначення правильної вимови.

На рисунку та тексті подано структурну схему бізнес-процесу визначення правильної вимови.

Коли користувач натискає кнопку запуску завдання в додатку для вивчення іноземної мови, сервер робить запит до бази даних, щоб отримати текстове представлення слова, і передає отримане слово клієнту для відображення користувачеві. Користувач натискає кнопку запису та вимовляє слово, а потім натискає кнопку зупинки запису. У той же час на стороні сервера на основі текстового представлення слова використовується технологія перетворення тексту в мовлення для генерації його звукового представлення. Після того, як користувач припиняє запис, клієнт передає запис на сервер, де порівнюються два аудіофайли - зчитаний з мікрофона користувача і згенерований системою. Якщо при аналізі двох аудіофайлів виявлено однакову вимову, кількість правильних відповідей збільшується на 1. Якщо вимова не збігається, кількість неправильних відповідей збільшується на 1, а неправильно вимовлене слово відображається на сторінці результатів після виконання завдання. Кожне слово можна прослухати, і відображається транскрипція кожного слова, вказуючи, який звук вимовлено неправильно.

1.1.2 Варіанти використання

Користуючись системою, користувач має можливість виконати завдання з вимови та отримати результати цього завдання. Після виконання завдання користувач

має можливість почути неправильно вимовлені слова та переглянути транскрипцію цих слів із зазначенням неправильно вимовленої частини.

Взаємодія між користувачем і системою показано на рисунку 1.2 – Діаграма структури варіантів використання.

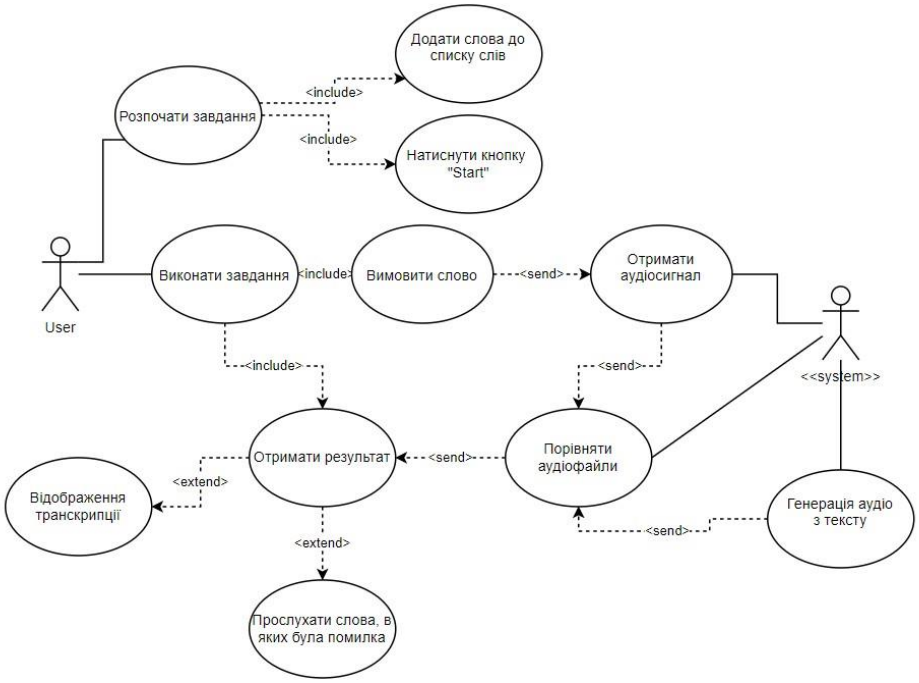


Рисунок 1.1 – Схема структурна варіантів використання

1.2 Постановка задачі

Метою цього дослідження є розробка системи, яка може порівнювати два аудіофайли для перевірки правильності вимови іноземних слів. Перший аудіофайл зчитується з мікрофона користувача, а другий аудіофайл генерується системою за допомогою технології перетворення тексту в мову.

Для досягнення цієї мети необхідно виконати наступні завдання:

- Проаналізувати існуючі рішення проблеми та існуючі аналогічні рішення;
- Зчитувати звуки з мікрофона користувача і зберігати їх в аудіофайли;
- Використовуйте технологію перетворення тексту в мовлення для створення аудіофайлів із тексту;

- Створення тексту з аудіофайлів за допомогою технології перетворення мови в текст;
- Реалізувати алгоритм попередньої обробки синхронізації аудіофайлів;
- Реалізуйте алгоритм, який порівнює два аудіофайли для виявлення неправильної вимови;
- Аналізуйте результати системи.

1.3 Наявні рішення

Основними завданнями цієї системи є синтез мовлення, розпізнавання та порівняння вимови при вивченні іноземних мов, особливо англійської.

1.3.1 Наявні технічні рішення для синтезу мовлення

При аналізі існуючих технологічних рішень синтезу мовлення були обрані найбільш популярні системи синтезу мовлення, а саме:

- Амазонка Поллі;
- Синтез тексту в мову Azure;
- Синтез мовлення Google Cloud;
- Перетворення тексту в мову IBM Watson.

Розглянемо кожне з них і спробуємо виділити його переваги та недоліки, щоб визначити, яке з цих рішень більше підходить для вирішення задач синтезу мовлення.

Amazon Polly — це сервіс, який перетворює текст на справжнє мовлення, що дозволяє створювати мовні програми та абсолютно нові категорії мовних продуктів. Служба перетворення тексту в мовлення Polly використовує передову технологію глибокого навчання для синтезу природного звучання людської мови. Завдяки автентичним голосам десятками різних мов ви можете створювати програми з підтримкою мови, які можна використовувати в багатьох країнах.

Окрім стандартного мовлення TTS, Amazon Polly пропонує мовлення Neural Text-to-Speech (NTTS), яке покращує якість мовлення за допомогою нових методів машинного навчання. Технологія Neural TTS від Polly також підтримує два стилі

мовлення, щоб краще відповідати стилю мовлення доповідача програмі: стиль читання новин, спеціально розроблений для розповіді новин, і стиль мовлення, ідеальний для програм телефонних дзвінків та іншого двостороннього спілкування.

Основні переваги Amazon Polly:

- Система підтримує більше 15 різних мов;
- Користувачі Amazon Polly мають повний контроль над своїм голосом, від висоти до вимови;
- Має підтримку платформи та мови програмування: підтримує всі мови програмування, що входять до набору інструментів Amazon Web Services SDK;
- Користувачі мають можливість створювати власні словники. Звичайно, ця система має і недоліки:
- Amazon Polly не розпізнає певні символи;
- Не підтримує переклад між мовами;
- Деякі голоси Амазонки Поллі все ще звучать дещо роботизовано.

Azure Text to Speech — це система TTS Microsoft, яка дозволяє програмам, інструментам або пристроям перетворювати текст у синтетичне мовлення, схоже на людське. Ви можете вибрати з понад 75 стандартних голосів, доступних понад 45 мовами. Є також 5 нервових голосів. За допомогою Azure Text-to-Speech ви можете створити власний голос, унікальний для продукту чи бренду, який ви розробляєте.

Ключові переваги синтезу мовлення Azure:

- Підтримує велику кількість мов;
- Можливість створювати унікальні звуки. Основні недоліки синтезу мовлення Azure:
- Якість звуку не дуже хороша;
- Звук нереалістичний;
- Досить складна інтеграція.

Google Cloud Text-to-Speech дозволяє розробникам синтезувати природне мовлення з понад 220 голосів понад 40 мовами. Система застосовує новаторські

дослідження DeepMind щодо WaveNet і потужних нейронних мереж Google для забезпечення високоякісного звуку.

Ключові переваги синтезу мовлення Google Cloud:

- Якісні презентації;
- Безліч звуків на вибір;
- Можливість створювати власні звуки;
- Людська інтонація;
- Проста інтеграція API.

При аналізі основних недоліків явних недоліків виявлено не було.

IBM Watson Text to Speech – служба синтезу мовлення розуміє текст і природну мову Генерує синтезований звуковий сигнал з відповідною частотою та інтонацією. Synthesis забезпечує 27 голосів (13 нейронних і 14 стандартних) на 7 мовах.

Ключові переваги IBM Watson Text to Speech:

- Проста інтеграція API;
- Підтримує кілька мов програмування. Основні недоліки IBM Watson Text to Speech:
- Підтримується не так багато мов, як у конкурентів;
- Вимова не завжди чітка;
- Дорожче, ніж інші аналогічні рішення, розглянуті вище.

1.3.2 Наявні технічні рішення для розпізнавання мовлення

При аналізі існуючих технологічних рішень розпізнавання мовлення були обрані найпопулярніші системи перетворення мови в текст, а саме:

- AssemblyAI – API перетворення мови в текст;
- IBM Watson voice-to-text;
- Транскрибовано Amazon;
- Google Cloud Speech to Text.

AssemblyAI - Speech to Text API використовує найновіші дослідження глибокого навчання для забезпечення точної транскрипції аудіо. Однією з

найважливіших особливостей AssemblyAI є можливість застосовувати кілька різних бібліотек для різних акцентів, якості запису та середовища запису, враховуючи при цьому кількість фонового шуму. Сама по собі ця функція робить AssemblyAI заслуговуючим уваги, оскільки транскрипція аудіо з динаміків із незнайомими акцентами може бути важкою та трудомісткою.

Основні переваги AssemblyAI:

- Значна точність і швидкість;
- хороша ціна;
- Проста інтеграція API;
- Підтримує кілька мов програмування. Основні недоліки AssemblyAI:
- Немає людської точності.

IBM Watson Speech to Text — це хмарне рішення, яке використовує алгоритми глибокого навчання для застосування знань про граматику, структуру мовлення та синтез аудіо для створення настроюваного розпізнавання мовлення для оптимальної транскрипції тексту.

Ключові переваги IBM Watson Speech to Text:

- Проста інтеграція API;
- Правильно тлумачити речення та їх контекст. Основні недоліки IBM Watson Speech to Text:
- Зосередьтеся на мові кількох людей. Якщо будь-яке мовлення розпізнається, IBM Watson Speech to Text намагається перетворити його на текст, що значно впливає на якість розпізнавання;
- Кілька підтримуваних мов.

Amazon Transcribe — це служба автоматичного розпізнавання мовлення (ASR), яка спрощує розробникам реалізацію перетворення мови в текст у програмних програмах. Використовуючи API Amazon Transcribe, ви можете аналізувати аудіофайли, що зберігаються в Amazon S3, і отримувати текстові файли транскрибованою мовою.

Основні переваги Amazon Transcribe:

- Підтримує кілька аудіоформатів;
- Можна підібрати розмовник. Основні недоліки Amazon Transcribe:
- Розпізнавання сильно залежить від контексту;
- Якщо розмір аудіосигналу, що подається на вхід, невеликий, час обробки може бути досить довгим.

Google Cloud Speech-to-Text використовує найсучасніший алгоритм глибокого навчання Google Neural Networks для автоматичного розпізнавання мовлення.

Ключові переваги голосового перетворення тексту в Google Cloud:

- Підтримує 125 мов і варіантів вимови;
- Можливість вибору навченої моделі;
- Проста інтеграція API.

При аналізі основних недоліків явних недоліків виявлено не було.

1.3.3 Відомі аналоги

Ми розробимо критерії оцінки існуючих програмних рішень для обраного завдання.

Сьогодні існує безліч голосових помічників, які можуть вирішувати різноманітні завдання, пов'язані з пошуком інформації, лікуванням, спрощеним використанням пристрою, бізнес-діяльністю тощо. Багато існуючі варіанти можуть бути використані для вирішення завдань програмування мовлення в тій чи іншій мірі, незважаючи на те, що якість таких продуктів залишає бажати кращого в контексті обраної проблеми. Зважаючи на специфіку ринку, для спрощення роботи інтегрованих середовищ програмування створено лише кілька існуючих сервісів. Тому першим і одним із основних критеріїв оцінки існуючих програмних рішень є позиціонування голосових помічників для прискорення та спрощення процесу написання програмних текстів для розробників.

Іншим критерієм оцінки є те, що система має бути відкритим кодом і вільною для використання. Програмне забезпечення з відкритим кодом схоже на науку, оскільки дослідження проводяться в рецензованому середовищі. Це означає, що наука, по суті, є відкритим кодом, що дозволяє ділитися отриманими знаннями з

іншими для спільного використання та подальшого розвитку. Програмні рішення, побудовані на принципах доступності та відкритості, дозволяють створювати нові, більш потужні інструменти, змінювати або комбінувати існуючі інструменти відповідно до ваших потреб. Програмне забезпечення з відкритим кодом є важливим, оскільки воно полегшує розробку програмного забезпечення та робить його доступним для більшого кола користувачів, ніж платні рішення.

Наступною важливою особливістю цього продукту є його платформенність. Перевага продукту платформи над традиційним продуктом полягає в тому, що користувачі можуть розширити продукт, динамічно додаючи нові функції, щоб принести користь іншим користувачам. Будь-який продукт може стати платформою, що дозволяє користувачам додавати послуги та функції. API — це інструменти, які полегшують використання платформи та дозволяють користувачам розширювати та додавати послуги поверх існуючих продуктів. Коли продукт стає платформою, він отримує новий тип користувачів: сторонніх розробників.

Ще одним важливим критерієм оцінки існуючих рішень є якість документації розробленої системи. Документація програмної системи — це технічний опис продукту, який містить інструкції щодо ефективного використання та інтеграції з системою. Документація містить усю інформацію, необхідну для використання системного API, детальну інформацію про функції, типи, які вони повертають, параметри, які вони приймають, класи тощо, і часто містить приклади використання певних функцій. Наявність високоякісної технічної документації скорочує час, необхідний для інтеграції з системами, заощаджуючи час, необхідний для підтримки існуючих систем, Допоможіть більшій кількості користувачів зрозуміти продукт і почати ним користуватися. Для платформних систем ці функції надзвичайно важливі.

Остаточним критерієм для оцінки існуючих програмних рішень є здатність розпізнавати команди, сформовані звичайною мовою, а не наперед визначені фрази. Використання обробки природної мови дає змогу використовувати мовлення для взаємодії з системами у спосіб, з яким люди знайомі, і без необхідності вивчати синтаксис нової мови чи нові команди для використання програмного рішення. Таким

чином, системи, які використовують цей підхід, мають значні переваги порівняно з іншими системами.

У результаті дослідження визначено критерії оцінки програмних засобів для створення голосових помічників, що дозволить провести детальний порівняльний аналіз існуючих програмних рішень та врахувати їх переваги та недоліки при розробці запропонованого засобу. Розглянуті моменти є основою для всіх інших аспектів дослідження.

Alexa — це універсальний голосовий помічник, розроблений і реалізований Amazon. Він включає в себе можливість відтворювати музику та аудіокниги, встановлювати будильники, створювати списки справ і надавати в реальному часі прогноз погоди, затори та інші новини. Користувачі можуть вибрати одну або кілька функцій за допомогою голосової взаємодії з помічником. Alexa також широко використовується в галузі Інтернету речей, де вона надає можливість керувати автоматизованими домашніми системами за допомогою розумних пристроїв.

Користувачі можуть розширити можливості Alexa, встановивши наявні або написавши власні «навички» (додатковий функціонал, створений сторонніми розробниками). Користувачі можуть публікувати створені ними «навички» в системі Alexa Skills, де інші користувачі можуть використовувати їх на своїх пристроях. Багато навичок повністю працюють у хмарі, використовуючи всі можливості Amazon Web Services.

Amazon також надає інтерфейси API, щоб дозволити розробникам інтегрувати голосові функції Alexa у власні продукти. Продукти, створені за допомогою цього інтерфейсу, мають доступ до всіх функцій голосового помічника. Alexa забезпечує автоматичне розпізнавання мовлення за допомогою хмарних служб і інструментів обробки природної мови. Використовуйте глибоке машинне навчання (тобто довгострокові та короткочасні штучні нейронні мережі), щоб генерувати голосові відповіді помічнику.

Недоліком цієї системи є те, що це загальна система і немає готових рішень для вирішення проблеми голосового керування системами розробки програмного

забезпечення. Крім того, використання Alexa API не є безкоштовним, тому текст системної програми не відкривається.

Перевагами голосового помічника Alexa є висока точність розпізнавання команд користувача, модульність системи та можливість розробникам додавати власні функції.

Голосовий помічник компанії Google доступний на мобільних платформах і пристроях розумного будинку. Користувачі в основному взаємодіють з Google Assistant за допомогою голосових команд, але також доступний ввід з клавіатури.

Google Assistant надає можливості голосового керування та пошуку, дозволяючи користувачам виконувати різноманітні завдання: керувати розумними будинками, отримувати доступ до своєї особистої інформації в службах Google, шукати інформацію в Інтернеті, бронювати найближчі ресторани чи квитки, перевіряти погоду та погоду та більше. Новини, слухайте музику, запускайте таймери та нагадування, організовуйте зустрічі, запускайте програми на своєму пристрої, надсилайте повідомлення, перекладайте в реальному часі тощо.

Google Assistant може визначити контекст, у якому ставиться запитання. Це важливо, оскільки голосове керування стає потужнішим, не лише відповідаючи на конкретні фрази чи команди, але й враховуючи попередні запитання користувача. За словами Google, у майбутньому помічник зможе здійснювати дзвінки та призначати зустрічі самостійно, а не за допомогою користувача.

Якщо ви порівняєте Google Assistant з голосовим помічником Amazon Alexa, про який йшлося вище, ви побачите, що обидві платформи схожі одна на одну, оскільки пропонують подібні функції та використовуються на тих самих пристроях. Однак Google має явну перевагу з операційною системою Android, оскільки вона має доступ до всієї особистої інформації користувача.

Недоліком цієї системи є те, що вона не орієнтована на вирішення задачі голосового керування системами розробки програмного забезпечення, але забезпечує широкий набір інших функцій. Інтеграція віртуального помічника Google теж не безкоштовна.

Перевагами голосового Google Assistant є висока точність обробки запитів користувачів, хороша документація та модульність системи.

Cortrana - це голосовий помічник, створений Microsoft. Можливості Cortana включають встановлення нагадувань і будильників, розпізнавання голосу користувача, Відповідайте на запитання, використовуючи інформацію, отриману з пошукової системи Bing (наприклад, надаючи інформацію про поточні спортивні результати, погоду, курси акцій і валют тощо). Cortrana здатна запам'ятовувати голос користувача і реагувати тільки на його команди. Крім голосового введення, голосові помічники дозволяють клієнтам надавати інформацію за допомогою клавіатури або іншого пристрою введення.

Cortana також пропонує можливість пошуку файлів і папок не лише на певному пристрої, але й на підключених пристроях у хмарі, наприклад у сховищі OneDrive. Новини, які надає голосовий помічник, створюються на основі інтересів користувача та історії пошуку, тож ви можете побачити Cortrana, інтегровану з системою рекомендацій Microsoft. Cortana може перевіряти вашу електронну пошту та календар і надсилати користувачам голосові сповіщення про них. Також голосовий помічник може розповідати користувачам анекдоти.

Cortrana — це модульна система, яка дозволяє розробникам додавати власні функції шляхом розробки та інтеграції

«Навички» схожі на голосового помічника Amazon. Інші переваги системи включають підтримку кількох мов для взаємодії з голосовими помічниками, документацію, яка містить детальні описи, і інтеграцію з операційними системами Windows.

Недоліком Cortrana є те, що вона не орієнтована на вирішення задачі голосового керування інтегрованим середовищем розробки програмного забезпечення, і вимагає значних зусиль для реалізації власних «навичок» для забезпечення вищевказаних функцій. Ще одним недоліком є те, що система не безкоштовна, тому не всі бажаючі зможуть нею скористатися.

VoiceCode — це голосовий помічник, який не тільки дозволяє розробляти програмне забезпечення за допомогою голосу, але й підвищує ефективність роботи

користувачів. Він використовує SmartNav замість миші та запускає Dragon у фоновому режимі для обробки голосових команд.

Команди, які може обробляти система, не є звичайними словами в розмовній мові, а складаються зі спеціальних слів, які зазвичай містять один склад для полегшення обробки. Щоб почати користуватися системою, користувач повинен ознайомитися з цими командами для подальшого використання. Іншим недоліком цієї системи є те, що вона розроблена лише для операційної системи Mac OS і не працює з найбільш часто використовуваними платформами, такими як Windows або Unix. Крім того, це не безкоштовно, оскільки окрім оплати самого голосового помічника, вам також потрібно окремо придбати ліцензію на зовнішню систему, яку використовує голосовий помічник, наприклад Dragonfly або SmartNav.

Незважаючи на ці недоліки, voicecode.io залишається одним із найбільш багатофункціональних рішень для спрощення голосового введення.

Сільвій — нащадок дракона Натури та Енеї. Він використовує власну структуру Kaldi для розпізнавання мовлення. Голосові помічники можуть працювати онлайн або на невеликих вбудованих пристроях. Silvius працює шляхом передачі вихідного сигналу мікрофона на сервер, який відповідає текстовим вмістом, який він може розпізнати. Потім згенерований текст буде пропущено через граматику, імітуючи натискання відповідної клавіші на клавіатурі. Хоча рішення використовує мережу для зв’язку з сервером, голосовий помічник все ще демонструє високу швидкість реагування.

Потужним нововведенням Silvius є те, що він спирається на незалежний від платформи алгоритм розпізнавання мовлення, що дає змогу використовувати голосового помічника в будь-якій операційній системі. Ще однією перевагою цієї системи є те, що вона безкоштовна, а її текст програми повністю відкритий і може бути розширений і змінений. Ще однією перевагою є те, що система дуже проста в установці. У процесі розробки використовувалися мова програмування Python і фреймворк аналізу даних Spark.

У голосовому помічнику Silvius, як і в VoiceCode, користувачі видають команди не у вигляді звичайних вимовлених слів, а за допомогою спеціальних слів, на

вивчення яких потрібен час. Крім того, команди, які надає система, не є командами високого рівня і часто зводяться до простого диктування тексту програми розробником. Іншим недоліком є низька якість документації, що ускладнює використання системи користувачами та збільшує час, необхідний для початку використання.

Vocola — це мова голосових команд для диктування програмного тексту за допомогою простого синтаксису. Завдяки цій системі багато частково паралізованих розробників навчилися програмувати за допомогою мови. Доступні команди описуються за допомогою спеціального словника, який створює відповідність між голосовими командами та клавішами на клавіатурі, які будуть натискатися.

Vocola проста в установці та використанні з простим синтаксисом для створення команд. Багато існуючих систем розпізнавання мовлення прив'язані до певних мов програмування. Це означає, що будь-яка поведінка системи може бути визначена лише на рівні синтаксису вибраної мови програмування. Навпаки, Vocola розроблена як мова голосових команд, а не як мова програмування загального призначення. Серед інших переваг цієї системи можна відзначити її повну безкоштовність.

Недоліком цієї системи є те, що користувач фактично диктує текст програми голосом, а не видає системі інструкції високого рівня. Загалом, ця система краще підходить для таких завдань, як відкриття файлів або вкладок у браузері, а не для програмування.

Ми переглянули кілька продуктів на ринку для таких програм і знайшли або програми для вивчення слів без завдань з вимови, або програми лише для вимови.

Програма Vocabulary надає користувачам можливість вивчати слова за допомогою карток і завдань на основі карток.

