МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ЦЕНТР ЗАОЧНОЇ, ДИСТАНЦІЙНОЇ ТА ВЕЧІРНЬОЇ ФОРМ НАВЧАННЯ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ВИПУСКНА РОБОТА

на тему:

«Чат-бот з використанням штучного інтелекту і технологій розпізнавання мовлення»

Завідувач
випускаючої кафедри Довбиш А.С.
Керівник роботи Колесніков В.А.
Студента групи ІНз-71с Шамаєв Д.В.

Суми 2021

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Центр заочної, дистанційної і вечірньої форм навчання

Кафедра комп'ютерних наук

Затверджую_____

Зав. кафедрою Довбиш А.С.

"_____" _____2021 г.

ЗАВДАННЯ

до випускної роботи

Студента четвертого курсу, групи ІНз-71с спеціальності "Інформатика" заочної форми навчання Шамаєва Данила Вікторовича.

Тема: "Чат-бот з використанням штучного інтелекту і технологій розпізнавання мовлення "

Затверджена наказом по СумДУ

№_____ от _____2021 г.

Зміст пояснювальної записки: 1) аналітичний огляд інструментів розробки; 2) постановка завдання й формування завдань дослідження; 3) опис основних положень, що використовуються в веб-технологіях 4) розробка інформаційного й програмного забезпечення; 5) аналіз результатів розробки.

Дата видачі завдання "_____"_____2021 г.

Керівник випускної роботи _____ Колесніков В.А.

Завдання прийняв до виконання _____ Шамаєв Д.В.

РЕФЕРАТ

Записка: 40 стор., 21 рис., 1 табл., 1 додаток, 19 джерел.

Об’єкт дослідження — прикладний програмний інтерфейс синтезу та розпізнавання мовлення веб-браузерів.

Мета роботи — розробка веб-застосунку чат-боту з розпізнаванням мовлення.

Методи дослідження — описовий, абстрактно-логічний, порівняльний.

Результати — розроблено програмне забезпечення системи ведення діалогу з розпізнаванням мовлення і синтезом штучного мовлення. Розроблений веб-застосунок реалізовано за допомогою інструментів веб-розробки, які включають мову програмування JavaScript, мову розмітки HTML та формальну мову описання зовнішнього виду документу CSS.

ВЕБ-ЗАСТОСУНОК, РОЗПІЗНАВАННЯ МОВЛЕННЯ, ВЕБ ТЕХНОЛОГІЇ, ШТУЧНЕ МОВЛЕННЯ, ПРИКЛАДНИЙ ПРОГРАМНИЙ ІНТЕРФЕЙС

ЗМІСТ

ВСТУП

Інформаційні технології не стоять на місці. Вони проникли у всі сфери діяльності людини, і використання нових технологій відкриває велику кількість можливостей для людей, яким до цього вони були недоступні, в силу знань або обмежених можливостей. Тому з кожним днем сфери знань в області інформаційних технологій розширяються і пересікаються один з іншим, а спеціалісти з розробки програмного забезпечення постійно вивчають і відкривають для себе нові методи і інструменти.

Веб-технології, безсумнівно, можна назвати однією, якщо не саму, з найбільш об’ємних серед інших областю знань в сфері сучасних інформаційних технологій. З’явившись в початку 90-х років 20 століття, веб-технології значно просунули та популяризували тогочасну мережу Інтернет, але при цьому стандарт мови HTML не мав достатньої уніфікованості і підтримка тих чи інших можливостей значно різнилась між веб-браузерами, і тільки з появою широко поширених стандартів мов CSS і JavaScript, Всесвітня Мережа набула вигляду, близького до сучасного.

Важливим аспектом в поширенні інформаційних технологій серед населення виявилось легкість та зручність в оперуванні програмним забезпеченням пристрою, але іншим, не менш важливим аспектом, було покращення досвіду використання для людей з особливими потребами. Саме такі люди потребують радикального покращення рівня доступності змісту і функціоналу програмного забезпечення.

Голосові помічники, що базуються на штучному інтелекті і технологіях розпізнавання природньої мови, продовжують активне розповсюдження по всій Землі. Майже у кожного користувача смартфону є вбудований голосовий помічник, будь то Siri або Асистент Google. Хоча домашні центри з вбудованими асистентами ще не досягли такого рівню актуальності в Україні як в розвинених країнах, але поза всяких сумнів можна стверджувати що вони

рано чи пізно будуть в майже в кожній сім’ї. З технології майбутнього, розпізнавання мовлення перетворилося в звичайну повсякденність.

Іншою стороною поліпшення та поглиблення галузі штучного інтелекту, а саме машинного навчання і природньої мови, є явище чат-ботів. Це програми, які намагаються зрозуміти користувача, його наміри та контекст розмови і надати підтримку, або виконати дії, по проханню користувача. Такі програми стали дуже в нагоді як малим, так і великим бізнесам, адже підтримка користувачів в багатьох випадках придатна до автоматизації, що економить людські години і затрати на них. На сьогодні вже складно сказати, принаймні, на початку розмови, чи спілкуєтесь в чаті ви з реальною людиною, або програмним кодом, що її імітує.

Отже, метою цієї роботи є розробка веб-застосунку чат-боту і дослідження сучасних методів роботи з розпізнаванням мовлення в браузері.