При аналізі вимовних додатків для дослідження були обрані такі додатки:

Скажи це. Інтерфейс Utter добре розроблений і виглядає професійно. Після авторизації в додатку користувач починає проходити напружений урок, на якому бот зі штучним інтелектом розповідає про програму. Він також перевіряє користувачів за допомогою інтерактивного чату, де вони повинні відповісти на кілька запитань. Це

важливо для правильного налаштування програми. Є різні курси для різних сценаріїв. Наприклад, ви можете вибрати курс, щоб поспілкуватися з друзями про подорожі, офіс, роботу тощо. Навчання проводиться на трьох рівнях:

- Початковий - для повсякденних завдань, таких як запит часу або пошук маршруту;
- Intermediate - для звичайних розмов з друзями;
- Додатковий - для офісного, робочого спілкування. [1]

Вивчивши ази, почніть серйозно вивчати англійську вимову. Користувачам необхідно записувати власні голоси та відповідати на різні запитання. Для кожного рівня передбачено два безкоштовних уроки. Щоб продовжити навчання, вам потрібно буде заплатити $2,49. [1]

Ельза. ELSA (назва розшифровується як English Language Voice Assistant) – це програма для інтерактивного спілкування. Це допомагає покращити вимову та загалом краще почати говорити англійською. ELSA пропонує теми курсів для різних студентів, від водіїв таксі до персоналу служби підтримки клієнтів. Для кожної теми є багато уроків. Коли користувач запускає одну з них, програма відтворює слова та фрази. Користувач повинен їх повторити. Користувач відразу зрозуміє оцінку його вимови. Залежно від результатів програма попросить вас повторити слово чи фразу або дозволить продовжити. Якщо вам важко зі словами, скористайтеся підказками щодо рухів язика та губ. Вам потрібно натиснути на кнопку у вигляді вуха, щоб прослухати слова, що відтворюються. Програма має вбудований словник із 2000 часто вживаних слів і фраз. [1]

Натті Вокерс. Nativox прагне допомогти користувачам краще зрозуміти особливості розмовної американської англійської мови. Використовуючи Nativox, користувачі чують посилені звуки та бачать, як слова поєднуються разом у веселих схемах. Це полегшує розуміння того, як вимовляється кожна частина речення. Потім користувач зможе повторити фразу і записати свій голос. Вимову можна імітувати до тих пір, поки користувач не зможе максимально наблизитися до зразка. Nativox розроблений, щоб дозволити користувачам вивчати переклад слів, навчаючись

говорити. Тож мозок може зосередитися на розмові, а не на граматиці чи словниковому запасі. [1]

Англійська вимова. English Pronunciation – це невелика програма, яка не лише повідомляє вам Як правильно вимовляти слова, а для цього показувати правильне положення артикуляторів. При першому запуску програми необхідно вибрати рідну мову. Потім програма покаже вам, як вимовляти голосні та приголосні звуки. Натиснувши на голосну або приголосну букву, користувачі побачать розширені параметри. Положення язика та губ буде відображено на екрані з детальними текстовими інструкціями. Є також різні приклади слів, які можна почути з британським або американським акцентом. У розділі практики є деякі слова, у яких пропущені літери. Потрібно підібрати правильні літери і набрати все слово повністю. Є контрольні точки та відстеження прогресу, щоб полегшити процес навчання. На жаль, ви не можете просто ввести слово, щоб почути його вимову. Програма поширюється безкоштовно, але має рекламу та покупки в програмі. Ви отримуєте додаткові тести та словесні приклади. [1].

1.4 Глибоке навчання

Глибоке навчання — це техніка машинного навчання, яка вчить комп’ютери робити те, що люди роблять природно: вчитися на прикладі. Глибоке навчання є ключовою технологією, що стоїть за безпілотними автомобілями, які можуть розпізнавати дорожні знаки, смуги руху або відрізняти пішоходів від ліхтарних стовпів. Це ключ до голосового керування такими домашніми пристроями, як телефони, планшети, телевізори та колонки. Глибоке навчання останнім часом приділяє багато уваги, і це не дарма. Це результат, якого раніше було неможливо досягти.

У глибокому навчанні комп’ютерні моделі вчаться виконувати завдання класифікації безпосередньо з зображень, тексту чи звуків. Моделі глибокого навчання можуть досягти найсучаснішої точності, яка іноді перевищує людський рівень. Модель навчається з використанням великих обсягів позначених даних і архітектури нейронної мережі, що містить багато рівнів.

Глибоке навчання забезпечує вищу точність розпізнавання, ніж будь-коли раніше. Це допомагає виробам споживчої електроніки відповідати очікуванням користувачів і має вирішальне значення для додатків, які мають на меті безпеку, наприклад автономних транспортних засобів. Останні досягнення в глибокому навчанні досягли такого рівня, коли глибоке навчання перевершує людей у певних завданнях, таких як класифікація об’єктів на зображеннях.

Хоча глибоке навчання вперше було сформульовано в 1980-х роках, останнім часом воно стало корисним з двох основних причин.

Для глибокого навчання потрібні великі обсяги позначених даних.

Наприклад, для розробки безпілотного автомобіля потрібні мільйони зображень і тисячі годин відео.

Для глибокого навчання потрібна потужна обчислювальна потужність. Високопродуктивні графічні процесори мають паралельну архітектуру, яка забезпечує ефективне глибоке навчання. У поєднанні з кластерними або хмарними

обчисленнями розробники можуть скоротити час навчання для моделей глибокого навчання з тижнів до годин або менше.

Більшість методів глибокого навчання використовують архітектуру нейронної мережі, тому моделі глибокого навчання часто називають глибокими нейронними мережами. Термін «глибина» зазвичай стосується кількості прихованих шарів нейронної мережі. Традиційні нейронні мережі містять лише 2-3 прихованих шари, тоді як глибокі мережі можуть мати до 150.

Моделі глибокого навчання навчаються з використанням великої кількості позначених даних і архітектур нейронних мереж, щоб вивчати функції безпосередньо з даних без необхідності вилучення функцій вручну.

Три основні типи нейронних мереж складають основу більшості попередньо навчених моделей глибокого навчання:

- Пряме розповсюдження нейронних мереж;
- Згорточна нейронна мережа;
- Повторювана нейронна мережа.

Усі нейронні мережі складаються з трьох типів шарів, як показано на рисунку 1.2. Рівні нейронної мережі:

- вхідні;
- приховати;
- вихідні.

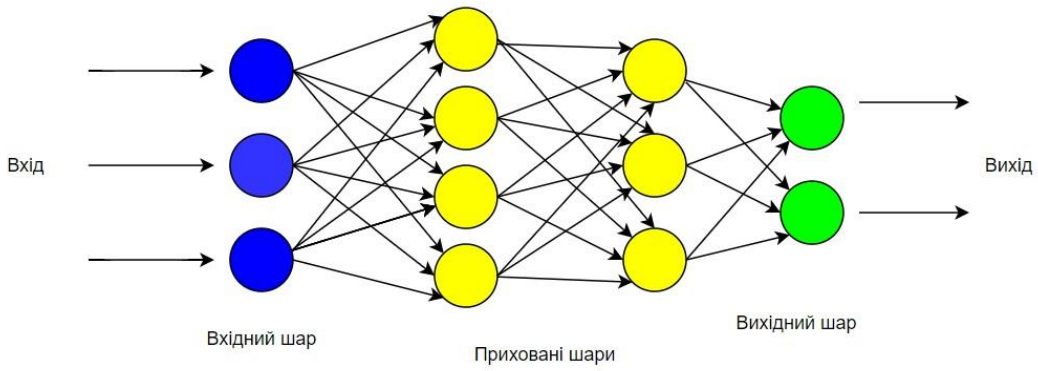


Рисунок 1.2 – Шари нейронної мережі

Нейронні мережі прямого зв’язку зазвичай використовуються для вирішення проблем із такими типами даних, як табличні та текстові дані, а також зображення. Рекурентні нейронні мережі використовуються для вирішення проблем, пов’язаних із даними часових рядів, текстовими даними та аудіоданими. Згорткові нейронні мережі особливо часто використовуються для вирішення проблем, пов'язаних з обробкою зображень і відео.

Висновок цього розділу

У цьому розділі розглядається бізнес-процес системи розпізнавання мовлення для вивчення іноземних мов, а також розроблено структурну діаграму бізнес-процесу та структурну діаграму варіантів використання.

Були розглянуті найпопулярніші технологічні рішення для синтезу мовлення, а саме Amazon Polly, Azure Text to Speech, Google Cloud Text-to Speech, IBM Watson Text to Speech. Після аналізу переваг і недоліків Google Cloud Text-to-Speech API було обрано для синтезу мовлення. Крім того, були розглянуті найпопулярніші технології розпізнавання мови. Також розглядалися API від Amazon, Google і IBM. Після аналізу переваг і недоліків Google Cloud Speech-to-Text API було вибрано для вирішення проблеми розпізнавання мовлення. На додаток до доступних технічних рішень, було проаналізовано додатки моделювання, і було виявлено, що розроблена система навчання іноземної мови не мала точного моделювання.

У цьому розділі представлені теоретичні відомості в області глибинного навчання.

2 ДЕТАЛЬНИЙ ОГЛЯД ТЕХНОЛОГІЙ СИНТЕЗУ МОВИ ТА РОЗПІЗНАВАННЯ МОВЛЕННЯ

2.1 Технологія синтезу мови

Комп'ютерне мовлення існує вже досить давно. Однак якість створюваного мовлення все ще недостатньо схожа на людину та її нелегко сприйняти. Хоча подібні системи досить близькі, вони недостатньо близькі, і навіть системи таких гігантів, як Google, Apple або Amazon, не схожі на людські. У цьому розділі досліджується, як останні досягнення в синтезі мовлення використовують методи глибокого навчання для створення природного звучання мови.

Щоб зрозуміти, чому методи глибокого навчання використовуються сьогодні для синтезу мовлення, важливо зрозуміти, як генерується мова. Існує два специфічні методи перетворення тексту в мову (TTS): параметричний TTS і спільний TTS. Також важливо визначити наступні два терміни для оцінки якості створеної мови: зрозумілість і природність. Чіткість — це якість виробленого звуку. Природність - це якість створеного мовлення.

2.1.1 Сполучна TTS система

Як випливає з назви, технологія заснована на записі високоякісних аудіосегментів (або фрагментів) і подальшому їх об’єднанні для формування мови. Хоча згенероване мовлення дуже чітке та чітке, воно звучить беземоційно. Звук чистий, але звучить неприродно. Це тому, що, враховуючи можливі комбінації емоцій, просодії, акценту тощо, досить важко знайти записи всіх можливих вимов слова. Звичайно, ці системи потребують величезних баз даних і вже закодованих комбінацій для утворення цих слів. Щоб розробити надійну систему, потрібно сім-вісім місяців.

2.1.2 Параметрична TTS система

Підключення систем TTS дуже обмежене через високі вимоги до даних і розподіл часу на розробку. Ось чому був розроблений більш статистичний підхід. Цей метод генерує мовлення шляхом поєднання таких параметрів, як основна частота, спектральна амплітуда тощо, і обробляє їх для генерування мовлення. Система параметрів TTS має дві фази. На першому етапі під час опрацювання тексту виділяються мовні ознаки. Такими ознаками можуть бути фонеми, тривалість тощо. На другому етапі виділяють ознаки вокодера, що представляють відповідні мовні сигнали. Такими ознаками можуть бути кепстр, спектрограма, основна частота тощо, які представляють деякі характеристики, властиві людській мові та використовуються для обробки звуку. Ці ручні параметри разом із лінгвістичними особливостями вводяться в математичну модель, яка називається вокодер. Вокодер приймає ці параметри та виконує кілька складних перетворень цих параметрів для генерації аудіосигналу. Під час генерації сигналу вокодер оцінює такі характеристики мови, як фаза, просодика (ритм і наголос), інтонація тощо.

Параметричне синтезоване мовлення є дуже модульним і цілком здійсненним. Якщо параметри, які впливають на мовлення, можна наближено визначити, то модель можна навчити генерувати всі типи мовлення. Створення такої системи вимагає набагато менше даних і зусиль, ніж підключення системи TTS.

2.1.3 Синтез мовлення з використанням глибокого навчання

Моделі глибокого навчання виявилися дуже ефективними для вивчення властивих характеристик даних. Ці функції не сприймаються людьми, але сприймаються комп’ютерами та краще представляють дані моделі. Іншими словами, модель глибокого навчання вивчає функцію, яка відображає вхідні дані X і вихідні дані Y. Згідно з цим припущенням, система синтезу мовлення з природним звучанням повинна сприймати вхідний сигнал X як текстовий рядок і виводити звуковий сигнал Y. Система не повинна використовувати жодних ручних функцій, але повинна вивчати нові багатовимірні функції, щоб відображати мову, людську поведінку.

Аудіофайли представлені в комп’ютерах шляхом оцифрування звукових сигналів. По суті, це часовий ряд зразків аудіо. Тож має сенс генерувати зразки аудіо безпосередньо замість того, щоб генерувати деякі приховані параметри, а потім обробляти їх для отримання аудіо. WaveNet від DeepMind — це новаторська робота з використанням глибоких нейронних мереж для створення звуків рівня семплів.

2.1.4 WaveNet

WaveNet генерує один семпл для аудіо, причому кожен семпл визначається всіма попередніми семплами, згенерованими в аудіо. Потім цей візерунок використовується з попереднім для створення наступного візерунка. Це називається авторегресійною генерацією.

$$p(X) = \prod_{i=0}^{T-1} p(x_{i+1}|x_1, \dots, x_i)$$

WaveNet побудовано з використанням стека згорткових шарів із залишками та пропусками зв’язків між ними. Він приймає оцифрований необроблений аудіосигнал як вхідний сигнал, потім проходить через ці згорткові шари та виводить зразки сигналів.

Модель є самодостатньою, тобто їй не подається жодна інформація про структуру мови, тому вона не генерує жодних значущих звуків. Якщо навчити цю модель на аудіо розмови людей, вона видаватиме звуки, схожі на те, що люди вимовляють якісь слова, але слова звучать так, ніби вони бурмочуть. Потім команда WaveNet надала моделі параметри вокодера з існуючої системи TTS, а також вихідний аудіосигнал на стороні входу. В результаті кінцева система синтезує якісну мову. Дуже чисто, без шуму, без нерозбірливих шепотів. Однак цей розрахунок дуже дорогий. Типовий WaveNet для високоякісної мови використовує 40 таких згорткових шарів з іншими зв’язками між ними. Оскільки зразки генеруються по одному, потрібно обробити 16 000 зразків, щоб створити 1 секунду звуку 16 кГц. Команда WaveNet повідомляє, що для створення 1 секунди звуку потрібно приблизно 4 хвилини.

2.1.5 SampleRNN

SampleRNN — це ще один метод генерації аудіосигналу, який використовує ієрархію повторюваних шарів з різною тактовою частотою для обробки звуку. Ідея полягає в тому, що кілька RNN з'єднані в ієрархічну структуру. Верхній рівень приймає великі фрагменти вхідних даних, обробляє їх і передає на нижній рівень; нижній рівень приймає менші фрагменти вхідних даних і передає їх далі. Це продовжується до найнижчого рівня в ієрархії, де генерується один зразок. Рисунок 2.1 – Трирівневий SampleRNN показує приклад трирівневого SampleRNN.

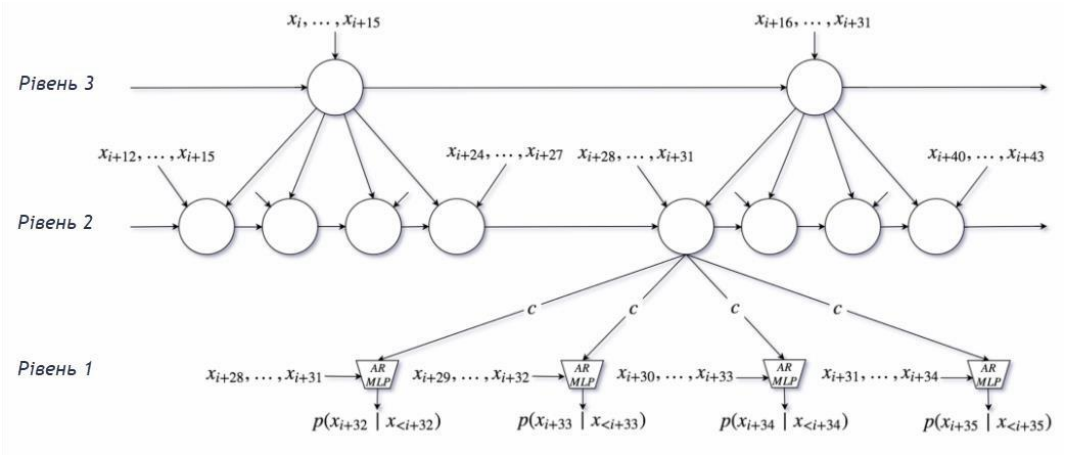


Рисунок 2.1 – Трирівневий SampleRNN

Як і WaveNet, це також авторегресійна генеративна модель. Але швидкість обчислень дуже висока порівняно з WaveNet.

2.1.6 Tacatron

Tacatron — це безперервний синтезатор мови. Tacotron не робить жодних припущень щодо того, які функції слід передати вокодеру, а також як текст має оброблятися. Команда Tacotron знає, що люди знають не все, тому вони дозволяють своїм моделям вивчати відповідні функції та процеси. Тому Tacotron переходить на рівень персонажа. Він приймає текстові символи як вхідні дані, передає їх до різних підмодулів нейронної мережі та генерує звукову спектрограму. Це більше схоже на

повну модель глибокого навчання для синтезу мовлення, і, на відміну від WaveNet, вона не вимагає жодних функцій від існуючих систем TTS.

Модель Tacatron показана на рисунку 2.2 – Архітектура Tacatron.

Модель приймає символи як вхідні дані та виводить відповідні необроблені спектрограми, які потім подаються в алгоритм реконструкції Гріффіна-Ліма для синтезу мови.

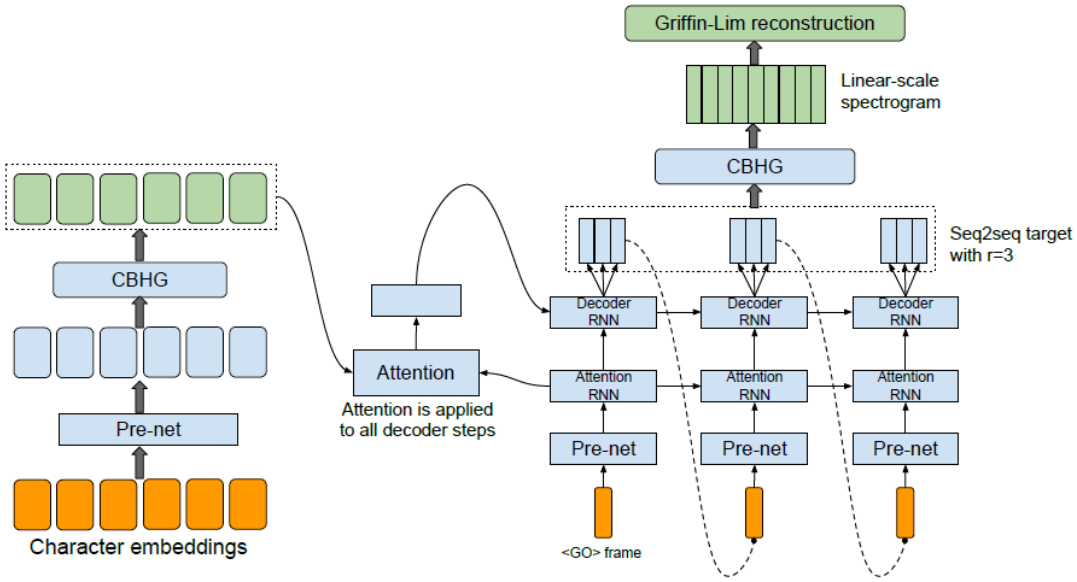


Рисунок 2.2 – Tacatron архітектура

Кодер працює для інтеграції символів вхідного тексту. Результати передаються на рівень Pre-net і CBHG. Pre-Net — це ієрархічна паралельна рекурентна нейронна мережа. Потім кодер Tacotron використовує модуль під назвою CBHG поверх Pre-Net. Модуль отримав свою назву від своїх будівельних блоків: одновимірний згортковий банк (CB), за яким слідує мережа магістралей (H) і двонаправлений GRU (Gated Recurrent Unit - G). Довгі залежності ГРУ вивчає послідовно. Модуль CBHG показано на рисунку 2.3 – CBHG.

Декодер використовує механізм уваги результатів кодера для створення кадрів мел-спектрограм. Декодер також має попередній мережевий рівень, за яким слідує однорівневий GRU, вихід якого поєднується з виходом кодера для отримання векторів контексту за допомогою механізму уваги. Результат GRU потім

комбінується з вектором контексту для отримання вхідного сигналу блоку декодера RNN.

Модуль RNN декодера генерує r mel кадрів спектрограми, а на наступному кроці рівень Pre-net використовує лише останній кадр.

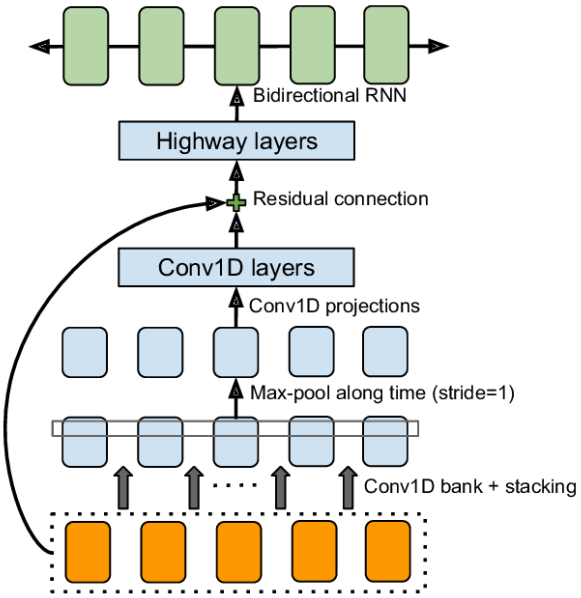


Рисунок 2.3 – CBHG

2.2 Розпізнавання мовлення

Розпізнавання мовлення вторгається в наше життя. Це вже з'являється в телефонах, ігрових консолях і навіть розумних годинниках. Навіть будинки можна автоматизувати за допомогою голосу. Але розпізнавання мовлення існує десятиліттями – тож чому воно стало таким популярним лише зараз? Причина в тому, що глибоке навчання нарешті робить розпізнавання мовлення достатньо точним, щоб працювати поза жорстко контрольованим середовищем. У цьому розділі ми дізнаємося, як використовувати глибоке навчання для розпізнавання мовлення.

Як ви можете здогадатися, ви можете просто передати нейронній мережі записи та навчити її. Але серйозною проблемою є непостійна швидкість мови. Одна людина може дуже швидко сказати таке слово, як «привіт», а інша — дуже повільно,

створюючи довший звуковий файл, який містить більше даних. При цьому обидва звукові файли повинні розпізнаватися як один текст.

Перший крок у розпізнаванні мовлення очевидний - потрібно передати звук на комп'ютер. Звук - це хвиля. Розглянемо, як звукові хвилі перетворити на числа. Ми будемо використовувати аудіофайл з текстом «привіт». Візуалізація цього аудіофайлу показана на рисунку 2.4 – Візуалізація аудіофайлу.

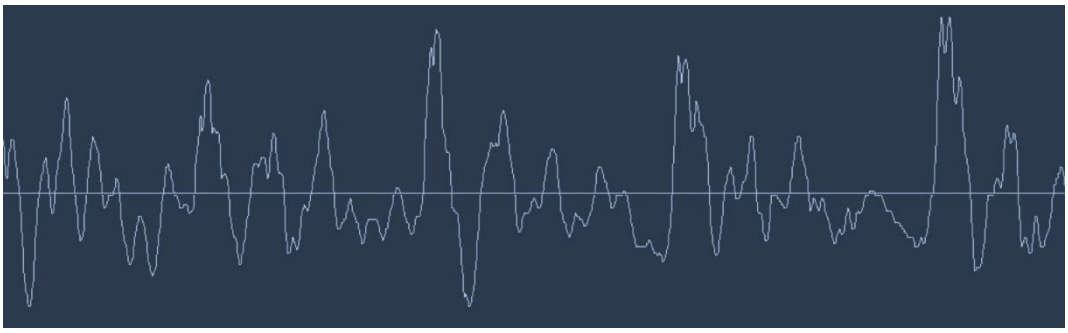


Рисунок 2.4 – Візуалізація аудіофайла

Звукові хвилі одновимірні. У кожен момент вони мають значення, яке залежить від амплітуди хвилі. Збільшимо невелику частину звукової хвилі. Ця частина звукової хвилі показана на рисунку 2.5 – Частина звукової хвилі.

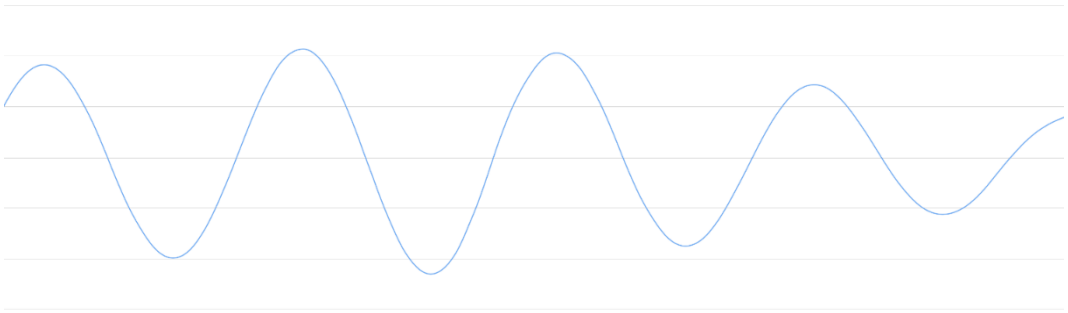


Рисунок 2.5 – Частина звукової хвилі

Щоб перетворити звукові хвилі в числа, просто запишіть значення амплітуди в рівновіддалених точках (Малюнок 2.6 - Значення амплітуди в рівновіддалених точках).

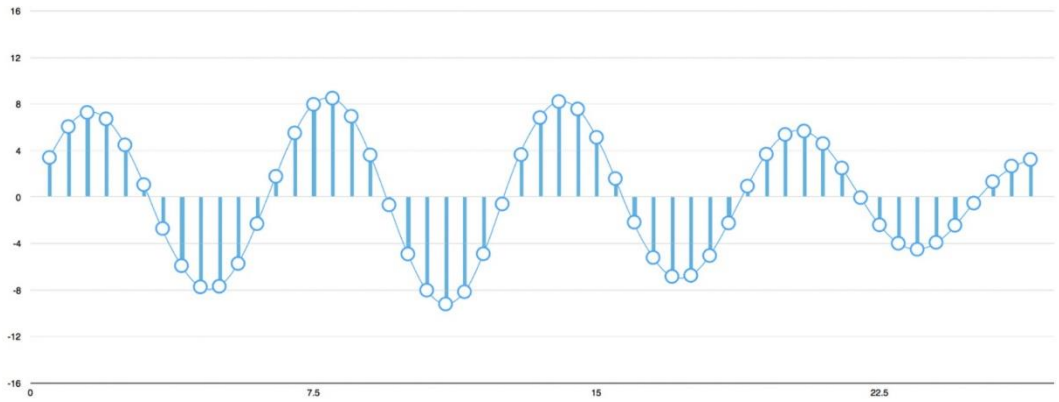


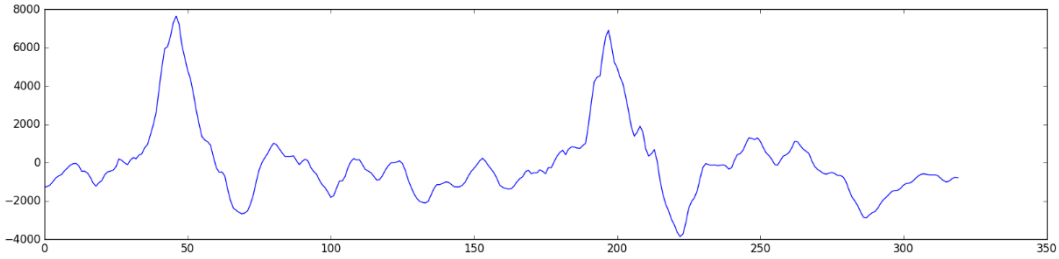
Рисунок 2.6 – Амплітуда хвилі в рівновіддалених точках

Цей процес називається дискретизацією. Дані зчитуються тисячі разів на секунду, і записується число, що відповідає амплітуді звукової хвилі в цей момент. Результатом є стиснутий аудіофайл у форматі .wav.

Аудіо, записане на компакт-диску, має частоту дискретизації 44,1 кГц (44 100 вибірок на секунду). Оскільки частотний діапазон людської мови не дуже широкий, для розпізнавання достатньо частоти дискретизації 16 кГц (16 000 вибірок в секунду).

Вибірка може створити лише приблизну версію вихідної звукової хвилі, оскільки ми зчитуємо випадкові значення, а дані втрачаються між читаннями. Відповідно до теореми Котельникова ми знаємо, що для найкращого відтворення вихідної звукової хвилі достатньо використовувати частоту дискретизації, яка вдвічі перевищує найвищу частоту записаного звуку.

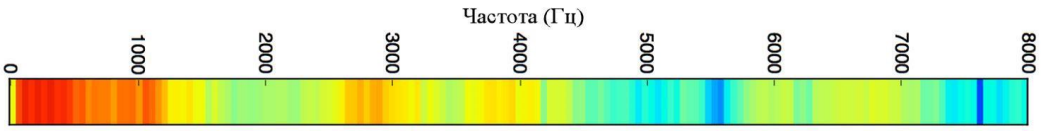
Далі слово «Hello» потрібно оцифрувати 16 000 разів на секунду. У результаті ми матимемо набір чисел, кожне число відображає амплітуду звукових хвиль з інтервалом 1/16000 секунди. Нейронні мережі можна навчити просто на числах, які вони отримують, але визначити лінгвістичні шаблони шляхом безпосередньої обробки цих чисел важко. Натомість ви можете спростити завдання, попередньо обробивши аудіоінформацію. Спочатку ми згрупували показання в сегменти по 20 мс. Побудова цих чисел у вигляді простого лінійного графіка, як показано на рисунку 2.7 – Приблизне зображення звукових хвиль за 20 мілісекунд, забезпечує приблизне зображення вихідної аудіохвилі протягом вибраного періоду 20 мілісекунд.



Рисунку 2.7 – Приблизне зображення звукової хвилі за 20 мілісекунд

Даний ролик триває лише 1/50 секунди. Але це також складна суміш різних звукових частот. Є трохи низькочастотного звуку, є трохи середньочастотного звуку, є трохи високочастотного звуку. Усі ці частоти змішуються разом для створення людської мови. Давайте розіб'ємо складні звукові хвилі на складові частини, починаючи з низьких частот, щоб спростити обробку цих даних нейронною мережею. Потім, додавши звукову потужність для кожної смуги частот, ми створимо частотний графік звуку. Це робиться за допомогою математичної операції, яка називається перетворенням Фур'є. Складні звукові хвилі розбиваються на прості звукові хвилі та складаються з цих простих звукових хвиль. Після отримання окремих звукових хвиль ми додамо звукову потужність кожної звукової хвилі. Тому ми оцінимо важливість кожного діапазону частот (від низьких до високих частот). Як показано на рисунку 2.8

– Частотний графік, що показує звукову потужність кожної смуги частот на 50 Гц в оригінальному кліпі.



Рисунку 2.8 – Діаграма частот

Якщо ми повторимо цей процес для кожного сегмента аудіо тривалістю 20 мілісекунд, ми отримаємо спектрограму. На спектрограмі, показаній на рисунку 2.9 – Спектрограма аудіофайлу, кожен стовпець зліва направо є сегментом 20 мілісекунд.

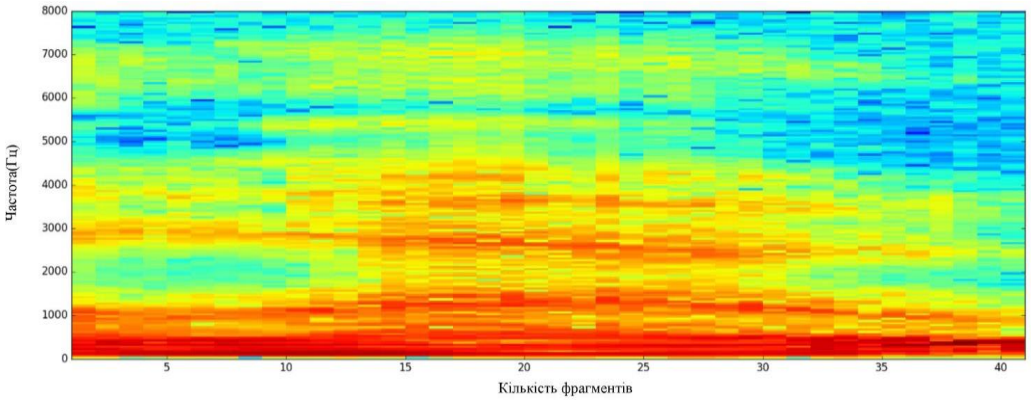


Рисунок 2.9 – Спектрограма аудіофайла

Тепер, коли ми отримали аудіоформат, з яким ми можемо працювати далі, давайте навчимо глибоку нейронну мережу на цих даних. Нейронна мережа отримає аудіозапис тривалістю 20 мс. Для кожного маленького сегмента мережа намагатиметься визначити, яка саме буква була вимовлена. Модель, у якій поточний стан впливає на подальші результати, показано на рисунку 2.10 – Модель із збереженням стану.

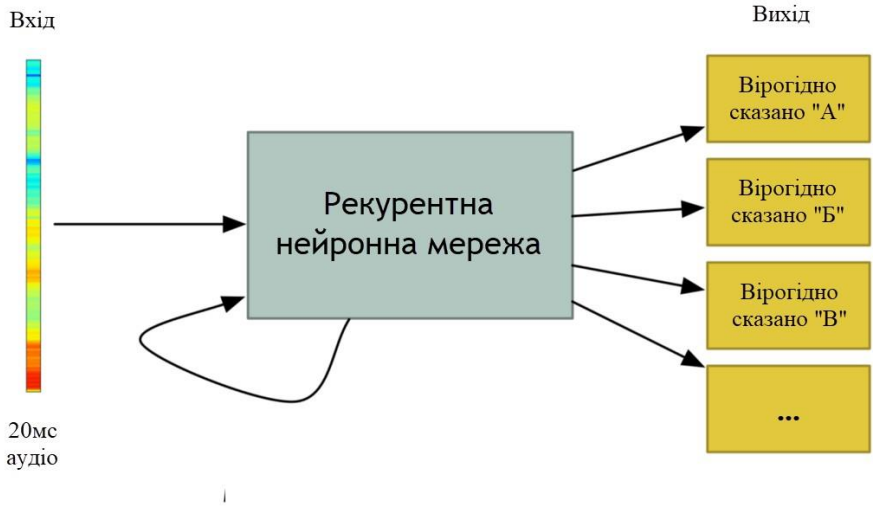


Рисунок 2.10 – Модель зі збереженням стану

Ми будемо використовувати рекурентну нейронну мережу, нейронну мережу, в якій кожен крок враховує результати попередніх кроків. Ми робимо це тому, що

кожна літера, розпізнана мережею, впливає на можливу наступну літеру. Наприклад, якщо ви вимовите фрагмент «Priv», ви, швидше за все, скажете «it», закінчуючи слово «Hello». Навряд чи хтось скаже щось невимовне на зразок «Tcc». Тому, запам’ятовуючи минулі результати, нейронна мережа зможе робити точніші прогнози в майбутньому.

Після обробки всього аудіофайлу (один сегмент за раз) за допомогою нейронної мережі ми ділимо кожен аудіосегмент на літери, які, швидше за все, вимовлятимуться в цьому сегменті. В результаті ми отримаємо практично готові слова, які можуть містити повторювані букви і пропуски. Щоб отримати кінцеве слово, потрібно замінити повторювані символи одиночними і видалити пробіли.

У цьому розділі розглядаються технології синтезу та розпізнавання мовлення. Детально описано методи синтезу мовлення, а також розглянуто алгоритми розпізнавання мовлення на основі рекурентних нейронних мереж.

3 МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ ПОРІВНЯННЯ АУДІОФАЙЛІВ

3.1 Синхронізація аудіофайлів за допомогою динамічних алгоритмів

Перетворення шкали часу

Проблема, яку потрібно вирішити, — порівняти два аудіофайли різної довжини. У нас є два сигнали x і y, які можуть мати однаковий звук, але різну тривалість. Навіть якщо для представлення двох аудіосигналів використовується та сама функція, неможливо підсумувати їх попарні відстані, оскільки сигнали мають різну довжину. Ця проблема називається синхронізацією аудіофайлів.

Для вирішення цієї проблеми використовується алгоритм динамічного перетворення шкали часу (англ. DTW - Dynamic Time Wrapping). Цей алгоритм використовується для вирівнювання двох послідовностей, які мають подібний вміст, але можуть відрізнятися за довжиною.

Вимірювання відстані між двома часовими рядами необхідно для визначення їх подібності та класифікації. Евклідова метрика є одним із таких ефективних показників. Це сума квадратів відстаней від кожної n-ої точки однієї послідовності до n-ої точки іншої послідовності. Однак у використанні цієї метрики є суттєвий недолік: якщо два часові ряди рівні, але один з них трохи зміщений у часі (вздовж осі часу), евклідова метрика може вказувати на те, що ряди відрізняються один від одного. Алгоритм DTW введено, щоб заповнити цю прогалину та забезпечити візуальне вимірювання міжрядкового інтервалу, ігноруючи глобальні та локальні відхилення на часовій шкалі. [2]

Розглянемо два часові ряди - Q довжиною n і C довжиною t.

$$Q = q_1, q_2, \dots, q_i, \dots, q_n$$
$$C = c_1, c_2, \dots, c_j, \dots, c_m \quad (3.1)$$

Перший етап алгоритму полягає в наступному. Необхідно побудувати матрицю d (матрицю відстані) порядку n×m, де елемент dij є відстанню d(qi,cj) між рядком і

точкою Cj. У більшості випадків використовується евклідова відстань: d(qi, cj) = (qi - cj)^2. Кожен елемент (i, i) матриці відповідає вирівнюванню між точками qi і cj.

На другому етапі ми будуємо матрицю перетворення, де кожен елемент обчислюється відповідно до наступного співвідношення:

$$D_{i\,j} = d_{i\,j} + min(D_{i-1\,j}, D_{i-1\,j-1}, D_{i\,j-1}) \tag{3.2}$$

Після заповнення матриці перетворення ми переходимо до завершального етапу, який передбачає побудову оптимальних шляхів перетворення та відстаней (вартості шляху).

W — шлях перетворення, представлений набором суміжних елементів матриці, що встановлює відповідність між Q і C. Це шлях, який мінімізує загальну відстань між Q і C. Це шлях, який мінімізує загальну відстань між Q і C. Формула обчислення -го елемента шляху W виглядає наступним чином:

$$w_k = (i,j)_k, \;\; d(w_k) = d(q_i, c_j) = (q_i - c_j)^2 \tag{3.3}$$

Отже,

$$W = w_1, w_2, \dots, w_k, \dots, w_K; \quad max(m,n) \leq K < m+n, \tag{3.4}$$

де W – довжина заданого шляху.

Шлях конверсії має відповідати таким умовам:

- Початкова точка шляху w1 = (1, 1), а кінцева точка wk = (m,n). Ця умова стверджує, що шлях трансформації містить усі точки обох часових рядів.

- Будь-які два суміжні елементи шляху W, wk = (wi,wj) і wk+1 = (wi+1,wj+1), задовольняють: wi - wi+1 <= 1, wi – wj+ 1 < = 1. Це обмеження гарантує, що шлях трансформації рухається поступово. Тобто індекси i та j можуть збільшуватися лише на 1 на кожному кроці шляху.

- Будь-які два суміжні елементи шляху W, wk = (wi,wj) і wk-1 = (wi-1,wj-1), такі що: wi – wi-1 => 0, wi – wj-1 => 0 . Це обмеження гарантує, що шлях трансформації не повертається до минулої точки. Тобто показники ступеня і та та і обидва залишаються незмінними або збільшуються, але ніколи не зменшуються.

Хоча існує велика кількість шляхів перетворення, які задовольняють усі перераховані вище умови, необхідний лише шлях, який мінімізує відстань DTW (вартість шляху).

Відстань DTW (вартість шляху) між двома послідовностями обчислюється на основі оптимального шляху перетворення за такою формулою:

$$DTW(Q,C) = min\left\{\frac{\sum_{k=1}^{K} d(w_k)}{K}\right\} \quad (3.5)$$

У знаменнику використовується для врахування того факту, що шляхи перетворення можуть мати різну довжину.

Просторова та часова складність цього алгоритму є квадратичною O(nm), тому що алгоритм DTW повинен вивчати кожну клітинку матриці перетворення.

3.2 Перетворення Фур’є для обробки аудіосигналів

Як відомо, звуковий сигнал в комп'ютері можна представити у вигляді його амплітудних показань, які створюються через певний проміжок часу (період дискретизації) і представлені певною кількістю двійкових цифр (бітрейт дискретизації). Це представлення зручно для зберігання звукових сигналів і перетворення їх назад у безперервні сигнали. Однак деякі операції обробки звуку в цьому поданні не завжди зручні. Це пояснюється тим, що справжні звукові сигнали складаються з частот з певними амплітудами та фазами. Таким чином, використання операцій обробки аудіосигналу, таких як «фільтр низьких частот» або «фільтр високих частот», вимагає перетворення подання аудіосигналу як еталонного в його частотному спектрі. Тоді перетворення звукового сигналу буде проявлятися у вигляді коефіцієнтів, що відповідають амплітуді і фазі частот, що утворюють сигнал. Наприклад, операція фільтра низьких частот, яка усуває всі частоти вище даного сигналу, тепер може просто обнулити коефіцієнти, що відповідають частотам, які потрібно видалити. Насправді сфери застосування цього перетворення набагато ширші: обробка растрових зображень, телекомунікації, дослідження та вимірювання сигналів, радіолокація тощо. Прикладом застосування перетворення є передача даних

у цифровій формі через аналогові лінії телефонної мережі (модем). Щоб передати дані в цифровій формі, вони спочатку перетворюються на певний набір частот і передаються по лінії передачі, а потім на кінці приймача,

Фундаментально трансформувати та відновлювати вихідні дані. Далі розглянемо основні поняття, необхідні для розуміння операції перетворення Фур’є. [3]

Ряд Фур'є — це нескінченна математична послідовність, що складається з коефіцієнтів функцій синуса і косинуса виду

$$\frac{a_0}{2} + \sum_{m=1}^{\infty} \left(a_m \cos\frac{m\pi x}{l} + b_m \sin\frac{m\pi x}{l} \right) \tag{3.6}$$

Відомо, що якщо періодична функція з періодом 2l на інтервалі [-l, l] задовольняє умови першого типу (умова Діріхле), тобто вона неперервна і має кінцеве число екстремумів і точок розриву , то цю функцію можна виразити як суму рядів Фур’є. Для визначення коефіцієнтів ряду Фур'є використовується наступна формула

$$a_m = \frac{1}{l} \int_{-l}^{l} f(x) \cos\frac{m\pi x}{l} dx,$$

$$a_m = \frac{1}{l} \int_{-l}^{l} f(x) \sin\frac{m\pi x}{l} dx \tag{3.7}$$

Якщо функція розкладання парна, тобто f(-x) = f(x) [4], ряд Фур’є складається лише з косинусів. Це означає, що всі коефіцієнти синуса дорівнюють нулю. Якщо функція непарна, тобто f(-x) = -f(x) [4], ряд Фур’є складається лише з синусів. Це означає, що всі коефіцієнти косинуса дорівнюють нулю. Загалом, коефіцієнти синуса і косинуса не дорівнюють нулю. Тому будь-яку періодичну функцію, що задовольняє умовам першого типу, можна розкласти в ряд Фур’є, виразивши функцію через суму косинусів і синусів.

Припустимо, що період повторення досліджуваної функції нескінченний. Тобто функція не є періодичною. У цьому випадку існує інтегральна формула Фур’є, отримана з ряду Фур’є періодичної функції періоду 2l на граничному переході:

$$f(x) = \frac{1}{\pi}\int_{0}^{+\infty} dz \int_{-\infty}^{+\infty} f(u)\cos z(u-x)du \tag{3.8}$$

Інтегральне перетворення Фур'є пов'язане з такою формулою:

$$F(z) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{+\infty} e^{izx} f(x)dx$$

$$f(x) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{+\infty} e^{-izx} F(z)dz \tag{3.9}$$

З цих формул ми можемо вивести синусне перетворення Фур'є непарних функцій:

$$f_s(z) = \sqrt{\frac{2}{\pi}}\int_{0}^{+\infty} f(x)\sin zx\,dx$$

$$f(x) = \sqrt{\frac{2}{\pi}}\int_{0}^{+\infty} f_s(z)\sin zx\,dx \tag{3.10}$$

Тепер виведемо косинусне перетворення Фур'є парної функції:

$$f_c(z) = \sqrt{\frac{2}{\pi}}\int_{0}^{+\infty} f(x)\cos zx\,dx$$

$$f(x) = \sqrt{\frac{2}{\pi}}\int_{0}^{+\infty} f_c(z)\sin zx\,dx \tag{3.11}$$

Отже, безперервне перетворення Фур'є дозволяє представити неперіодичну функцію у вигляді функціонального інтеграла, який представляє коефіцієнти ряду Фур'є неперіодичної функції в кожній її точці.

У дискретному перетворенні Фур'є (DFT — дискретне перетворення Фур'є) досліджувана функція є періодичною, має скінченний повторюваний період і є дискретною. Для визначення амплітуди та фази частотних компонентів сигналу в ДПФ використовуються співвідношення до базисних функцій синуса та косинуса.

Спектр у DFT визначається амплітудами синусів і косинусів у досліджуваному зразку при частоті повторення від 0 до N/2, де N — кількість елементів зразка. Перетворення Фур’є розкладає дискретизований сигнал із N вибірок на N/2+1 синусоїдних компонентів і N/2+1 косинусних компонентів. Це призводить до наступної формули DFT:

$$ReX[k] = \sum_{i=0}^{N-1} x[i] \cos\left(\frac{2\pi k i}{N}\right)$$
$$ImX[k] = -\sum_{i=0}^{N-1} x[i] \sin\left(\frac{2\pi k i}{N}\right) \quad (3.12)$$

Ці формули описують пряме перетворення Фур'є. ReX[k] — це масив, що містить значення косинусних компонентів, а ImX[k] — це масив синусних компонентів.

Досліджуваний сигнал розкладається на його спектр за наведеною вище формулою.

Після розкладання сигналу на синусну і косинусну складові можна отримати амплітудні і фазові значення частотних складових сигналу. Щоб перетворити пару відповідних коефіцієнтів синуса і косинуса в амплітуду і фазу частотної складової, ми використовуємо таку формулу:

$$MagX[k] = (ReX[k]^2 + ImX[k]^2)^{1/2}$$
$$PhaseX[k] = arctan\left(\frac{ImX[k]}{ReX[k]}\right) \quad (3.13)$$

Отримані значення амплітуди і фази називаються полярними представленнями. Значення коефіцієнтів синуса і косинуса характеризують прямокутне зображення. Значення фази сигналу несе інформацію про форму та зміни сигналу.

Швидке перетворення Фур’є (ШПФ — швидке перетворення Фур’є) — це алгоритм, який часто використовується в цифровій обробці сигналів під час перетворення дискретних даних із представлення часу в частотну область. Швидке перетворення Фур'є — метод реалізації перетворення Фур'є. Алгоритм БПФ обчислює дискретне перетворення Фур’є (ДПФ – дискретне перетворення Фур’є) або

зворотне дискретне перетворення Фур’є (IDFT – зворотне дискретне перетворення Фур’є). Оскільки ДПФ надто повільно обчислюється, часто використовується БПФ, яке швидко обчислює такі перетворення шляхом розкладання матриці ДПФ на добутки розсіяних (переважно нульових) факторів. Таким чином, алгоритм ШПФ вдається зменшити складність ДПФ від O(n^2) до O(nlogn), де n – це розмір даних. Різниця у швидкості може бути значною, особливо для великих наборів даних, де N може бути тисячами або мільйонами.

3.3 Середньоквадратична помилка

Після перетворення Фур'є отримують набір характерних частот двох звукових сигналів. Для визначення відносної різниці між ними використовується метод визначення середньої квадратичної похибки.

Середня квадратична похибка визначається за такою формулою:

$$E = \frac{1}{n}\sum_{i=1}^{n}(MagA_i - MagB_i)^2, \tag{3.14}$$

де MagAi — амплітуда частоти i в сигналі A, а MagBi — амплітуда частоти i в сигналі B.

Під час визначення середньоквадратичної похибки відмінності у компонентах фази ігноруються, оскільки вони не несуть корисної інформації.

У цьому розділі розглядається математичне забезпечення реалізації алгоритму порівняння аудіофайлів. Розглянуто алгоритми динамічного перетворення шкали часу для синхронізації аудіофайлів, перетворення Фур'є для обробки аудіосигналів. Крім того, описано методи обчислення середньої квадратичної помилки для визначення відносних відмінностей між аудіофайлами.

4 ОПИС АРХІТЕКТУРИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ РОЗПІЗНАВАННЯ МОВЛЕННЯ

4.1 Архітектура програмного забезпечення

Мова програмування Java була обрана для розробки веб-додатків через її такі особливості:

- Простота – Java має чисті, тісно пов’язані та прості у вивченні мовні засоби;
- Безпека – Java є надійним інструментом для створення веб-додатків;
- Портативність – програми Java можуть працювати в будь-якому середовищі з системою виконання Java;
- Суть об'єктно-орієнтованого програмування - Java втілює сучасну філософію об'єктно-орієнтованого програмування;
- Надійність – Java знижує ймовірність програмних помилок завдяки суворому набору змінних і реалізації відповідних елементів керування під час виконання;
- Багатопотоковість – Java забезпечує інтегровану підтримку багатопотокового програмування;
- Незалежність від архітектури - Java не залежить від конкретного типу комп'ютера або архітектури операційної системи;
- Можливість інтерпретації – Java надає байт-код, який забезпечує незалежність від платформи;
- Висока продуктивність – байт-код Java оптимізований для продуктивності;
- Розповсюдження - Java була розроблена з її застосуванням у розподіленому Інтернет-середовищі;
- Динамічні - програми Java містять велику кількість інформації, яка використовується під час виконання для перевірки та надання доступу до об'єктів. [5]

Програма побудована на основі архітектурного шаблону MVC. Блок-схема шаблону MVC показана на рисунку 4.1 – Структура шаблону MVC.

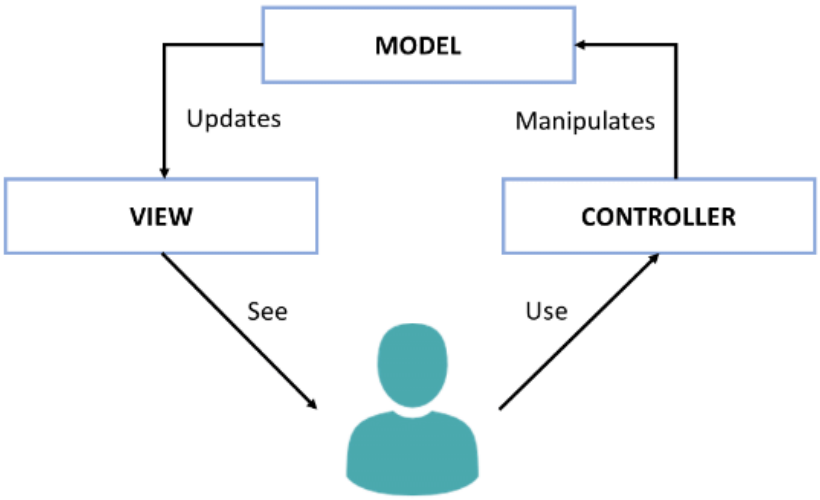


Рисунок 4.1 – Структура шаблону MVC

MVC — це шаблон проектування програмного забезпечення, який зазвичай використовується в дизайні інтерфейсу користувача, який розділяє відповідну логіку програми на три взаємопов'язані елементи. Це робиться для того, щоб відокремити внутрішнє представлення інформації від того, як користувачі її подають і отримують. [6]

Модель є основним компонентом шаблону MVC. Це динамічна структура даних програмного забезпечення, яка не залежить від інтерфейсу користувача. Цей компонент безпосередньо керує даними, логікою та програмними правилами. Ця модель відповідає за керування даними програми. Цей компонент отримує дані користувача від контролера.

View — це компонент шаблону MVC, який відповідає за представлення об'єктів у програмному забезпеченні та реагування на зміни в моделі. Відображення означає відображення моделі в певному форматі.

Контролер є компонентом шаблону MVC, який оновлює як модель, так і представлення. Він приймає введені користувачем дані та перетворює їх на команди для моделі або подання. Контролери реагують на введення користувачами та взаємодіють з об'єктами моделі даних. Контролер отримує вхідні дані, додатково перевіряє їх, а потім передає вхідні дані в модель.

Spring Framework — це фреймворк загального призначення, який забезпечує повну модель розробки та налаштування для сучасних бізнес-додатків, написаних мовою програмування Java. Ключовим елементом Spring є підтримка інфраструктури на рівні програми, тобто розробники можуть зосередитися на бізнес-логіці програми без необхідності додаткового налаштування на основі середовища виконання [7].

Основні функції пружини:

- Ввести залежності;
- Аспектно-орієнтоване програмування, включаючи декларативне управління транзакціями;
- Створення додатків Spring MVC;
- Spring можна розглядати як набір інших менших фреймворків або фреймворк у фреймворку. Більшість із цих фреймворків можуть працювати незалежно, але при спільному використанні ці фреймворки можуть надати більше функціональних можливостей.
- Інверсія керування контейнером: конфігурація компонентів програми та керування життєвим циклом об’єкта Java;
- Аспектно-орієнтована система програмування. Фреймворк допомагає працювати з функціями, які неможливо без втрат включити в функції об’єктно-орієнтованого програмування Java;
- Структура доступу до даних. Фреймворк використовує інструменти JDBC і ORM у поєднанні з системами керування базами даних на платформі Java, щоб забезпечити вирішення великої кількості повторюваних проблем у середовищах Java.
- Структура керування транзакціями: координує різні API керування транзакціями та інструменти конфігурації керування транзакціями для об’єктів Java;
- Фреймворк MVC. Фреймворк підтримує шаблон архітектури MVC і реалізує всі основні функції ядра Spring. Детальне зображення схеми взаємодії компонентів Spring MVC показано на рисунку 2.6 – Схема взаємодії компонентів Spring MVC.

– Структура автентифікації та авторизації: надає інструменти для реалізації та налаштування процесів автентифікації та авторизації, підтримуючи багато стандартних протоколів, інструментів і практик через підпроект Spring Security. [7].

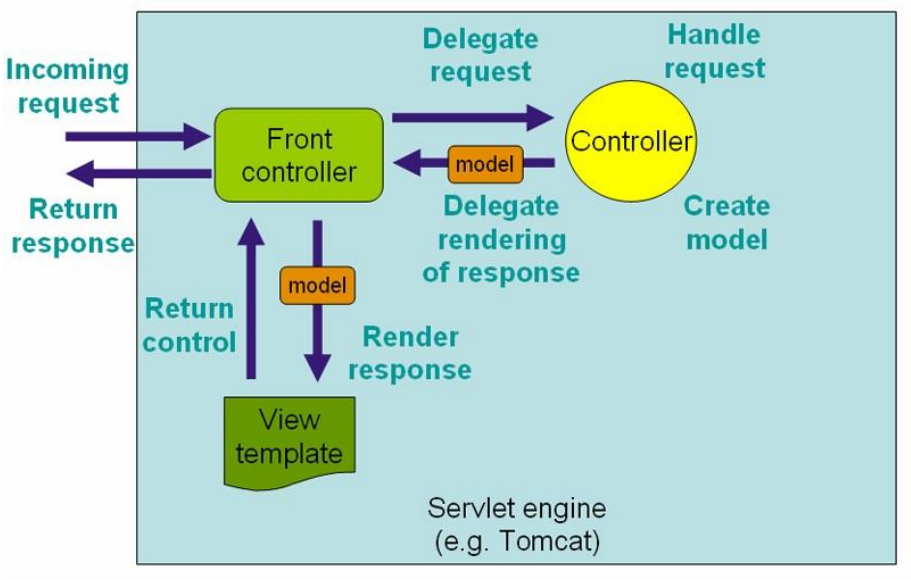


Рисунок 4.2 – Схема взаємодії Spring MVC

Hibernate — це бібліотека Java, призначена для вирішення проблем об'єктно-реляційного відображення (ORM) [8].

Мета Hibernate — полегшити розробникам виконання великої кількості низькорівневого програмування під час роботи з об'єктно-орієнтованими програмами в реляційних базах даних. Як правило, розробники використовують Hibernate не тільки для спрощення процесу проектування систем класів і таблиць з самого початку, але й для використання існуючих баз даних [8].

Hibernate не тільки вирішує зв'язок між класами Java і таблицями в базі даних (і типами даних Java – і типами даних SQL), але й дозволяє автоматизувати процес створення й оновлення таблиць, запитів до бази даних і обробки результатів цих . Довідка про використання спеціальних коштів. Крім того, фреймворк допомагає значно скоротити час розробки програми, який програмісти витрачають на написання коду SQL і JDBC. Hibernate автоматично генерує запити SQL і звільняє програмістів

від ручного керування наборами даних результатів і перетвореннями об’єктів, що значно спрощує перенесення програм до будь-якої бази даних SQL [8].

Виберіть PostgreSQL як базу даних. PostgreSQL — безкоштовна розподілена об’єктно-реляційна система керування базами даних (ORDBMS), найбільш розвинена відкрита СУБД у світі та справжня альтернатива комерційним базам даних [9]. Він підтримується всіма сучасними системами Unix, включаючи найпоширеніші, такі як Linux, FreeBSD, NetBSD, OpenBSD, SunOS, Solaris, DUX і Mac OS X. Починаючи з версії 8.X, PostgreSQL працює в «власному» режимі під MS Windows.

4.2 Конструювання програмного забезпечення

Як зазначалося вище, архітектура розробленої системи базується на моделі MVC. Репозиторій Spring Data знаходиться на рівні доступу до даних. Вони споживаються службами на рівні бізнес-логіки, які, у свою чергу, споживаються контролерами даних, які обробляють запити користувачів і відображають їх як веб-сторінки.

Об’єкти, які ми використовуємо, є сутностями, які мають пряме представлення в базі даних. На графічному матеріалі подано структурну схему бази даних.

Діаграма класів моделі, що відповідає сутностям у базі даних, показана на рисунку 4.4 – Діаграма структури класів моделі.

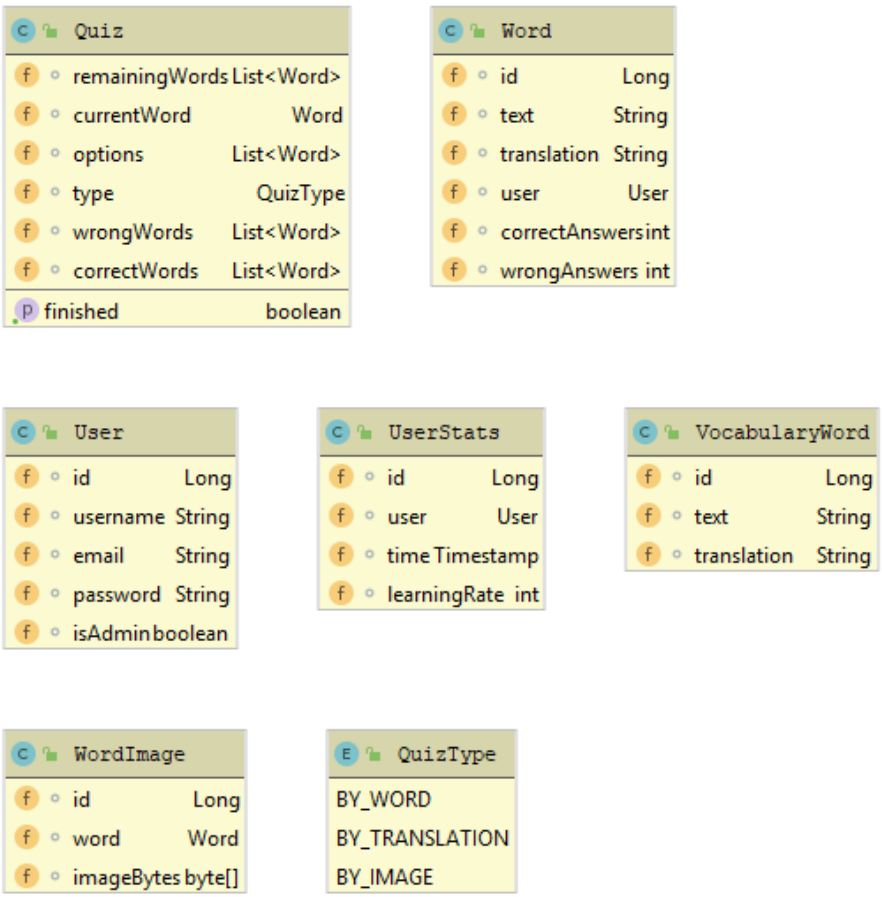


Рисунок 4.3 – Схема структурна класів моделей

Бізнес-логіка повністю реалізована в сервісному класі. Структурна схема інтерфейсу сервісу та його реалізація показана на рисунку 4.5 – структурна схема класу сервісу.

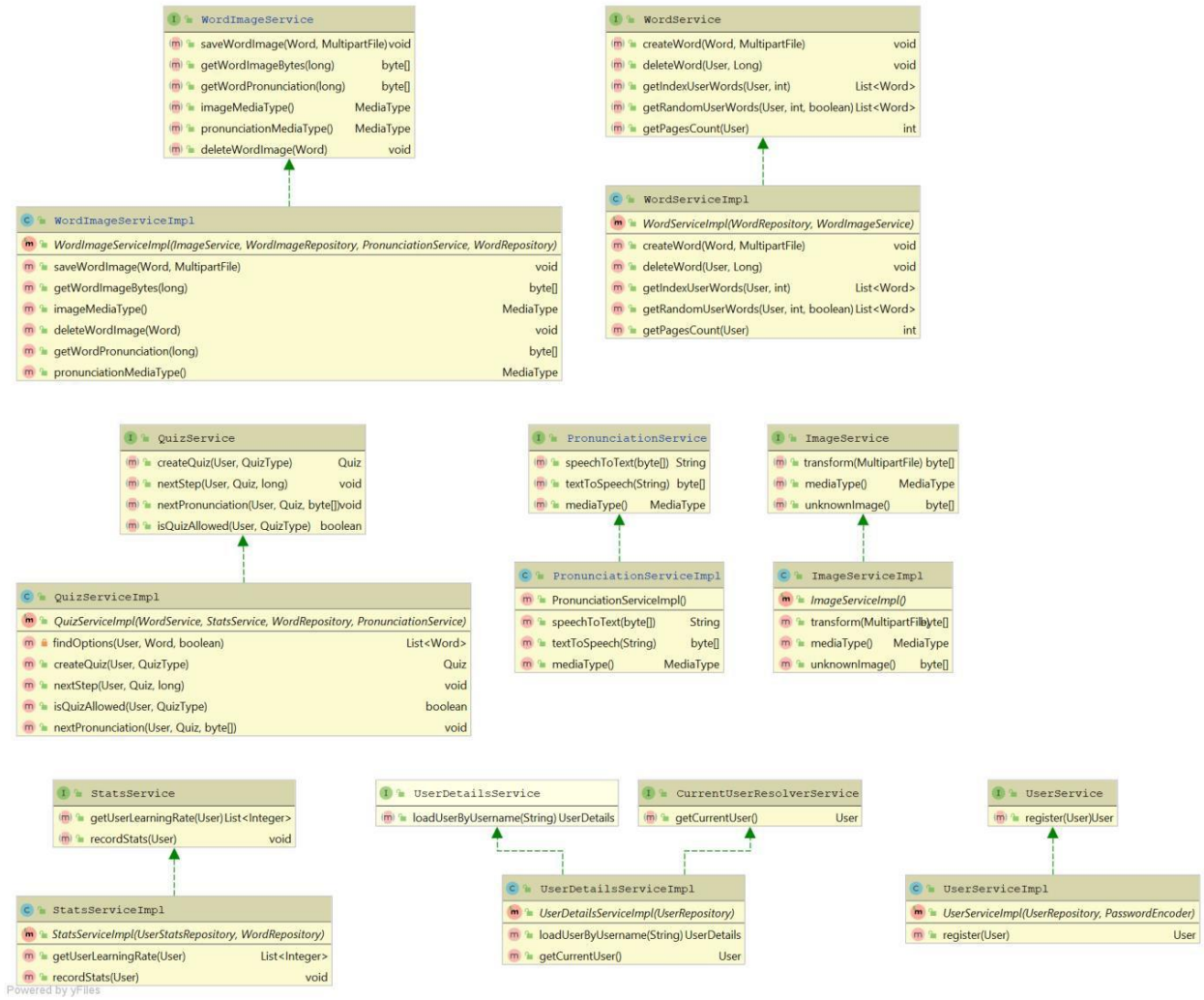


Рисунок 4.4 – Схема структурна діаграми класів Сервісів

4.3 Програмний засіб для реалізації розробленої методики визначення правильності вимови

Для реалізації розроблених методів використовується бібліотека librosa.

librosa — це бібліотека Python для аналізу музики та аудіо. Він забезпечує будівельні блоки, необхідні для створення систем пошуку звукової інформації. Хоча сама бібліотека призначена для пошуку аудіоінформації, її можна використовувати в алгоритмах, розроблених для порівняння аудіофайлів, оскільки вона містить необхідні методи для реалізації алгоритмів порівняння аудіофайлів.

4.4 Способи взаємодії між процесами

Оскільки серверна частина програми розроблена мовою програмування Java, а алгоритм порівняння аудіофайлів розроблено мовою програмування Python, необхідно реалізувати міжпроцесну комунікацію (IPC).

Різні підходи IPC були адаптовані до різних вимог програмного забезпечення, таких як продуктивність, модульність і системні міркування, такі як пропускна здатність мережі та затримка. Основні методи наведені в таблиці 4.1.

Таблиця 4.1 – Підходи до IPC

Метод	Опис
Файл	Запис, що зберігається на диску, або запис, синтезований на вимогу файловим сервером, до якого можна отримати доступ за допомогою декількох
Файл зв'язку	Унікальна форма IPC наприкінці 1960-х, яка найбільше нагадує протокол 9P плану 9
Сигнал	Системне повідомлення, надіслане з одного процесу в інший, зазвичай не використовується для передачі даних, а замість цього використовується для віддаленого керування процесом, що складається з партнерів.
Сокет	Дані надсилаються через мережевий інтерфейс або до іншого процесу на тому самому комп'ютері, або до іншого комп'ютера в мережі.
Unix сокет	Подібно до інтернет-сокета, але все спілкування відбувається всередині ядра. Сокети домену використовують файлову систему як свій адресний простір. Процеси посилаються на сокет домену як на анод, і кілька процесів можуть взаємодіяти з одним

Черга повідомлень	Потік даних, схожий на сокет, але який зазвичай зберігає межі повідомлень. Зазвичай реалізовані операційною системою, вони дозволяють безлічі процесів читати та записувати в чергу повідомлень, не будучи безпосередньо підключеними один до одного.
Анонімний ріре	Односпрямований канал даних, що використовує стандартний вхід і вихід. Дані, записані на кінець запису конвеєра, буферизуються операційною системою, доки вони не будуть зчитуватися з кінця конвеєра конвеєра. Двосторонній зв'язок між процесами можна досягти, використовуючи дві труби в
Іменований ріре	Ріре, який обробляється як файл. Замість того, щоб використовувати стандартні вхідні та вихідні дані, як для анонімного, процеси записують і читають із іменованого, ніби це звичайний файл.
Спільна пам'ять	Кілька процесів отримують доступ до одного блоку пам'яті, що створює спільний буфер для взаємодії
Передача повідомлення	Дозволяє декільком програмам спілкуватися за допомогою черг повідомлень та / або каналів, якими не керує ОС. Зазвичай використовується в моделях
Файл із трансляцією пам'яті	Файл, зіставлений з оперативною пам'яттю, може бути змінений шляхом безпосередньої зміни адрес пам'яті замість виведення в потік. Це надає ті ж переваги, що і стандартний файл.

Для досягнення взаємодії між процесами був обраний метод використання файлів і читання виводу підпроцесу (конвеєр).

У цьому розділі детально обговорюються техніки, інструменти та архітектурні шаблони, що використовуються в розробленій системі. Наведено та описано схему бази даних, а також структурну діаграму діаграми інтерфейсу та структурну діаграму класів, які реалізують інтерфейс. Також розглядаються способи взаємодії між процесами.

5. АНАЛІЗ РЕАЛІЗАЦІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1. Огляд архітектури програмного забезпечення

Як архітектурний підхід до розробки інтегрованих голосових помічників навколишнього середовища ми обрали мікросервісний підхід. За допомогою мікросервісів система розбивається на невеликі функціональні компоненти, незалежні один від одного. Порівняно з традиційними монолітними архітектурами (де система складається з одного великого модуля), мікросервіси відокремлені та працюють разом для виконання одного завдання.

- В результаті використання такого підходу система отримає наступні переваги:
- Швидко розгортайте та змінюйте певні служби. Крім того, ці процеси стають більш керованими та передбачуваними порівняно з традиційними методами.
- З точки зору розробки продуктів, які використовують моделі машинного навчання, важливою перевагою є те, що кожен мікросервіс можна написати за допомогою технології, яка найкраще підходить для вирішення завдання, за яке він відповідає. При створенні голосових помічників для вирішення завдань розпізнавання та обробки тексту рекомендується використовувати бібліотеки та мову програмування Python, але для реєстрації інформації більше підходить стек ELK, написаний на мові програмування Java.
- Кожну службу можна масштабувати горизонтально, що покращує швидкість і стабільність системи, оскільки система може нормально функціонувати, навіть якщо певний екземпляр служби виходить з ладу.
- Оскільки служби не тісно пов’язані між собою та взаємодіють через інтерфейси API, реалізації послуг можна замінити незалежно від інших реалізацій послуг.

Архітектура розробленого інтегрованого середовища розробки системи голосового помічника показана на рисунку 1. 12 склад

7 сервісів: Client - плагін для певного середовища розробки програмного забезпечення (Client), сервіс, який розпізнає мову користувача та перетворює його в текст (Speech Recognition service), сервіс розпізнавання іменованих сутностей, сервіс

журналювання (Logging service), і служба конфігурації (Configuration service), служба виявлення служби (Discovery service) і шлюз API (API Gateway).

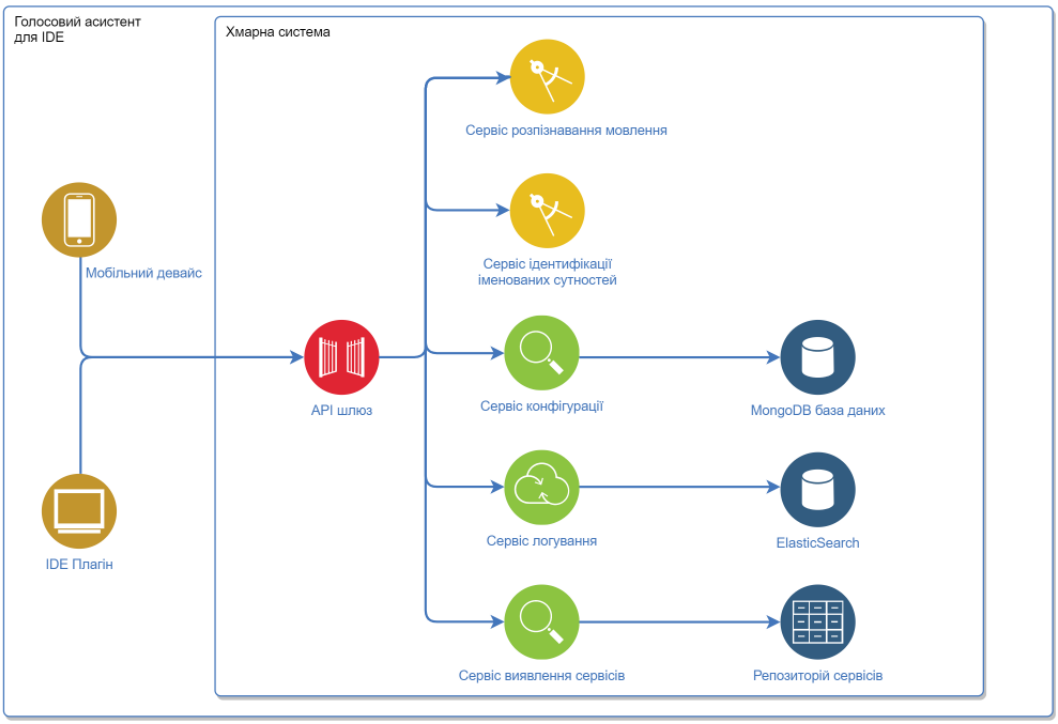


Рис. 5.1. Архітектура створеної системи

Після аналізу даної архітектури можна побачити, що користувач безпосередньо взаємодіє з клієнтом (мобільний пристрій або плагін IDE), який, у свою чергу, спілкується з хмарним сервером, який виконує всі важкі обчислення, такі як розпізнавання команд користувача та ідентифікація програми. текст Іменовані сутності. Наявність кількох екземплярів кожної служби гарантує, що система стійка до помилок, тому що якщо одна зі служб виходить з ладу, інша займе її місце. Так само, завдяки встановленій архітектурі системи, щоб додати підтримку нового інтегрованого середовища розробки програмного забезпечення або нової мови програмування, достатньо створити нового клієнта, відповідального за виконання ідентифікованих команд користувача.

ElasticSearch використовується для забезпечення журналювання, а база даних MongoDB використовується для зберігання конфігурації. У свою чергу, служби

пошуку служб використовують репозиторії, які містять інформацію про активні та доступні служби.

5.1.1. Клієнт

При розробці програмного забезпечення даної магістерської роботи клієнтами виступають обране інтегроване середовище розробки (IntelliJ IDEA) та плагін для мови програмування Java. Однак через логічне розділення клієнтської та серверної частин клієнтами можуть виступати мобільні пристрої або плагіни, розроблені для інших середовищ і мов програмування. Зокрема, ви можете створювати плагіни для інтегрованих середовищ розробки програмного забезпечення та текстових редакторів, таких як Eclipse, NetBeans, PyCharm, Sublime, Microsoft Visual Code, CLion тощо.

IntelliJ IDEA — це інтегроване середовище розробки програмного рішення, створене JetBrains, яке підтримує велику кількість мов програмування, а саме Python, Java, JavaScript. Середовище дозволяє розробникам швидко реорганізовувати вихідний текст програми та має широкі інструменти рефакторингу. Візуальний інтерфейс системи спрямований на підвищення продуктивності розробників, дозволяючи їм зосередитися на розробці функцій. Продукт доступний у безкоштовній версії зі зменшеними можливостями (Community) і преміум-версії (Ultimate). Це середовище надає можливість розширити свою функціональність шляхом створення спеціальних плагінів. Саме таким чином розроблений голосовий помічник інтегрується в обране середовище за допомогою замовника.

При створенні плагінів для IntelliJ IDEA використовується мова програмування Java та SDK обраного інтегрованого середовища розробки ПЗ. Комплект розробки програмного забезпечення (SDK) — це набір інструментів, необхідних для розробки додатків або плагінів для існуючої рамки чи платформи. Наприклад, для розробки програмного забезпечення з використанням Java потрібен Java SDK (JDK). SDK містить двійкові файли, вихідні коди програм для двійкових файлів і документацію для вихідних кодів програм. SDK зазвичай є глобальними. Це означає, що один SDK можна використовувати в кількох проектах і модулях.

Клієнт відповідає за отримання голосових команд користувача, взаємодію з інтегрованим середовищем розробки, створення доступних іменованих об’єктів, присутніх у контексті проекту, і надсилання їх на віддалений сервер через протокол HTTP, використовуючи архітектурний стиль REST для виконання ресурсомістких обчислювальні операції. Голосові команди користувача надсилаються на хмарний сервер. Команди користувача записуються за допомогою Java Sound API.

Після отримання відповіді від сервера клієнт знову взаємодіє з інтегрованим середовищем SDK і виконує розпізнану команду користувача. Система дозволяє користувачам використовувати мовлення для створення та модифікації змінних, методів, файлів, класів і виконання основних операцій, використовуючи їх та інтегроване середовище розробки програмного забезпечення за власним вибором. Описаний алгоритм взаємодії між клієнтом системи та користувачем та інтегрованим середовищем показаний на діаграмі послідовності, зображеній на рисунку 1. 5.2.

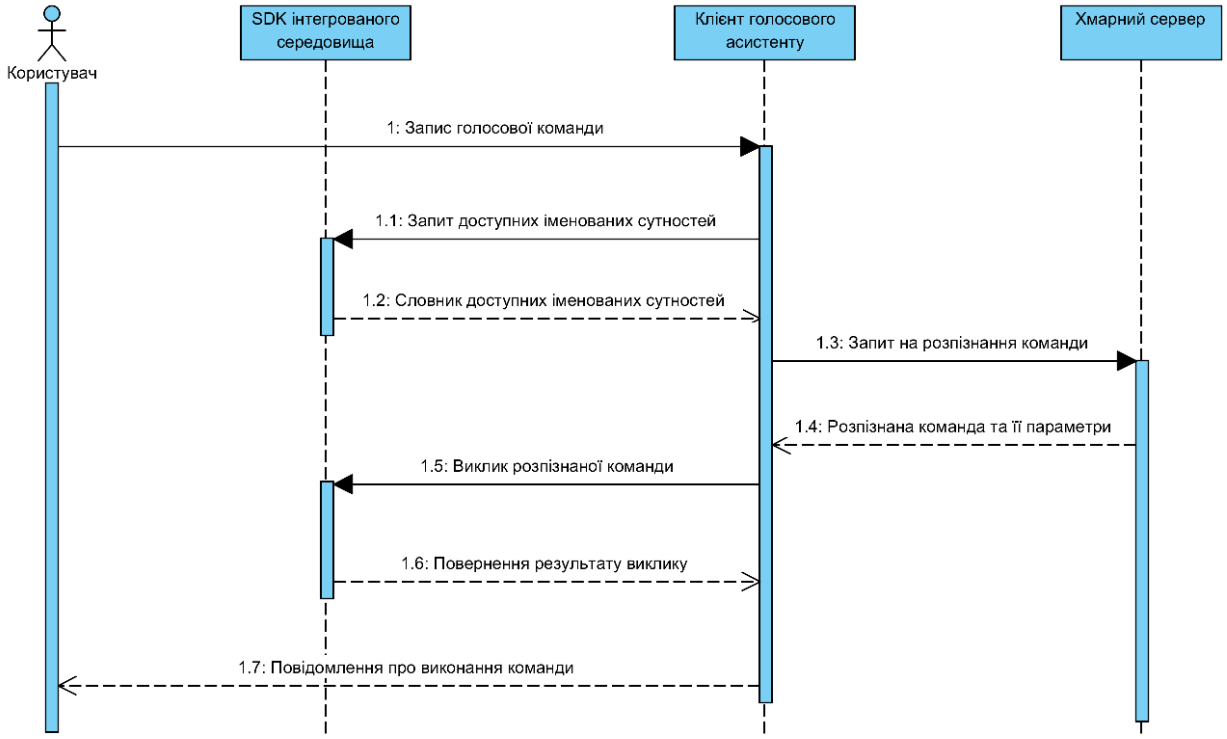


Рис. 5.2. Обробка голосової команди користувача

Уточнення розпізнаних команд користувача також відбувається на стороні клієнта за допомогою словника існуючих іменованих сутностей і за допомогою

алгоритму модифікації, наведеного в розділі 3.2. Це дозволяє уникнути зайвого навантаження на мережу та вирішити питання конфіденційності, оскільки вся інформація про текст програми користувача залишається на його локальному комп'ютері та не передається зовнішнім службам.

5.1.2. API шлюз

Шлюз API відповідає за агрегацію отриманих даних і маршрутизацію запитів до системних служб. Він діє як проміжний рівень між мікросервісами та системними клієнтами. Будь-які запити від користувачів або інших систем спочатку надсилаються до API шлюзу, який потім перенаправляє їх до кількох або до одного конкретного мікросервісу. Результати, отримані від сервісу, агрегуються та надсилаються клієнту як відповідь.

Теоретично клієнт може напряму надсилати запити до будь-якого мікросервісу, але цей метод внесе певні обмеження та проблеми в архітектуру системи. Однією з проблем, яка може виникнути, є невідповідність між інформацією, яку надає кожен мікросервіс, і тим, що потрібно клієнту. Залежно від розміру системи, щоб отримати необхідні дані, які цікавлять клієнта, йому потрібно буде здійснити багато дзвінків в різні мікросервіси. У цьому випадку він повинен сам зводити дані, тому такий підхід значно ускладнює текст програми для клієнта. Змінити систему стає складніше, оскільки вона спілкується безпосередньо з сервісом.

Однією з великих переваг підходу API шлюзу є інкапсуляція внутрішньої структури системи. Клієнт робить запити лише до API шлюзу, а не викликає певну службу. У цьому випадку текст програми на клієнті буде виглядати набагато простіше.

З огляду на наведені вище причини, у цій магістерській роботі було вибрано використання API шлюзу замість клієнта, який безпосередньо взаємодіє зі службою.

Spring Cloud Gateway використовується для реалізації API шлюзу. Це приклад реалізації шаблону Gateway у мікросервісному середовищі. Він використовує можливості сервера Netty для перенаправлення запитів до внутрішньої служби. Шлюз Spring не блокує, оскільки він побудований на Spring WebFlux.

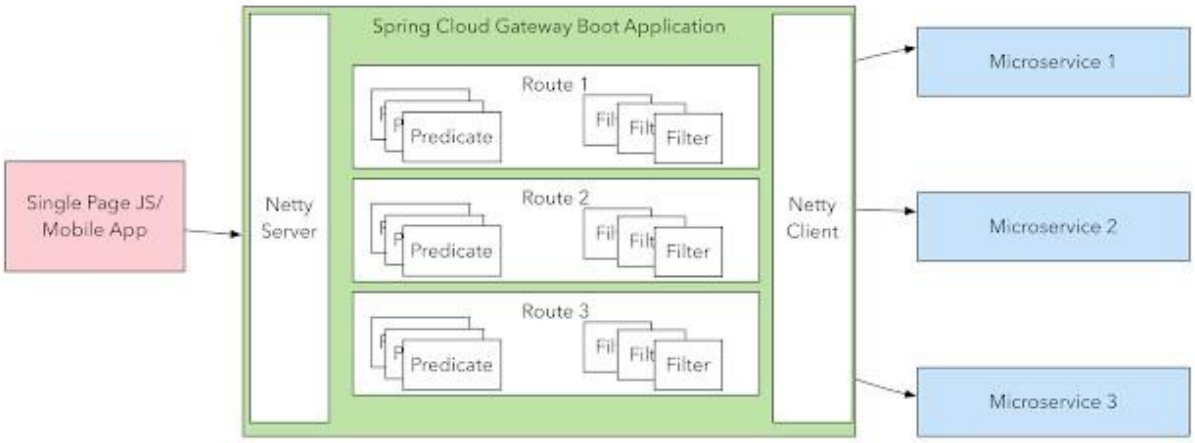


Рис. 5.3. Використання Spring API шлюзу

Spring Gateway обробляє деякі запити, просто перенаправляючи їх на певний мікросервіс, тоді як для інших запитів він обробляє багато сервісів, що містять дані, і поєднує результати, отримані від цих сервісів. Крім того, API Gateway використовує служби авторизації та автентифікації для автентифікації користувачів перед пересиланням запитів до інших внутрішніх служб.

Щоб підвищити загальну продуктивність системи, база даних Redis і Spring Caching використовуються для кешування запитів користувачів на рівні шлюзу. Spring Actuator використовується для збору індикаторів і моніторингу стану системи, а також доповнює інтерфейс API шлюзу з точки зору технічних функцій. Redis — це система керування базою даних NoSQL, яка використовує структуру ключ-значення. Він використовується як для кешування та реалізації брокерів повідомлень, так і для баз даних.

5.1.3. Сервіс розпізнавання мовлення

Сервіси розпізнавання мовлення призначені для перетворення голосових команд користувачів у текст. В архітектурі, створеній для голосового помічника, сервіс виконується на хмарному сервері та спілкується з ним через шлюз API.

Системи розпізнавання мовлення важко реалізувати, оскільки існує багато джерел варіативності, які впливають на точність розпізнавання речень. Нижче наведено найважливіші фактори, які слід враховувати під час створення служби розпізнавання мовлення користувача.

По-перше, такою особливістю є стиль мовлення користувача. Кожен має різний стиль розмови, зокрема різний акцент. Наприклад, для англійської мови такими акцентами є американська англійська, британська англійська, австралійська англійська тощо, оскільки це найпоширеніша мова у світі. Вимова певних слів або звуків також ускладнює завдання системи розпізнавання мовлення.

Середовище також впливає на точність розпізнавання, оскільки воно впливає на рівень фонового шуму під час запису команд. Голосові команди, записані в ізольованій кімнаті, міститимуть більше шуму, ніж в аудиторії, оскільки навіть луна може внести шум у систему. Слід також враховувати, що кожна людина має свої особливості людського мовлення, які залежать від багатьох факторів, у тому числі від якості звуку та розбірливості. Системи розпізнавання мовлення повинні враховувати та вирішувати вищезазначені проблеми, щоб досягти достатньої точності.

Мова програмування Python містить багато модулів і бібліотек, які вирішують завдання розпізнавання мови та її перетворення в текст, зокрема ApiAI, SpeechRecognition, Google Speech Cloud, AssemblyAI, PocketSphinx, Watson Developer Cloud, WIT та інші. Використовуйте мову програмування Python і модуль SpeechRecognition для впровадження служб розпізнавання мовлення. Цей модуль надає різні інструменти від різних постачальників для розпізнавання мовлення та перетворення його на текст. Служба розпізнавання мовлення використовує модель перетворення мови в текст IBM. Використане рішення базується на методах глибокого машинного навчання, а саме на використанні рекурентних нейронних мереж з блоками LSTM і моделями WaveNet. Використане рішення надає можливість розпізнавати голосові команди кількома мовами та аудіоформатами.

У результаті служба розпізнавання мовлення повертає відповідь JSON через протокол HTTP REST, що містить розпізнаний вміст у кодуванні UTF-8.

5.1. 5. Сервіс ідентифікації іменованих сутностей

Служба розпізнавання іменованих сутностей відповідає за ідентифікацію команд користувача та їх параметрів. Для досягнення розпізнавання використовували рекурентну нейронну мережу LSTM та бібліотеку SpaCy, реалізовану на мові програмування Python.

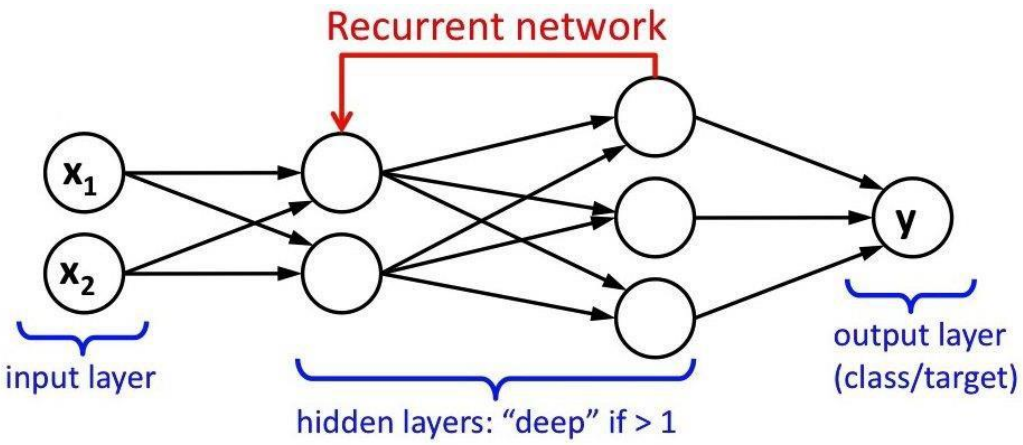


Рис. 5. 5. Рекурентна нейронна мережа

SpaCy — це популярна бібліотека аналізу тексту та обробки природної мови Python. Він розроблений на основі мови програмування Cython, яка є підвидом Python, але має набагато кращу продуктивність. Тому багато компаній віддають йому перевагу при реалізації власних проектів.

Бібліотека містить інструменти для токенізації та класифікації тексту, визначення частин мови, визначення залежностей між словами в реченнях, розпізнавання іменованих сутностей та інтеграцію з інструментами глибокого машинного навчання та візуалізації даних.

Сервіс використовує для передачі повідомлень протокол HTTP і методи REST. Використовуйте службу реєстрації подій, щоб реєструвати кожен вхідний запит і розпізнавання іменованих об’єктів. Таким чином ви можете відстежувати іменовані сутності для конкретних команд користувача.

5.1.5. Сервіс конфігурації

Служба конфігурації — це централізоване місце для керування зовнішніми налаштуваннями для кожної служби, незалежно від середовища. В якості реалізації було обрано Spring Cloud Config, який спрощує зовнішнє керування конфігурацією як на стороні сервера, так і на стороні клієнта великих розподілених систем.

До того, як мікросервіси стали поширеними, конфігураційні файли створювалися таким чином, що під час їх зміни контейнер потрібно було перезавантажувати. Як правило, під час розробки системи використовується підхід із кількома різними середовищами, кожне з яких має підтримувати власні конфігураційні файли, а для перемикання між ними всю систему потрібно призупинити. Тому дуже важливим питанням є те, що кодова база тісно пов'язана з конфігураційними файлами, і відповідно будь-які зміни призведуть до перерозгортання та збирання всієї системи.

Усі перераховані вище проблеми можна вирішити за допомогою хмарних служб налаштування.

Перевага цього підходу до централізованої конфігурації полягає в тому, що коли в параметри мікросервісу вносяться зміни, вони обробляються на льоту без необхідності перекомпілювати мікросервіс.

Усі параметри зберігаються службою конфігурації в SVN (системі контролю версій) і можуть бути змінені під час роботи системи.

Мікросервіс, якщо вам потрібна та чи інша конфігурація, викликайте її через REST API. Схема роботи служби іменування показана на рисунку 2. 5.5.

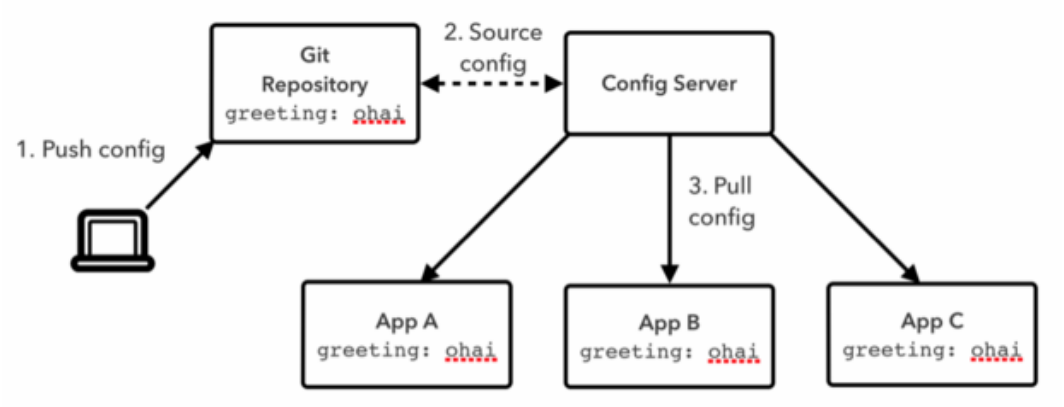


Рис. 5.5. Принцип роботи сервісу конфігурацій

5.1.6. Сервіс виявлення сервісів

В архітектурі мікросервісів служби ідентифікації сервісів відіграють важливу роль, оскільки кожна служба має кілька екземплярів, тому потрібен механізм ідентифікації адрес інших служб для наступних викликів без необхідності явного введення кожного мікросервісу раніше. Виклики хосту та порту служби є виконано. Крім того, під час роботи системи адреса мікросервісу може змінюватися в реальному часі. Наприклад, можуть бути створені нові екземпляри мікросервісу або старі екземпляри мікросервісу можуть вийти з ладу. Тому механізм автоматичної реєстрації дуже важливий для виявлення послуги. Сервіси виявлення та реєстрації сервісів реалізовані за допомогою рішень Netflix Eureka та Spring Cloud. Netflix Eureka використовується для створення реєстру мікросервісів і автоматичного запису в нього кожного мікросервісу під час запуску. Після цього всі інші сервіси мають можливість викликати його за унікальним ідентифікатором сервісу, введеним під час реєстрації.

Spring Cloud використовується для спрощення пошуку служб і створення реєстру, використовуючи стрічку як балансир. Він балансує клієнтські запити до сервера. Перш ніж зробити запит, клієнт повинен проконсультуватися з мікросервісом ідентифікації служби, щоб знайти необхідну інформацію, таку як ім’я хоста, IP-адреса, тип порту тощо. Після цього Ribbon використовує Round-Robin, щоб вибрати екземпляр зі створеного списку для виконання запиту. Щоб не виконувати кожного разу запит до сервісного сервісу, дані, отримані від нього, записуються в кеш кожного клієнта. Спосіб клієнтського запиту на розповсюдження показаний на рисунку 1. 5.6.

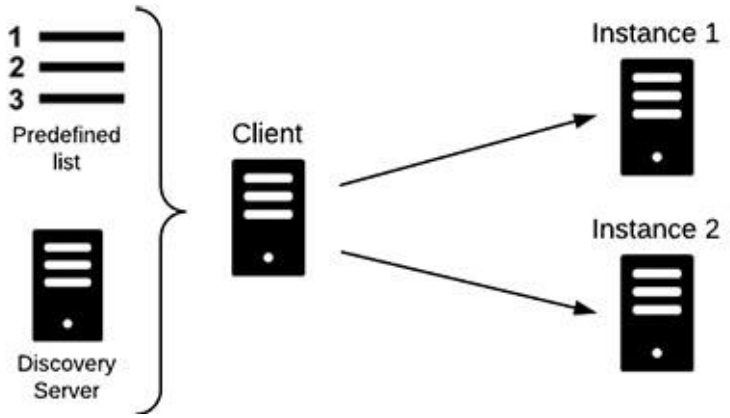


Рис. 5.6. Схема балансування на стороні клієнта

Перевагами балансування на стороні клієнта є стабільність і децентралізація, тому цей підхід був обраний для реалізації в даному дипломному проекті.

5.1.7. Сервіс логування

Оскільки всі служби працюють на різних портах і хостах, для зберігання інформації про події, які відбуваються під час роботи голосового помічника, потрібне централізоване зовнішнє розташування. Розподілена система журналювання дозволяє переглядати й аналізувати інформацію про всі події, які відбуваються в багатьох різних мікросервісах у вашій системі, а також спрощує процес налагодження та пошуку помилок.

Сервіс використовує для передачі повідомлень протокол HTTP і методи REST. Використовуйте службу реєстрації подій, щоб реєструвати кожен вхідний запит і розпізнавання іменованих об’єктів. Таким чином ви можете відстежувати іменовані сутності для конкретних команд користувача.

Щоб вирішити проблему реєстрації подій, ELK вибрала набір технологій, а саме Logstash, ElasticSearch і Kibana.

1. Filebeat + Logstash — це динамічний конвертер інформації з розгалуженою системою надбудов і потужною взаємодією з ElasticSearch. Оскільки Filebeat збирає журнали подій з кожної служби в системі та записує їх у сховище ElasticSearch, його потрібно встановити на всіх серверах з екземплярами мікросервісів. Filebeat — це програма, єдине

завдання якої — надсилати журнали з кожної служби в систему Logstash, і вона дозволяє встановлювати компоненти Logstash в одному місці. Він невеликий за розміром і займає менше системних ресурсів.

2. ElasticSearch — це система пошуку, розповсюдження та аналітики, створена для максимальної надійності, горизонтальної масштабованості та простого керування інформацією.
3. Kibana використовується для забезпечення візуалізації даних через інтерфейс користувача.

На рисунку. На рисунку 18 показано принцип взаємного зв'язку між компонентами групи ELK. Кожна служба записує події, що відбуваються під час її роботи, у свій власний файл. Filebeat встановлюється на кожному сервері, який потребує збору даних для подальшого пересилання, і передає дані з цих файлів у компонент Logstash. У свою чергу, він отримує передані дані, перетворює їх у формат JSON і надсилає в ElasticSearch для подальшого зберігання. Kibana отримує доступ до записів у ElasticSearch і виконує візуалізацію для клієнта.

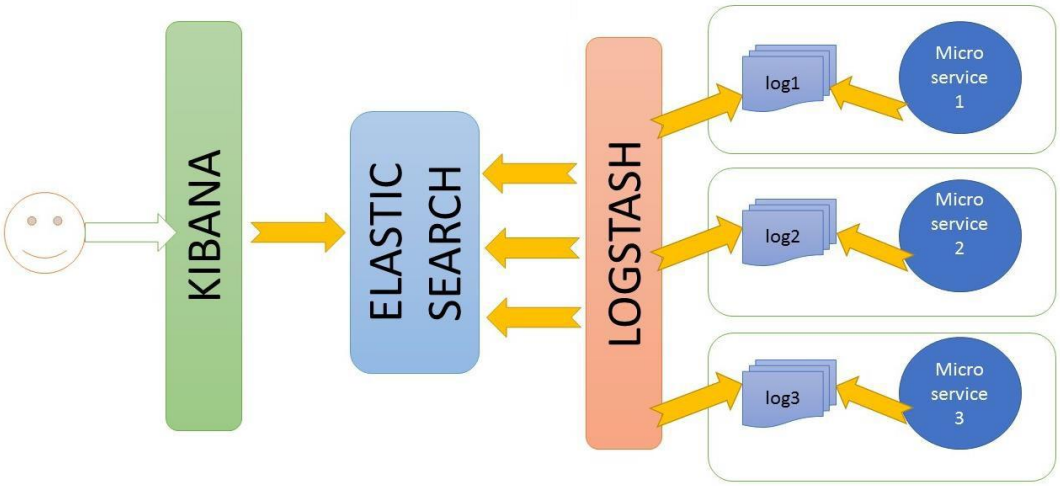


Рис. 5.7. ELK стек для логування

5.2. Тестування розробленого програмного забезпечення

Через розподілений характер коду тестувати мікросервіси стає набагато складніше, ніж тестувати повну систему. Оскільки я глибоко розумію систему та прагну до максимального охоплення тестами, я використав підхід білої скриньки.

Модульні тести використовуються для тестування кожного мікросервісу окремо. Це робиться для перевірки того, що нові зміни в коді не викликають регресії. Використовуйте бібліотеку Jasmine для тестування клієнтського коду, а серверна частина використовує фреймворки JUnit і Mockito для реалізації модульного тестування. Модульне тестування виявляє більшість проблем на початку циклу розробки програмного забезпечення. Вони включають як помилки під час реалізації проекту, так і «відсутні» компоненти в специфікації модуля або частини системи. Вартість пошуку помилок на ранніх стадіях розробки та тестування набагато нижча, ніж вартість їх виявлення та виправлення на наступних стадіях розробки програмного забезпечення. Крім того, модульне тестування значно спрощує інтеграцію з іншими системами, оскільки ви можете переконатися, що всі модулі працюють ізольовано відповідно до реалізації розробника. Структура Selenium використовується для виконання «димового тестування» або так званого димового тестування та покриття критичного шляху системи. На цьому етапі тестування всі основні перевірені

Система функціонує відповідно до вимог.

Створені тести дають можливість покращити якість впровадженого програмного забезпечення та отримати впевненість, що внесення нових змін не викличе помилок у існуючій реалізації.

5.3. Аналіз розробленої системи

Програмні рішення, розроблені для голосових помічників, інтегрованих із середовищами розробки програмного забезпечення, створюються з використанням архітектури мікросервісів. Система складається з клієнта, API шлюзу та 7 сервісів, кожен сервіс відповідає за свою частину програми.

Клієнт надається як плагін для IntelliJ IDEA, але подібні клієнти можна створювати для інших інтегрованих середовищ розробки програмного забезпечення, а також для IDE на мобільних пристроях.

Система розпізнає англійські команди користувача та надає можливість створювати та редагувати змінні, функції, класи, файли та керувати середовищами розробки. Точність розпізнавання команд досить висока завдяки використанню вдосконаленого алгоритму розпізнавання іменованих сутностей, який враховує контекст користувача.

Серед недоліків цієї системи можна виділити те, що для використання повної її функціональності потрібен доступ до мережі, оскільки всі обчислення для ідентифікації команд користувача та іменованих сутностей виконуються на сервері. Також рекомендується додати підтримку інших мов, крім англійської.

Голосові помічники, створені для інтегрованих середовищ розробки, можуть підвищити продуктивність розробників, надаючи можливість писати програмний текст на мобільних пристроях і людям з обмеженими можливостями.

Голосовий помічник відповідає всім критеріям оцінювання, зазначеним у розділі 1, тобто він спеціально створений для інтегрованого середовища розробки програмного забезпечення, дозволяє користувачам видавати розширені команди звичайною розмовною мовою, є безкоштовним у використанні, містить програмування з відкритим текстом і використовує архітектура мікросервісів Створити дизайн. Архітектура дозволяє легко вносити зміни у вашу систему.

5. 5. Рекомендації щодо подальшого вдосконалення

Для подальшого вдосконалення системи пропонується розширити перелік підтримуваних інтегрованих середовищ розробки програмного забезпечення, мов програмування та команд користувача за рахунок впровадження нових клієнтів. Оскільки створювана архітектура програми не прив’язана до конкретного клієнта і легше досягти кросплатформної сумісності, доцільно було б реалізувати клієнт на основі мобільних платформ.

Оскільки створені наразі голосові помічники повністю функціонують лише в онлайн-режимі (при наявності підключення до Інтернету), доцільно розпізнавати команди користувача та розширювати клієнт за допомогою менш ресурсомістких алгоритмів, хоча точність розпізнавання буде втрачена. Визначте іменовані сутності. Точність розпізнавання можна підвищити шляхом усунення шумів на етапі перетворення голосового повідомлення користувача в текст. Оскільки наразі система підтримує лише команди, подані англійською мовою, рекомендується розширити її, підтримуючи команди введення іншими мовами.

Незважаючи на ці зауваження, система цілком придатна для використання. У цьому розділі розроблено архітектуру системи голосового помічника, інтегрованої з середовищем розробки програмного забезпечення. Надається детальний опис кожного компонента (мікросервісу) системи. Описано процес тестування системи та методи подальшого вдосконалення. Створену систему проаналізовано на відповідність встановленим стандартам.

6 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

6.1 Загального схема системи

Перед тим, як пристрій почне працювати, нейронна мережа повинна бути заздалегідь навчена на власних зразках, які будуть створені індивідуально для користувача для підвищення точних характеристик.

Під час навчання система отримує зразки з набором тренувальних і тестових команд, а потім передає їх на блок MFCC, який, у свою чергу, відображає їх у частотних характеристиках mel. Потім нейронна мережа отримує ці дані та вдосконалює їх за допомогою алгоритмів оптимізації.

Після навчання нейронна мережа перетворюється на бібліотеку для середовища розробки Arduino. Бібліотека включена в проект і буде завантажена на дошку разом з бібліотекою для виконання завдань, розпізнаних командою.

Сучасні системи розпізнавання мови часто мають багатошарову модульну структуру. На першому рівні виконується попередня обробка і

Виберіть акустичні характеристики, які характеризують голосові команди. Одним із найбільш часто використовуваних методів сьогодні є вибір Кепстральних коефіцієнтів Мела (MFCC).

Mel — психофізична одиниця висоти, пов’язана з частотою. Функції, отримані на основі цього підходу, мають багато корисних властивостей - їх легко обчислити, вони забезпечують компактне представлення мовних команд і стійкі до шумових перешкод з навколишнього середовища.

Наступним рівнем у системах розпізнавання голосових команд є мова. Він передбачає процес пошуку в стандартному словнику голосової команди. При розпізнаванні однієї голосової команди диктор вимовляє слово без навколишнього контексту. Навчання таких систем є трудомістким завданням, і для підвищення надійності зазвичай використовується велика кількість навчальних зразків (5 і більше варіантів вимови з однієї голосової команди). Кожна команда записується в

стандартний словник як набір дрібнозернистих кепстральних коефіцієнтів. Структура цієї системи показана на рисунку. 6.1.

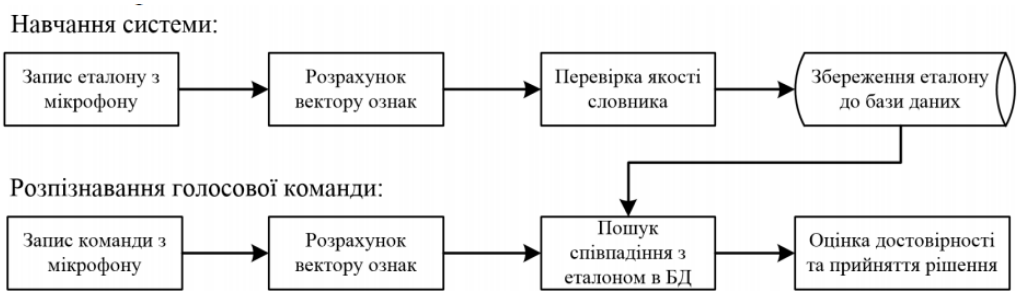


Рис. 6.1. Структура системи

Алгоритм розрахунку низькочастотних кепстральних коефіцієнтів. Цей спосіб отримання ознак є одним із найпоширеніших методів у системах розпізнавання дикторів і системах розпізнавання мови. Вхідними даними для алгоритму є послідовність підрахунків для частини сигналу, за якою слідує ця ітерація. До послідовності застосовується вагова функція та виконується дискретне перетворення Фур'є.

Вагові функції використовуються для зменшення спотворень, спричинених кінцевою природою зразків у аналізі Фур'є. На практиці вікно Хеммінга (1) і вікно Ханни (2) часто використовуються як вагові функції.

Алгоритм Адама використовується для навчання нейронних мереж. Цей алгоритм можна розглядати як комбінацію RMSprop і стохастичного градієнтного спуску імпульсу. Він використовує квадратичні градієнти для масштабування швидкості навчання, як RMSprop, і імпульс, який використовує ковзне середнє градієнта замість самого градієнта, як SGD з імпульсом.

Adam — це метод адаптивної швидкості навчання, що означає, що він розраховує індивідуальну швидкість навчання для різних параметрів. Його назва походить від оцінки адаптивного моменту, яка називається так тому, що Адам використовує оцінки першого та другого моментів градієнта для адаптації до швидкості навчання кожної ваги нейронної мережі. Момент N випадкової величини визначається як очікуване значення змінної в n-му степені.

6.2 Реалізація програмного забезпечення та тестування

Процес розробки програми ділиться на 2 етапи:

По-перше, це написати програму на Python, яка дозволить вам отримати навчену модель і пізніше перетворити її в бібліотеку для середовища розробки програми керування платою пристрою.

Наступні 6 команд призначені для управління автомобілем, створюючи клас і окремий клас "шуму" для кожної команди: Вперед, Назад, Вліво, Вправо, Швидше, Стоп.

Другим кроком є написання команд керування, які працюватимуть на інвалідному візку та розпізнавання команд керування. Під час роботи команда управління буде записувати звуки на вбудований мікрофон і класифікувати отримані записи за допомогою попередньо отриманої бібліотеки з навченими нейронними мережами. Загальна схема керуючої програми представлена на рисунку 2. 6.2.

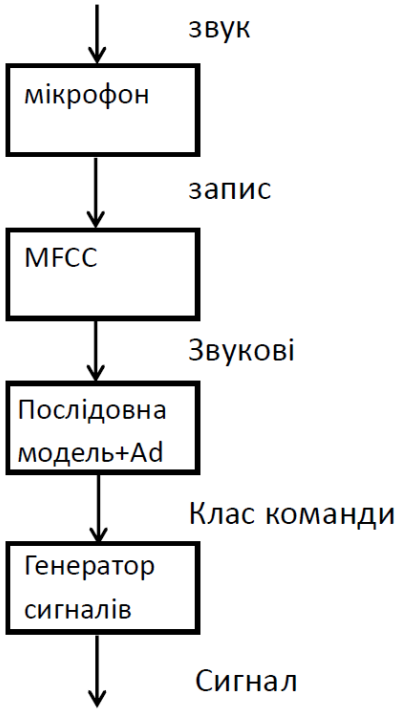


Рис. 6.2. Загальна схема роботи керуючої програми

Блок MFCC має такі характеристики:

- Тривалість запису – 2 секунди Тривалість вікна – 1200 мс

- кроки - 50 мс.

Результати тестування показують, що система показує свою продуктивність. Всі 6 команд управління перевірені на розпізнавання.

У процесі тестування були випадки, коли програма не могла правильно визначити команди або не могла розрізнити команди. Загальна точність розпізнавання, отримана системним тестом: 98%

Також були отримані інші характеристики навченої нейронної мережі. Ці характеристики наведені в таблиці 6.1.

Таблиця 6.1. Результати експериментів

Кількість елементів у класі	Точність, %	Втрати	Час взаємодії, мс	Максимальна задіяна оперативна пам'ять, kB	Використання постійної пам'яті, kB
100	78.1	0.52	70	16	35.4

Як видно з таблиці, отримані результати свідчать про працездатність та ефективність системи, а отримані дані дозволяють говорити про можливість подальшого розвитку системи.

Для більш детального опису отриманих функцій було протестовано 3 додаткові бібліотеки з іншими вхідними параметрами під час навчання нейронної мережі. Системи з різною кількістю елементів у кожному класі та різною кількістю класів також були перевірені, щоб отримати повне розуміння впливу кількості класів на продуктивність системи.

Табл 6.2. Розширені результати експерементів

Кількість класів	Кількість елементів у класі	Точність, %	Втрати	Час взаємодії, мс	Максимальна задіяна оперативна пам'ять, kB	Використання постійної пам'яті, kB
7	100	78.1	0.52	70	16	35.4
7	15	80,5	0.42	67	18,3	38,7
3	100	81.1	0.45	66	12	29.9
3	15	92.3	0.23	55	15.6	35.5

Тому розроблене рішення дозволяє розпізнавати дві голосові команди за допомогою попередньо навченої мережі, яка згодом використовується на платі з обмеженими ресурсами. Основною перевагою є використання власних прикладів, що дозволяє налаштувати систему під ваших користувачів. Ще одна перевага полягає в тому, що система використовує невелику, але сучасну плату, яка може легко взаємодіяти з іншими платами тієї ж серії, забезпечуючи універсальність і масштабованість. Система демонструє свою працездатність на практиці на основі статистичних характеристик. Надалі систему можна вдосконалити шляхом збільшення навчальних вибірок, тим самим підвищивши точність розпізнавання, а також систему можна вдосконалити шляхом додавання словників, що позитивно позначиться на практичності системи.

ВИСНОВКИ

Постійний розвиток інформаційних технологій призвів до збільшення кількості існуючих апаратних і програмних засобів, які використовуються в повсякденному житті. Це дає можливість впроваджувати сучасні технології в будь-яку сферу життя суспільства. Це призводить до того, що різні групи людей можуть покращити своє життя за допомогою сучасних технологій, тобто за допомогою новітніх систем і алгоритмів розпізнавання голосу є можливість створювати системи розпізнавання вимови.

Результатом виконання магістерської роботи є:

- Досліджено існуючі способи розпізнавання команд на основі нейронних мереж. Розглядаються їх основні характеристики, принципи роботи, робочі характеристики та області застосування.

- Порівняльна характеристика існуючих рішень і підтвердження необхідності створення їх аналогів.

– Висвітлено основні особливості системи розпізнавання, зокрема використання сучасних інструментів, використання української мови як нейромережевого словника та автоматизацію використання системи на прикладі додатку покращення для вимови.

Детально розглянуто техніку розпізнавання та синтезу мови. Для можливості порівняння вимови розроблено алгоритм порівняння аудіофайлів та розглянуто математичні методи реалізації алгоритму.

Результатом дослідження стало визначення технологічного стеку та розробка архітектури програмного забезпечення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. 9 додатків для вивчення правильної вимови англійських слів [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: http://headtechnology.com.ua/120-9-dodatkiv-dlya-vivchennya-pravilnoyi-vimovi- angliyskikh-sliv.html.

2. Müller M. Fundamentals of Music Processing — Audio, Analysis, Algorithms, Applications. / Meinard Müller., 2015. – 480 с.

3. Применение преобразования Фурье в цифровой обработке звука [Електронний ресурс] – Режим доступу до ресурсу: http://shackmaster.narod.ru/fourier.htm.

4. В. П. Денисюк, В. К. Репета. Вища математика – Київ: НАУ, 2013. – 145 с.

5. Schildt H. Java: A Beginner's Guide / Herbert Schildt., 2014. – 720 с. – (6thEdition).

6. Модель-представлення-контролер [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Модель-вид-контролер

7. Walls C. Spring in Action / Craig Walls., 2014. – 624 с.

8. Christian Bauer, Gavin King Hibernate in action – M: Manning Publications, 2005 – 408 p.

9. PostgreSQL [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/PostgreSQL

10. Deep Voice: Real-time Neural Text-to-Speech / Coates Adam, 2017. – 17 с.

11. Kamath U. Deep Learning for NLP and Speech Recognition / U. Kamath, J. Liu, J. Whitaker., 2019. – 640 с. – (1st Edition).

12. Dutta A. Audio Processing and Speech Recognition: Concepts, Techniques and Research Overviews / A. Dutta, S. Sen, N. Dey., 2019. – 107 с. – (1st Edition).

13. Mehta P. Comparative Study of Various Algorithms for Speech Synthesis System / P. Mehta, B. Prajapti., 2016. – 96 с.

14. Dutoit T. An Introduction to Text-to-Speech Synthesis / Thierry Dutoit., 1997. – 330 с.

15. Aggarval C. C. Neural Networks and Deep Learning / Charu C. Aggarval., 2018. – 497 с.

16. Худа А.О., Халус О. А. «Метод порівняння аудіофайлів у системі для вивчення іноземних слів» // Матеріали IV всеукраїнська науково-практична конференція молодих вчених та студентів «Інформаційні системи та технології управління» – м. Київ.: НТУУ «КПІ ім. Ігоря Сікорського»

17. Introduction to Voice Recognition With Elechouse V3 and Arduino [електронний ресурс] – Режим доступу: https://www.instructables.com/Introduction-to-Voice- Recognition-With-Elechouse-V/

18. Голосовое управление посредством Arduino [електронний ресурс] – Режим доступу: https://future2day.ru/golosovoe-upravlenie-posredstvom-arduino/

19. MIT Intelligent Wheelchair Project: A Voice-Commandable Robotic Wheelchair [електронний ресурс] – Режим доступу: https://techtv.mit.edu/videos/6465-mit-intelligent-wheelchair-project-a-voice-commandable-robotic-wheelchair

20. Metz, Cade (November 9, 2015). "Google Just Open Sourced Tensor Flow, Its Artificial Intelligence Engine". Wired. RetrievedNovember 10, 2015.

21. Piatetsky, Gregory. "Python eatsawayat R: Top Software for Analytics, Data Science, Machine Learningin 2018: Trendsand Analysis". KDnuggets. KDnuggets. Retrieved 30 May 2018.

22. A. Y. Ngand M. I. Jordan. On Discriminative vs. Generative Classifiers: A comparison of logistic regression and Naive Bayes. in NIPS 14, 2002.

23. Ben-Hur, Asa; Horn, David; Siegelmann, Hava; Vapnik, Vladimir N. ""Support vector clustering" (2001);". Journal of Machine Learning Research. 2: 125–137.

24. R. Quinlan, " Learning efficient classification procedures ", Machine Learning: an artificial intelligence approach, Michalski, Carbonell & Mitchell (eds.), Morgan Kaufmann, 1983, p. 463–482.

25. Altman, Naomi S. (1992). " An introduction to kernel and nearest-neighbor nonparametric regression " (PDF). The American Statistician. 46 (3): 175–185.

26. N. Qian, “On the momentum termin gradient descent learning algorithms.” Neural Networks : The Official Journal of the International Neural Network Society, 1999, 12(1), 145–151.

27. Дідерік П. Кінгма та Джиммі Лей Ба. Адам: Метод стохастичної оптимізації . 2014

ДОДАТКИ

Додаток А

(обов'язковий)

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації

З А Т В Е Р Д Ж У Ю

Завідувач кафедри АІІТ
д.т.н., професор Олег БІСІКАЛО

_____
(підпис)

«____» _____ 2023 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на магістерську кваліфікаційну роботу

Автоматизована система розпізнавання виголошених слів методами машинного навчання

08-31.МКР.004.02.000 ТЗ

Керівник к.т.н. доцент кафедри АІІТ
_____ Володимир ГАРМАШ
« _____» _____ 2023 р.

Розробив студент гр. ЗАКІТ-22м
_____ Антон БІЛОУС
«_____» _____ 2023 р.

Вінниця ВНТУ 2023

1. Назва та галузь застосування.

Автоматизована система розпізнавання виголошених слів методами машинного навчання. Інформаційні системи та технології.

2. Підстава для проведення робіт.

Підставою для виконання роботи є наказ №__ по ВНТУ від «__» _____ 2023р., та індивідуальне завдання на МКР, затверджене протоколом №__ засідання кафедри АІІТ від «__» _____ 2023р.

Термін виконання робіт:

3. Мета та вихідні дані для проведення робіт.

Метою магістерської кваліфікаційної роботи є розробка автоматизованої системи розпізнавання виголошених слів за допомоги методів машинного навчання.

4. Джерела розробки.

4.1. В. П. Денисюк, В. К. Репета. Вища математика – Київ: НАУ, 2013. – 145 с.
4.2. Методичні вказівки до виконання магістерських кваліфікаційних робіт для студентів спеціальностей 126 – «Інформаційні системи та технології», 151 – «Автоматизація та комп'ютерно-інтегровані технології» / Уклад. Р. Н. Квєтний, О. М. Бевз, О. В. Бісікало, Маслі Р.В. – Вінниця: ВНТУ, 2020. – 34 с.

5. Технічні дані.

5.1. Середній час розпізнавання голосової команди: 2,5 с.
5.2. Максимальний час розпізнавання голосової команди: 3,2 с.
5.3. Середній час запуску програми: 6 с.

6. Стадії розробки.

а) Дослідження предметної області	28.10 – 05.11
б) Обгрунтування методів реалізації та контролю	06.11 – 17.11
в) Розробка структури та функціональної схеми	18.11 – 24.11
г) Розробка прогармного забезпечення	25.11 – 30.11
д) Тестування та перевірка припущень	31.11 – 06.12
е) Оформлення матеріалів до захисту МКР	07.12 – 12.12

7. Порядок контролю та приймання.

Рубіжний контроль провести до «___» _____ 2023 р.
Попередній захист МКР провести «___» _____ 2023 р.
Захист МКР провести до «___» _____ 2023 р.

Розробив студент групи 3АКІТ-22м _____Антон БІЛОУС

Додаток Б

(обов’язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

АВТОМАТИЗОВАНА СИСТЕМА РОЗПІЗНАВАННЯ ВИГОЛОШЕНИХ СЛІВ МЕТОДАМИ МАШИННОГО НАВЧАННЯ

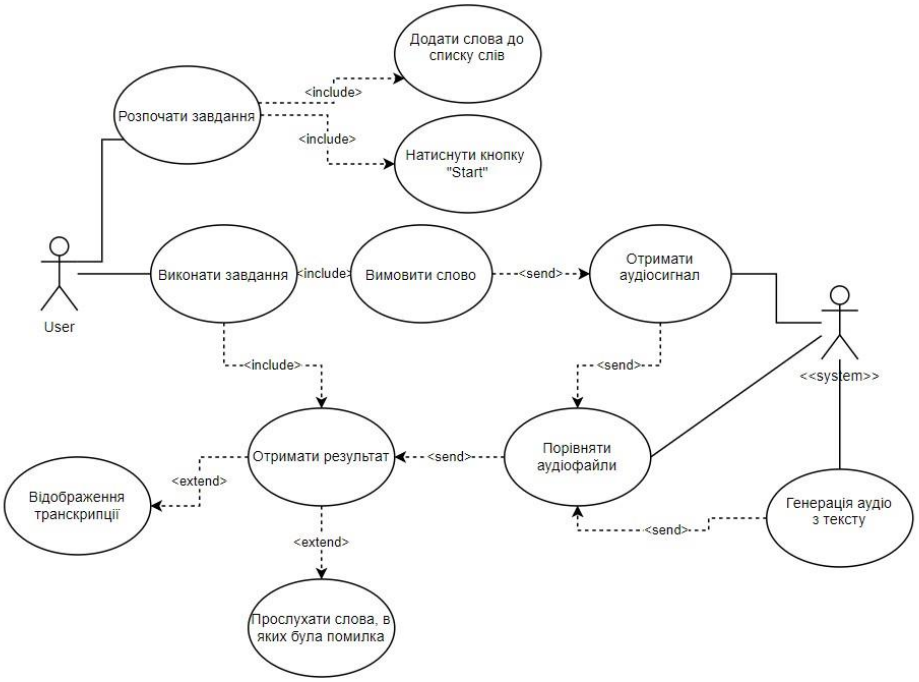


Рисунок Б.1 – Схема структурна варіантів використання

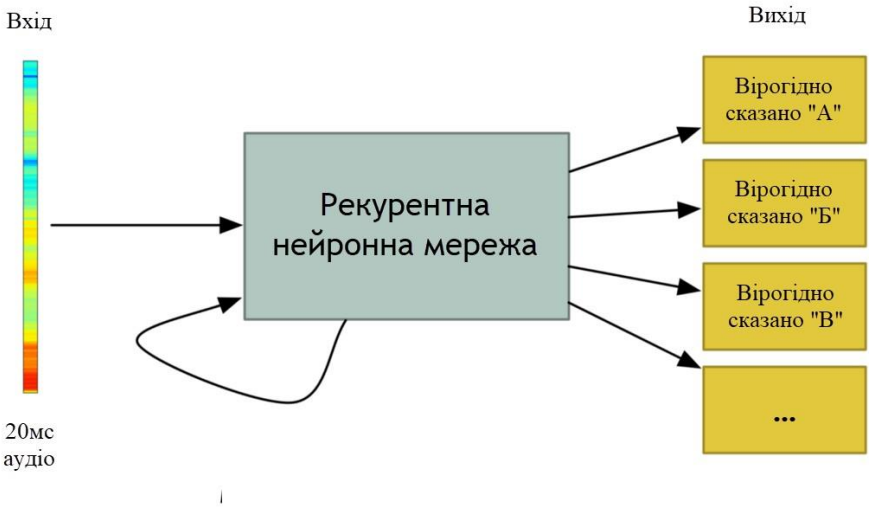


Рисунок Б.2 – Модель зі збереженням стану

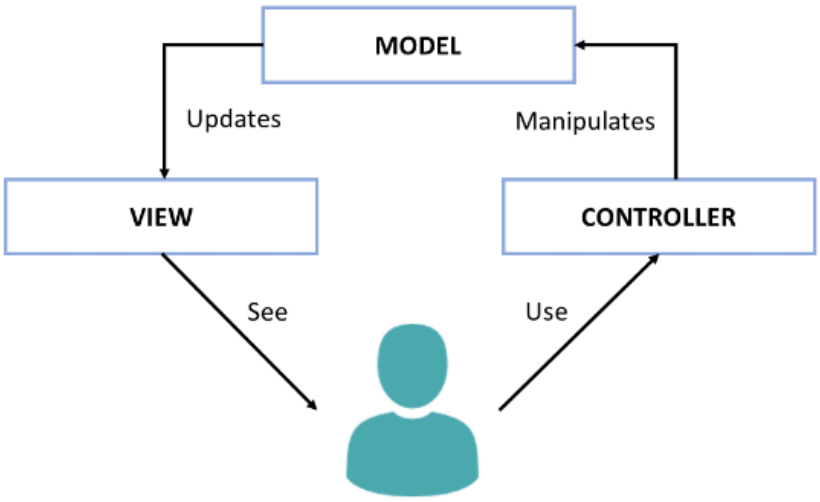


Рисунок Б.3 – Структура шаблону MVC

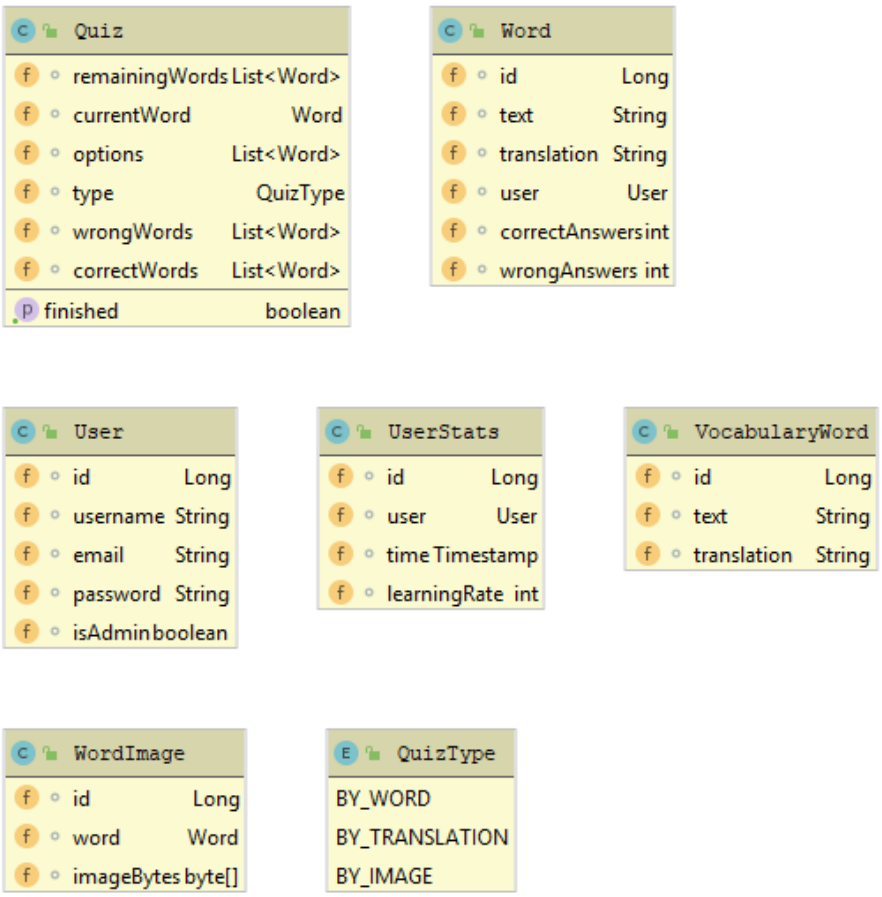


Рисунок Б.4 – Схема структурна класів моделей

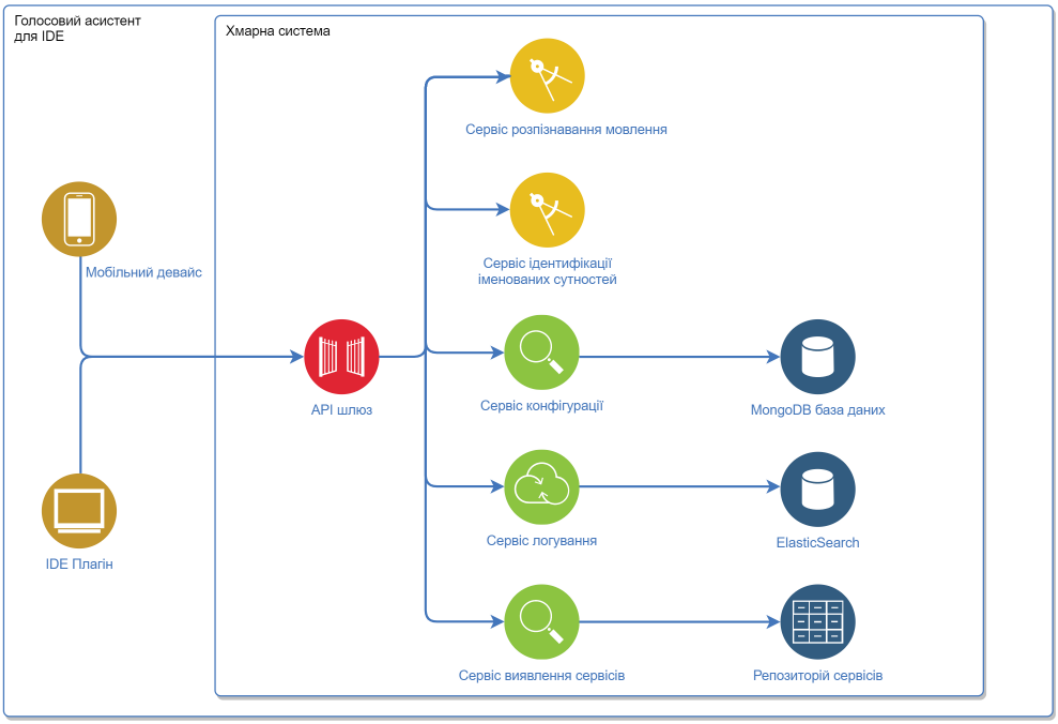


Рисунок Б.5 – Архітектура створеної системи

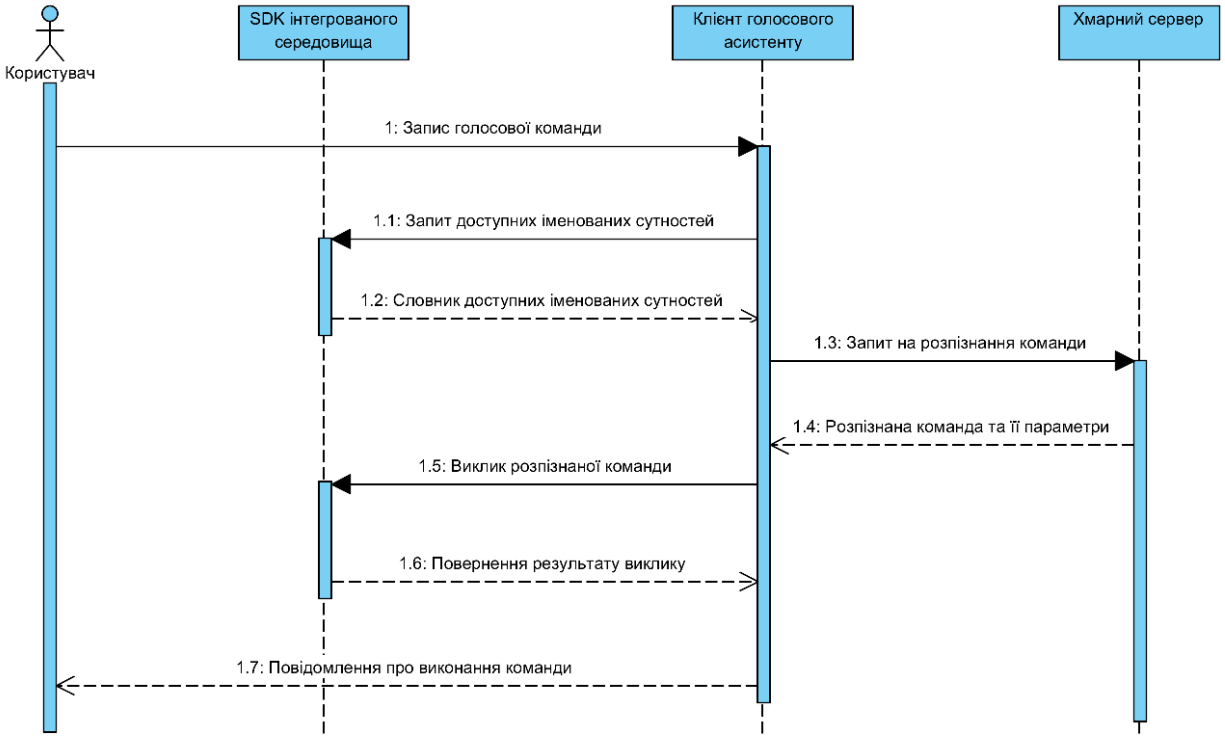


Рисунок Б.6 – Обробка голосової команди користувача

Додаток В
(обов'язковий)
Фрагмент лістингу програми

Quiz.java

```
package org.kpi.easylearn.model; import lombok.Data;
import java.util.List;

@Data
public class Quiz {
List<Word> remainingWords; Word currentWord; List<Word> options; QuizType type;
List<Word> wrongWords; List<Word> correctWords;

public boolean isFinished() { return currentWord == null;
}
}
```

QuizType.java

```
package org.kpi.easylearn.model; public enum QuizType {
BY_WORD,
BY_TRANSLATION, BY_IMAGE
}
```

User.java

```
package org.kpi.easylearn.model; import lombok.Data;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue; import javax.persistence.Id;
import javax.persistence.Table;
import javax.validation.constraints.NotEmpty; import javax.validation.constraints.Pattern; import javax.validation.constraints.Size;

@Entity
@Data
@Table(name = "users") public class User {
@Id
@GeneratedValue Long id;
@Size(min = 4, max = 255) String username;
@Size(max = 255)
@Pattern(regexp = "[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\\.[A-Za-z]{2,4}") String email;
@NotEmpty
String password; boolean isAdmin;
}
```

UserStats.java

```
package org.kpi.easylearn.model; import lombok.Data;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue; import javax.persistence.Id;
import javax.persistence.ManyToOne; import java.sql.Timestamp;



@Data
@Entity
public class UserStats {
@Id
@GeneratedValue Long id;
@ManyToOne(optional = false) User user;
Timestamp time; int learningRate;
}
```

VocabularyWord.java

```
package org.kpi.easylearn.model; import lombok.Data;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue; import javax.persistence.Id;
import javax.persistence.ManyToOne; import javax.validation.constraints.Min; import javax.validation.constraints.Size;

@Entity
@Data
public class VocabularyWord {
@Id
@GeneratedValue Long id;
@Size(min = 1, max = 255, message = "Word size is invalid") String text;
@Size(min = 1, max = 255, message = "Translation size is invalid") String translation;
}
```

Word.java

```
package org.kpi.easylearn.model; import lombok.Data;
import lombok.NonNull;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue; import javax.persistence.Id;
import javax.persistence.ManyToOne; import javax.validation.constraints.Min;
import javax.validation.constraints.NotEmpty; import javax.validation.constraints.Size;

@Entity
@Data
public class Word {
```

```
@Id
@GeneratedValue Long id;
@Size(min = 1, max = 255, message = "Word size is invalid") String text;
@Size(min = 1, max = 255, message = "Translation size is invalid") String translation;
@ManyToOne(optional = false) User user;
@Min(0)
int correctAnswers;
@Min(0)
int wrongAnswers;
}
```

WordImage.java

```
package org.kpi.easylearn.model; import lombok.Data;



import javax.persistence.Entity;
import javax.persistence.GeneratedValue; import javax.persistence.Id;
import javax.persistence.OneToOne;
import javax.validation.constraints.NotEmpty;

@Data
@Entity
public class WordImage {
@Id
@GeneratedValue Long id;
@OneToOne(optional = false) Word word;
@NotEmpty
byte[] imageBytes;
}
```

UserRepository.java

```
package org.kpi.easylearn.repository; import org.kpi.easylearn.model.User;
import org.springframework.data.jpa.repository.Query; import org.springframework.data.repository.CrudRepository; import org.springframework.data.repository.query.Param;

public interface UserRepository extends CrudRepository<User, Long> {

@Query("select u from User u where u.username = :identifier or u.email = :identifier") User findByIdentifier(@Param("identifier") String identifier);



}
```

UserStatsRepository.java

```
package org.kpi.easylearn.repository;

import org.kpi.easylearn.model.User; import org.kpi.easylearn.model.UserStats;
```

```
import org.springframework.data.repository.CrudRepository; import java.util.List;
public interface UserStatsRepository extends CrudRepository<UserStats, Long> { List<UserStats> findTop10ByUserOrderByTimeDesc(User user);
}
```

WordImageRepository.java

```
package org.kpi.easylearn.repository;

import org.kpi.easylearn.model.Word; import org.kpi.easylearn.model.WordImage;
import org.springframework.data.repository.CrudRepository;

public interface WordImageRepository extends CrudRepository<WordImage, Long> { WordImage findByWord(Word word);
}
```

WordRepository.java

```
package org.kpi.easylearn.repository;

import org.kpi.easylearn.model.User; import org.kpi.easylearn.model.Word;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.repository.JpaRepository; import org.springframework.data.jpa.repository.Modifying; import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.CrudRepository; import org.springframework.data.repository.query.Param;
import org.springframework.transaction.annotation.Transactional; import java.util.List;
public interface WordRepository extends CrudRepository<Word, Long> {


List<Word> findByUserOrderByTextAsc(User user, Pageable page); int countByUser(User user);
@Query("select count(w) from WordImage wi join Word w on wi.word = w where w.user = :user") int countByUserWithImages(@Param("user") User user);



* 2 " +
```

```
rand()")



}

@Query("select count(w) from Word w where w.user = :user and w.correctAnswers > w.wrongAnswers "and w.correctAnswers > 2")
int countLearnedWords(@Param("user") User user);

@Modifying
@Transactional
@Query("update Word w set w.correctAnswers = w.correctAnswers + 1 where w in (:words)") void incrementCorrectAnswers(@Param("words") List<Word> words);

@Modifying
@Transactional
@Query("update Word w set w.wrongAnswers = w.wrongAnswers + 1 where w in (:words)") void incrementWrongAnswers(@Param("words") List<Word> words);

@Query("select w from Word w where w.user = :user order by rand()") List<Word> findRandomUserWords(@Param("user") User user, Pageable page);

@Query("select w from WordImage wi join Word w on wi.word = w where w.user = :user order by List<Word> findRandomUserWordsWithImages(@Param("user") User user, Pageable page);
Word findByUserAndText(User user, String text);



CurrentUserResolverService.java package org.kpi.easylearn.service; import org.kpi.easylearn.model.User;
public interface CurrentUserResolverService { User getCurrentUser();
}

ImageService.java

package org.kpi.easylearn.service;

import org.kpi.easylearn.exception.ImageTransformationException; import org.springframework.http.MediaType;
import org.springframework.web.multipart.MultipartFile;

public interface ImageService {
byte[] transform(MultipartFile imageFile) throws ImageTransformationException; MediaType mediaType();
byte[] unknownImage();
}

compare.py

from future import print_function import numpy as np
import matplotlib
```

```
import matplotlib.pyplot as plt

import librosa
import librosa.display

D, wp = librosa.core.dtw(X=x_1_chroma, Y=x_2_chroma, metric='cosine') wp_s = np.asarray(wp) * hop_size / fs

def scale_minmax(X, min=0.0, max=1.0):
X_std = (X - X.min()) / (X.max() - X.min())
X_scaled = X_std * (max - min) + min return X_scaled

def stft(y, n_fft=2048): if win_length is None:
win_length = n_fft if hop_length is None:



hop_length = int(win_length // 4)

fft_window = get_window(window, win_length, fftbins=True) fft_window = util.pad_center(fft_window, n_fft) fft_window = fft_window.reshape((-
1, 1))
util.valid_audio(y) if center:
y = np.pad(y, int(n_fft // 2), mode=pad_mode)
y_frames = util.frame(y, frame_length=n_fft, hop_length=hop_length) stft_matrix = np.empty((int(1 + n_fft // 2), y_frames.shape[1]),
dtype=dtype, order='F')

n_columns = int(util.MAX_MEM_BLOCK / (stft_matrix.shape[0] *
stft_matrix.itemsize))

for bl_s in range(0, stft_matrix.shape[1], n_columns): bl_t = min(bl_s + n_columns, stft_matrix.shape[1])

stft_matrix[:, bl_s:bl_t] = fft.fft(fft_window *
y_frames[:, bl_s:bl_t], axis=0)[:stft_matrix.shape[0]]

x_1, fs = librosa.load(sys.args[1]) x_2, fs = librosa.load(sys.args[2])

n_fft = 4410
hop_size = 2205

x_1_chroma = librosa.feature.chroma_stft(y=x_1, sr=fs, tuning=0, norm=2,
hop_length=hop_size, n_fft=n_fft) x_2_chroma = librosa.feature.chroma_stft(y=x_2, sr=fs, tuning=0, norm=2,
hop_length=hop_size, n_fft=n_fft)

print(norm(diff(x_1_chroma, x_2_chroma)))

QuizService.java

package org.kpi.easylearn.service;

import org.kpi.easylearn.model.Quiz; import org.kpi.easylearn.model.QuizType; import org.kpi.easylearn.model.User;
```

```
public interface QuizService {
Quiz createQuiz(User user, QuizType quizType);
void nextStep(User user, Quiz quiz, long selectedWordId); boolean isQuizAllowed(User user, QuizType quizType);
}
```

StatsService.java

```
package org.kpi.easylearn.service; import org.kpi.easylearn.model.User; import java.util.List;
public interface StatsService {
List<Integer> getUserLearningRate(User user);

void recordStats(User user);
}
```

UserService.java

```
package org.kpi.easylearn.service;

import org.kpi.easylearn.exception.RegistrationException; import org.kpi.easylearn.model.User;

public interface UserService {
User register(User user) throws RegistrationException;
}
```

WordImageService.java

```
package org.kpi.easylearn.service;

import org.kpi.easylearn.exception.ImageTransformationException; import org.kpi.easylearn.model.Word;
import org.springframework.http.MediaType;
import org.springframework.web.multipart.MultipartFile;

public interface WordImageService {
void saveWordImage(Word word, MultipartFile imageFile) throws ImageTransformationException; byte[] getWordImageBytes(long wordId);
MediaType imageMediaType();
void deleteWordImage(Word word);
}
```

WordService.java

```
package org.kpi.easylearn.service;

import org.kpi.easylearn.exception.ImageTransformationException; import org.kpi.easylearn.exception.WordException;
import org.kpi.easylearn.model.User; import org.kpi.easylearn.model.Word;
import org.springframework.web.multipart.MultipartFile; import java.util.List;
public interface WordService {
void createWord(Word word, MultipartFile multipartFile) throws WordException, ImageTransformationException;
void deleteWord(User user, Long id);
```

```
List<Word> getIndexUserWords(User user, int page);
List<Word> getRandomUserWords(User user, int count, boolean imageRequired); int getPagesCount(User user);
}

ImageServiceImpla.java
package org.kpi.easylearn.service.impl; import lombok.AllArgsConstructor;
import lombok.experimental.FieldDefaults;
import org.kpi.easylearn.exception.ImageTransformationException; import org.kpi.easylearn.service.ImageService;
import org.springframework.core.io.ClassPathResource; import org.springframework.http.MediaType;
import org.springframework.stereotype.Service;
import org.springframework.web.multipart.MultipartFile;

import javax.imageio.ImageIO;
import java.awt.image.BufferedImage; import java.io.ByteArrayOutputStream; import java.io.IOException;
import java.nio.file.Files; import java.nio.file.Paths;

import static lombok.AccessLevel.PRIVATE;

@Service
@AllArgsConstructor
@FieldDefaults(level = PRIVATE, makeFinal = true) public class ImageServiceImpl implements ImageService {

static final int IMAGE_SIZE = 256;

static byte[] UNKNOWN_IMAGE_BYTES; static {
byte[] bytes = null; try {
bytes = Files.readAllBytes(Paths.get(new ClassPathResource("static/img/unknown.png").getURI()));
} catch (IOException e) {
throw new RuntimeException("Failed to initialize unknown image bytes", e);
} finally {
UNKNOWN_IMAGE_BYTES = bytes;
}
}



@Override
public byte[] transform(MultipartFile imageFile) throws ImageTransformationException { try {
BufferedImage bufferedImage = ImageIO.read(imageFile.getInputStream()); if (bufferedImage == null) {
throw new ImageTransformationException("Unknown image format");
}

if (bufferedImage.getHeight() < IMAGE_SIZE || bufferedImage.getWidth() < IMAGE_SIZE)
{
throw new ImageTransformationException("Image is too small. Minimum size is " +
IMAGE_SIZE + "x" + IMAGE_SIZE);
}

if (bufferedImage.getWidth() < bufferedImage.getHeight()) {
```

```
int cropTop = (bufferedImage.getHeight() - bufferedImage.getWidth()) / 2; bufferedImage = bufferedImage.getSubimage(0, cropTop, bufferedImage.getWidth(),
bufferedImage.getWidth());
}

if (bufferedImage.getHeight() < bufferedImage.getWidth()) {
int cropLeft = (bufferedImage.getWidth() - bufferedImage.getHeight()) / 2; bufferedImage = bufferedImage.getSubimage(cropLeft, 0,
bufferedImage.getHeight(), bufferedImage.getHeight());
}

BufferedImage resultImage = new BufferedImage(IMAGE_SIZE, IMAGE_SIZE, BufferedImage.TYPE_INT_RGB);



null);


resultImage.getGraphics().drawImage(bufferedImage, 0, 0, IMAGE_SIZE, IMAGE_SIZE,


ByteArrayOutputStream out = new ByteArrayOutputStream(); ImageIO.write(resultImage, "PNG", out);
out.close();
return out.toByteArray();
} catch (IOException e) {
throw new RuntimeException(e);
}
}

@Override
public MediaType mediaType() { return MediaType.IMAGE_PNG;
}

@Override
public byte[] unknownImage() { return UNKNOWN_IMAGE_BYTES;
}

}

QuizServiceImpl.java
package org.kpi.easylearn.service.impl; import lombok.AllArgsConstructor;
import lombok.experimental.FieldDefaults; import org.kpi.easylearn.common.ListUtil;
import org.kpi.easylearn.exception.QuizException; import org.kpi.easylearn.model.Quiz;
import org.kpi.easylearn.model.QuizType; import org.kpi.easylearn.model.User; import org.kpi.easylearn.model.Word;
import org.kpi.easylearn.repository.WordRepository; import org.kpi.easylearn.service.QuizService; import org.kpi.easylearn.service.StatsService;
import org.kpi.easylearn.service.WordService; import org.springframework.stereotype.Service;

import java.util.ArrayList; import java.util.List;

import static lombok.AccessLevel.PRIVATE;
```

```
@Service

@AllArgsConstructor
@FieldDefaults(level = PRIVATE, makeFinal = true) public class QuizServiceImpl implements QuizService {

WordService wordService; StatsService statsService; WordRepository wordRepository;

static int QUIZ_WORDS_COUNT = 20; static int OPTIONS_COUNT = 4;

private List<Word> findOptions(User user, Word currentWord, boolean imageRequired) { List<Word> options =
wordService.getRandomUserWords(user, OPTIONS_COUNT, imageRequired);

if (options.size() != OPTIONS_COUNT) {
throw new QuizException("Not enough words");
}

options.remove(currentWord);
options = options.subList(0, Math.min(options.size(), OPTIONS_COUNT - 1)); ListUtil.randomInsert(options, currentWord);
return options;
}

@Override
public Quiz createQuiz(User user, QuizType quizType) {

if (!isQuizAllowed(user, quizType)) {
throw new QuizException("Quiz is not allowed");
}

Quiz quiz = new Quiz();

quiz.setCorrectWords(new ArrayList<>()); quiz.setWrongWords(new ArrayList<>());

boolean imageRequired = quizType == QuizType.BY_IMAGE;
List<Word> words = wordService.getRandomUserWords(user, QUIZ_WORDS_COUNT, imageRequired); Word currentWord = words.get(0);
words.remove(0);

quiz.setCurrentWord(currentWord); quiz.setRemainingWords(words); quiz.setOptions(findOptions(user, currentWord, imageRequired));
quiz.setType(quizType);

return quiz;
}

@Override
public void nextStep(User user, Quiz quiz, long selectedWordId) {

if (quiz.getCurrentWord().getId() == selectedWordId) { quiz.getCorrectWords().add(quiz.getCurrentWord());
} else {
```

```
quiz.getWrongWords().add(quiz.getCurrentWord());
}

if (!quiz.getRemainingWords().isEmpty()) {
Word currentWord = quiz.getRemainingWords().get(0); quiz.setCurrentWord(currentWord); quiz.getRemainingWords().remove(0);

quiz.setOptions(findOptions(user, currentWord, quiz.getType() == QuizType.BY_IMAGE));
} else {
quiz.setCurrentWord(null); quiz.setOptions(null);

if (!quiz.getCorrectWords().isEmpty()) { wordRepository.incrementCorrectAnswers(quiz.getCorrectWords());
}

if (!quiz.getWrongWords().isEmpty()) { wordRepository.incrementWrongAnswers(quiz.getWrongWords());
}


statsService.recordStats(user);
}
}

@Override
public boolean isQuizAllowed(User user, QuizType quizType) { int wordsCount;
if (quizType == QuizType.BY_TRANSLATION || quizType == QuizType.BY_WORD) { wordsCount = wordRepository.countByUser(user);
} else {
wordsCount = wordRepository.countByUserWithImages(user);
}

return wordsCount >= OPTIONS_COUNT;
}
}
```

StatsServiceImpl.java

```
package org.kpi.easylearn.service.impl; import lombok.AllArgsConstructor;
import lombok.experimental.FieldDefaults;
import org.kpi.easylearn.model.User; import org.kpi.easylearn.model.UserStats;
import org.kpi.easylearn.repository.UserStatsRepository; import org.kpi.easylearn.repository.WordRepository; import org.kpi.easylearn.service.StatsService;
import org.springframework.stereotype.Service;

import java.sql.Timestamp; import java.util.Collections; import java.util.List;
import java.util.stream.Collectors; import static lombok.AccessLevel.PRIVATE;
@Service
@AllArgsConstructor
@FieldDefaults(level = PRIVATE, makeFinal = true) public class StatsServiceImpl implements StatsService {

UserStatsRepository userStatsRepository; WordRepository wordRepository;

@Override
```

```
public List<Integer> getUserLearningRate(User user) { List<Integer> result = userStatsRepository
.findTop10ByUserOrderByTimeDesc(user)
.stream()
.map(UserStats::getLearningRate)
.collect(Collectors.toList()); Collections.reverse(result); return result;
}

@Override
public void recordStats(User user) {

UserStats stats = new UserStats();
stats.setUser(user);
stats.setTime(new Timestamp(System.currentTimeMillis()));
stats.setLearningRate(wordRepository.countLearnedWords(user) * 100 /
wordRepository.countByUser(user));

userStatsRepository.save(stats);
}
}

UserDetailsServiceImpl.java
package org.kpi.easylearn.service.impl; import lombok.AllArgsConstructor;
import lombok.experimental.FieldDefaults;
import org.kpi.easylearn.exception.NotAuthenticatedException; import org.kpi.easylearn.model.User;
import org.kpi.easylearn.repository.UserRepository;
import org.kpi.easylearn.service.CurrentUserResolverService;


import org.springframework.security.authentication.AnonymousAuthenticationToken; import org.springframework.security.core.Authentication;
import org.springframework.security.core.authority.SimpleGrantedAuthority; import
org.springframework.security.core.context.SecurityContextHolder; import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException; import org.springframework.stereotype.Service;

import java.util.Collections;

import static lombok.AccessLevel.PRIVATE;

@Service
@AllArgsConstructor
@FieldDefaults(level = PRIVATE, makeFinal = true)
public class UserDetailsServiceImpl implements UserDetailsService, CurrentUserResolverService { UserRepository userRepository;
static String USER_ROLE = "USER";

private static class UserInfo extends org.springframework.security.core.userdetails.User { private User user;
public UserInfo(User user) {
super(user.getUsername(), user.getPassword(), Collections.singletonList(new SimpleGrantedAuthority(USER_ROLE)));
this.user = user;
}
```

```
public User getUser() { return user;
}
}

@Override
public UserDetails loadUserByUsername(String identifier) throws UsernameNotFoundException { User user = userRepository.findByIdentifier(identifier);
if (user == null) throw new UsernameNotFoundException("User " + identifier + " not found"); return new UserInfo(user);
}

@Override
public User getCurrentUser() {
Authentication authentication = SecurityContextHolder.getContext().getAuthentication(); if (!(authentication instanceof AnonymousAuthenticationToken)) {
return ((UserDetailsServiceImpl.UserInfo) authentication.getPrincipal()).getUser();
} else {
throw new NotAuthenticatedException();
}
}
}

UserServiceImpl.java
package org.kpi.easylearn.service.impl; import lombok.AllArgsConstructor;
import lombok.experimental.FieldDefaults;
import org.kpi.easylearn.exception.RegistrationException; import org.kpi.easylearn.model.User;
import org.kpi.easylearn.repository.UserRepository; import org.kpi.easylearn.service.UserService;
import org.springframework.security.crypto.password.PasswordEncoder; import org.springframework.stereotype.Service;

import static lombok.AccessLevel.PRIVATE;


@Service
@AllArgsConstructor
@FieldDefaults(level = PRIVATE, makeFinal = true) public class UserServiceImpl implements UserService {

UserRepository userRepository; PasswordEncoder passwordEncoder;


@Override
public User register(User user) throws RegistrationException {

if (userRepository.findByIdentifier(user.getUsername()) != null) {
throw new RegistrationException("User with such username already exists");
}

if (userRepository.findByIdentifier(user.getEmail()) != null) {
throw new RegistrationException("User with such email already exists");
}
```

```
user.setPassword(passwordEncoder.encode(user.getPassword()));

return userRepository.save(user);
}
}
```

WordImageServiceImpl.java

```
package org.kpi.easylearn.service.impl; import lombok.AllArgsConstructor;
import lombok.experimental.FieldDefaults;
import org.kpi.easylearn.exception.ImageTransformationException; import org.kpi.easylearn.model.Word;
import org.kpi.easylearn.model.WordImage;
import org.kpi.easylearn.repository.WordImageRepository; import org.kpi.easylearn.repository.WordRepository; import org.kpi.easylearn.service.ImageService;
import org.kpi.easylearn.service.WordImageService; import org.springframework.http.MediaType;
import org.springframework.stereotype.Service;
import org.springframework.web.multipart.MultipartFile; import java.util.Optional;
import static lombok.AccessLevel.PRIVATE;

@Service
@AllArgsConstructor
@FieldDefaults(level = PRIVATE, makeFinal = true)
public class WordImageServiceImpl implements WordImageService {

ImageService imageService; WordImageRepository wordImageRepository; WordRepository wordRepository;

@Override
public void saveWordImage(Word word, MultipartFile imageFile) throws ImageTransformationException {
if (!imageFile.isEmpty()) {
WordImage wordImage = new WordImage(); wordImage.setImageBytes(imageService.transform(imageFile)); wordImage.setWord(word);

wordImageRepository.save(wordImage);
}
}

@Override
public byte[] getWordImageBytes(long wordId) { Optional<Word> word = wordRepository.findById(wordId);

if (!word.isPresent()) {
return imageService.unknownImage();
} else {
WordImage wordImage = wordImageRepository.findByWord(word.get());

byte[] imageBytes;
if (wordImage != null) {
imageBytes = wordImage.getImageBytes();
} else {
imageBytes = imageService.unknownImage();
}
```

```
return imageBytes;
}
}

@Override
public MediaType imageMediaType() { return imageService.mediaType();
}

@Override
public void deleteWordImage(Word word) {
WordImage wordImage = wordImageRepository.findByWord(word); if (wordImage != null) {
wordImageRepository.delete(wordImage);
}
}
}
```

WordServiceImpl.java

```
package org.kpi.easylearn.service.impl; import lombok.AllArgsConstructor;
import lombok.experimental.FieldDefaults;
import org.kpi.easylearn.exception.AccessDeniedException;
import org.kpi.easylearn.exception.ImageTransformationException; import org.kpi.easylearn.exception.WordException;
import org.kpi.easylearn.model.User; import org.kpi.easylearn.model.Word;
import org.kpi.easylearn.repository.WordRepository; import org.kpi.easylearn.service.WordImageService; import org.kpi.easylearn.service.WordService; import org.springframework.data.domain.PageRequest; import org.springframework.stereotype.Service;
import org.springframework.web.multipart.MultipartFile;

import java.util.List; import java.util.Optional;

import static lombok.AccessLevel.PRIVATE;

@Service
@AllArgsConstructor
@FieldDefaults(level = PRIVATE, makeFinal = true) public class WordServiceImpl implements WordService {

WordRepository wordRepository; WordImageService wordImageService;

static int PAGE_SIZE = 20;

@Override
public void createWord(Word word, MultipartFile imageFile) throws WordException, ImageTransformationException {
if (wordRepository.findByUserAndText(word.getUser(), word.getText()) != null) { throw new WordException("Such word already exists");
}

word = wordRepository.save(word); wordImageService.saveWordImage(word, imageFile);
}

@Override
```

```
public void deleteWord(User user, Long id) { Optional<Word> word = wordRepository.findById(id); if (word.isPresent()) {
if (!word.get().getUser().getId().equals(user.getId())) {
throw new AccessDeniedException("Assess denied! Trying to delete not own word!
User: " + user.getId());
}

wordImageService.deleteWordImage(word.get()); wordRepository.delete(word.get());
}
}

@Override
public List<Word> getIndexUserWords(User user, int page) {
return wordRepository.findByUserOrderByTextAsc(user, PageRequest.of(page - 1,
PAGE_SIZE));
}

@Override


public List<Word> getRandomUserWords(User user, int count, boolean imageRequired) { if (imageRequired) {
return wordRepository.findRandomUserWordsWithImages(user, PageRequest.of(0, count));
} else {
return wordRepository.findRandomUserWords(user, PageRequest.of(0, count));
}
}

@Override
public int getPagesCount(User user) {
return Math.max(1, (int) Math.ceil(wordRepository.countByUser(user) / (double)
PAGE_SIZE));
}

}
```

ExceptionController.java

```
package org.kpi.easylearn.controller; import lombok.AllArgsConstructor;
import lombok.experimental.FieldDefaults;
import org.kpi.easylearn.exception.UserAsseptebleException; import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.boot.web.servlet.error.ErrorController; import org.springframework.http.HttpStatus;
import org.springframework.stereotype.Controller; import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping; import javax.servlet.http.HttpServletRequest;
import static lombok.AccessLevel.PRIVATE;

@Controller
@AllArgsConstructor
@FieldDefaults(level = PRIVATE, makeFinal = true)
public class ExceptionController implements ErrorController { Logger log = LoggerFactory.getLogger(this.getClass());
@RequestMapping("/error")
```

```
public String handleError(HttpServletRequest request, Model model) {
int errorCode = (Integer) request.getAttribute("javax.servlet.error.status_code");

if (HttpStatus.valueOf(errorCode).is4xxClientError()) { switch (errorCode) {
case 400: {
model.addAttribute("error", "400, Bad Request"); break;
}
case 401: {
model.addAttribute("error", "401, Unauthorized"); break;
}
case 404: {
model.addAttribute("error", "404, Not found"); break;
}
}
} else {
Exception exception = (Exception) request.getAttribute("javax.servlet.error.exception");

if (exception.getCause() instanceof UserAsseptebleException) { model.addAttribute("error", exception.getCause().getMessage()); log.warn("User acceptable exception", exception);
} else {
log.error("Critical exception", exception);
}
}

return "error";
}

@Override
public String getErrorPath() { return "/error";



}
}

IndexController.java
package org.kpi.easylearn.controller; import lombok.AllArgsConstructor;
import lombok.experimental.FieldDefaults; import org.kpi.easylearn.common.CurrentUser; import org.kpi.easylearn.model.QuizType; import org.kpi.easylearn.model.User;
import org.kpi.easylearn.model.Word;
import org.kpi.easylearn.service.QuizService; import org.kpi.easylearn.service.StatsService; import org.kpi.easylearn.service.WordService; import org.springframework.stereotype.Controller; import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping; import org.springframework.web.bind.annotation.RequestParam;

import java.util.ArrayList; import java.util.List;

import static lombok.AccessLevel.PRIVATE;

@Controller
@AllArgsConstructor
```

```
@FieldDefaults(level = PRIVATE, makeFinal = true) public class IndexController {

WordService wordService; StatsService statsService; QuizService quizService;

static int PAGES_TO_SHOW = 5;

@GetMapping("/")
String indexPage(@RequestParam(required = false, defaultValue = "1") int page,
@CurrentUser User user, Model model) {

int lastPage = wordService.getPagesCount(user);
int fromPage = Math.max(1, page - (PAGES_TO_SHOW / 2) + 1); int toPage = Math.min(lastPage, fromPage + PAGES_TO_SHOW - 1); fromPage =
Math.max(1, toPage - PAGES_TO_SHOW + 1); List<Integer> pages = new ArrayList<>();
for (int p = fromPage; p <= toPage; p++) { pages.add(p);
}

model.addAttribute("lastPage", lastPage); model.addAttribute("currentPage", page); model.addAttribute("pages", pages);

model.addAttribute("user", user);
model.addAttribute("words", wordService.getIndexUserWords(user, page)); model.addAttribute("word", new Word());
model.addAttribute("learningRate", statsService.getUserLearningRate(user));

model.addAttribute("isByWordAllowed", quizService.isQuizAllowed(user, QuizType.BY_WORD));
model.addAttribute("isByTranslationAllowed", quizService.isQuizAllowed(user, QuizType.BY_TRANSLATION));
model.addAttribute("isByImageAllowed", quizService.isQuizAllowed(user, QuizType.BY_IMAGE));

return "index";
}

}
```

QuizController.java

```
package org.kpi.easylearn.controller;

import lombok.AllArgsConstructor;
import lombok.experimental.FieldDefaults; import org.kpi.easylearn.common.CurrentUser; import org.kpi.easylearn.model.Quiz;
import org.kpi.easylearn.model.QuizType; import org.kpi.easylearn.model.User;
import org.kpi.easylearn.service.QuizService; import org.springframework.stereotype.Controller; import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping; import org.springframework.web.bind.annotation.PostMapping; import
org.springframework.web.bind.annotation.RequestParam; import org.springframework.web.bind.annotation.SessionAttribute;

import javax.servlet.http.HttpServletRequest; import static lombok.AccessLevel.PRIVATE;
@Controller
@AllArgsConstructor
```

```
@FieldDefaults(level = PRIVATE, makeFinal = true) public class QuizController {

QuizService quizService;

@GetMapping("/quiz")
public String quizPage(@SessionAttribute(required = false) Quiz quiz,
@CurrentUser User user, Model model) {

if (quiz == null) { return "redirect:/";
}

model.addAttribute("user", user);

int totalWords = quiz.getRemainingWords().size() + quiz.getCorrectWords().size() + quiz.getWrongWords().size() + 1;
int wrongPercent = quiz.getWrongWords().size() * 100 / totalWords; int correctPercent = quiz.getCorrectWords().size() * 100 / totalWords;

model.addAttribute("wrong", wrongPercent); model.addAttribute("correct", correctPercent);


if (quiz.getType() == QuizType.BY_TRANSLATION) { return "quiz-by-translation";
} else if (quiz.getType() == QuizType.BY_WORD) { return "quiz-by-word";
} else if (quiz.getType() == QuizType.BY_IMAGE) { return "quiz-by-image";
} else {
throw new IllegalStateException("Invalid quiz type");
}
}

@GetMapping("/result")
public String resultPage(@SessionAttribute(required = false) Quiz quiz,
@CurrentUser User user, Model model) {

if (quiz == null) { return "redirect:/";
}

model.addAttribute("user", user);

int totalWords = quiz.getCorrectWords().size() + quiz.getWrongWords().size(); int wrongPercent = quiz.getWrongWords().size() * 100 / totalWords;
int correctPercent = 100 - wrongPercent;

model.addAttribute("wrong", wrongPercent); model.addAttribute("correct", correctPercent);

return "result";
}

@PostMapping("/quiz/start")
public String startQuiz(@RequestParam QuizType type,


@CurrentUser User user, HttpServletRequest request) {
```

```
Quiz quiz = quizService.createQuiz(user, type); request.getSession().setAttribute("quiz", quiz);

return "redirect:/quiz";
}

@PostMapping("/quiz")
public String submitAnswer(@RequestParam long answer,
@SessionAttribute Quiz quiz,
@CurrentUser User user, HttpServletRequest request) {

quizService.nextStep(user, quiz, answer); request.getSession().setAttribute("quiz", quiz);

if (!quiz.isFinished()) { return "redirect:/quiz";
} else {
return "redirect:/result";
}
}
}

UserController.java
package org.kpi.easylearn.controller; import lombok.AllArgsConstructor;
import lombok.experimental.FieldDefaults;
import org.kpi.easylearn.exception.RegistrationException; import org.kpi.easylearn.model.User;
import org.kpi.easylearn.model.dto.RegistrationUser; import org.kpi.easylearn.service.UserService; import
org.springframework.stereotype.Controller; import org.springframework.ui.Model;
import org.springframework.validation.BindingResult; import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute; import org.springframework.web.bind.annotation.PostMapping;

import javax.validation.Valid;

import static lombok.AccessLevel.PRIVATE;

@Controller
@AllArgsConstructor
@FieldDefaults(level = PRIVATE, makeFinal = true) public class UserController {

UserService userService;

@GetMapping("/register")
public String registerPage(Model model) { model.addAttribute("user", new RegistrationUser()); return "register";
}

@PostMapping("/register")
public String register(@ModelAttribute @Valid RegistrationUser user, BindingResult bindingResult, Model model) {
if (bindingResult.hasErrors()) { model.addAttribute("error",
bindingResult.getAllErrors().get(0).getDefaultMessage());
model.addAttribute("user", user); return "register";
}
```

```
User u = new User(); u.setUsername(user.getUsername()); u.setEmail(user.getEmail()); u.setPassword(user.getPassword());

try {
userService.register(u);
} catch (RegistrationException e) { model.addAttribute("error", e.getMessage());


model.addAttribute("user", user); return "register";
}

return "redirect:login";
}

@GetMapping("/login") public String loginPage() {
return "login";
}

}
```

WordController.java

```
package org.kpi.easylearn.controller;


import lombok.AllArgsConstructor;
import lombok.experimental.FieldDefaults; import org.kpi.easylearn.common.CurrentUser;
import org.kpi.easylearn.exception.ImageTransformationException; import org.kpi.easylearn.exception.WordException;
import org.kpi.easylearn.model.User; import org.kpi.easylearn.model.Word;
import org.kpi.easylearn.service.WordImageService; import org.kpi.easylearn.service.WordService; import org.springframework.http.HttpEntity;
import org.springframework.http.HttpHeaders; import org.springframework.stereotype.Controller; import org.springframework.ui.Model;
import org.springframework.validation.BindingResult; import org.springframework.web.bind.annotation.*; import org.springframework.web.multipart.MultipartFile;

import javax.validation.Valid;

import static lombok.AccessLevel.PRIVATE;

@Controller
@RequestMapping("/word")
@AllArgsConstructor
@FieldDefaults(level = PRIVATE, makeFinal = true) public class WordController {

WordService wordService; WordImageService wordImageService;
```

```
@PostMapping("/create")
public String createWord(@ModelAttribute @Valid Word word,
BindingResult bindingResult,
@RequestParam MultipartFile imageFile,
@CurrentUser User user) {

if (bindingResult.hasErrors()) {
return "redirect:/?error=" + bindingResult.getAllErrors().get(0).getDefaultMessage()
+ "#dictionary";
}

try {
word.setUser(user); wordService.createWord(word, imageFile); return "redirect:/#dictionary";
} catch (WordException | ImageTransformationException e) { return "redirect:/?error=" + e.getMessage() + "#dictionary";
}
}

@PostMapping("/delete")
public String deleteWord(@RequestParam long id,
@CurrentUser User user) { wordService.deleteWord(user, id);
return "redirect:/#dictionary";
}



@GetMapping("/{id}/img")
@ResponseBody
public HttpEntity<byte[]> image(@PathVariable Long id) { HttpHeaders headers = new HttpHeaders();
headers.setContentType(wordImageService.imageMediaType()); byte[] imageBytes = wordImageService.getWordImageBytes(id);
headers.setContentLength(imageBytes.length);
return new HttpEntity<>(imageBytes, headers);
```

Додаток Г

(обов'язковий)

ПРОТОКОЛ
ПЕРЕВІРКИ МАГІСТЕРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: «Автоматизована система розпізнавання виголошених слів методами машинного навчання»

Тип роботи: магістреська кваліфікаційна робота
(БДР, МКР)

Підрозділ: кафедра АІІТ, ФІІТА, ЗАКІТ-22м
(кафедра, факультет, навчальна група)

Науковий керівник: Гармаш В.В., доц. каф. АІІТ
(прізвище, ініціали, посада)

Показники звіту подібності Unicheck

Оригінальність 89.2% Схожість 10.8%

Аналіз звіту подібності (відмітити потрібне)

■ Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату

☐ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на розгляд експертної комісії кафедри.

☐ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень

Особа, відповідальна за перевірку _____ Роман МАСЛІЙ
(підпис) (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи

Автор _____ Антон БІЛОУС
(підпис) (прізвище, ініціали)

Керівник роботи _____ Володимир ГАРМАШ
(підпис) (прізвище, ініціали)