1 ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

1.1 Вступ

Поняття «веб-застосунку» передбачає насамперед клієнт-серверну архітектуру, в якій клієнт взаємодіє з веб-сервером за допомогою веб-браузеру [1]. Таке розділення сутностей дозволяє використовувати широкий спектр веб-серверів і операційних систем, в яких вони працюють, не зважаючи на клієнтську частину, яка також може працювати в різних браузерних і системних середовищах. Хоча розмежування понять «динамічних веб-сторінок» і «веб-застосунку» було досить розмитим, з впровадженням стандарту HTML5 ці поняття набули біль чітких обрисів: веб-застосунку набули можливості зберігання локальних даних і продовження роботи в офлайн-режимі [2].

1.2 Технології, що застосовуються в клієнтській частині застосунку

Основним стеком технологій, який використовуються для сучасної розробки клієнтської частини веб-застосунків, залишається HTML5, CSS3 та JavaScript (стандарт ES5 та більш нові).

Hyper-Text Markup Language (HTML) – це мова розмітки, яка використовується для опису структури веб-сторінки та визначає її зміст. HTML-документ складається з елементів (тегів) які інтерпретуються браузером і відображаються на веб-сторінці.

Вперше HTML5 був випущений в публічній формі 22 січня 2008 р. Консорціумом Всесвітньої Мережі (W3C), а в 2014 р. отримав значні доповнення і отримав статус «рекомендованого» [3]. Цілями мови HTML5 були:

- впровадити підтримку новітніх медійних та інших функціональних можливостей;
- зберегти легку читабельність як для людини, так і для комп’ютерів і програмного забезпечення – браузерів, парсерів (синтаксичних аналізаторів), тощо;

- і при цьому залишити повну зворотну сумісність зі старим програмним забезпеченням [4].

Cascading Style Sheets (CSS) є мовою таблиць стилів, які описують зовнішній вигляд документу, в свою чергу описаного за допомогою мови розмітки HTML.

CSS розроблена для розділення презентації та змісту документу, включаючи розмітку, шрифти та кольори [5]. Таке розділення дозволяє покращити доступність змісту для користувачів з особливими потребами, надати більше контролю та гнучкості в специфікації характеристик представлення. Не менш важливою особливістю такого підходу є можливість повторного або багаторазового використання стилів між багатьма веб-сторінками, що значно зменшує складність і повторюваність коду, з чого витікає можливість кешування .css файлів задля покращення швидкості завантаження веб-сторінки та економії Інтернет-трафіку.

JavaScript – це високорівнева мультипарадігмальна мова програмування, яка відповідає специфікації ECMAScript [6]. Вона має синтаксис з фігурних дужок, слабке динамічне типування, прототипно-базовану об’єктну орієнтованість та функції першого класу. JavaScript, наряду з HTML і CSS, є одною з основних технологій Всесвітньої мережі. Її використовують в клієнтській частині для програмування поведінки веб-сторінок більш ніж 97% веб-застосунків в мережі. Всі основні браузери використовують той чи інший двигун JavaScript, який інтерпретує та виконує код на пристрої користувача.

1.3 Серверні технології

Веб-сервер – це програмне забезпечення, яке приймає HTTP запити від клієнтів (зазвичай веб-браузерів) і надсилає HTTP відповіді, в яких може міститись HTML документ, медіа файли, поточні медіа дані, тощо.

Веб-фреймворк – це каркас, призначений для створення динамічних веб-сайтів, веб-застосунків, сервісів та інших ресурсів. Такі рішення дозволяють краще структурувати розробку і спрощують роботу с базами даних [7].

Основні рішення для серверної розробки (веб-сервери, веб-фреймворки) на сьогодні це програмна платформа Node.js, фреймворки Django (мова Python), Spring (Java), ASP.NET (C#), Ruby on Rails та Laravel (PHP) [8]. Вони сильно відрізняються як за мовами програмування і середовищами, так і за архітектурними підходами.

Досить часто веб-фреймворки реалізують такі архітектури Model-View-Controller, Model-View-ViewModel, Model-View-Presenter, де компонент Model представлений саме серверною частиною веб-застосунку [9].

1.4 Прикладний програмний інтерфейс розпізнавання і синтезу мовлення

Web Speech API має на меті надати веб-розробникам можливість використовувати у веб-браузері функції розпізнавання (на англ. speech recognition) і синтезу (на англ. text-to-speech) мовлення [10]. Сам API може підтримувати як розпізнавання та синтез на основі сервера, так і на основі клієнта або вбудованого мовлення, специфічного для операційних систем пристроїв. Також передбачено як одноразове розпізнавання мовлення, так і безперервне. Складається з двох компонентів:

- Розпізнавання мовлення. Доступ забезпечується інтерфейсом SpeechRecognition, який в свою чергу забезпечує можливість розпізнавання тексту з вхідного аудіо потоку (наприклад, мікрофон пристрою). Інтерфейс SpeechGrammar надає контейнер для заданого набору граматики, яку веб-застосунок повинен використовувати. Граматика визначається за допомогою JSpeech Grammar Format (**JSGF**).
- Синтез мовлення. Доступ здійснюється за допомогою інтерфейсу SpeechSynthesis, компонент text-to-speech дозволяє застосункам озвучити власний текстовий контент (зазвичай через синтезатор мовлення пристрою). В об’єкті SpeechSynthesisVoice є різні типи голосів, а за

допомогою об’єктів SpeechSynthesisUtterance можна виділяти різні частини тексту [11].

1.5 Сторонні платформи штучного інтелекту

На сьогодні існує досить багато сервісів, які надають послуги з оренди прикладних програмних інтерфейсів з обробки природної мови (NLP). В основі цих API багатокомпонентні системи, здатні відповідати та реагувати на запитання, задані природною мовою, а також навчатися і розуміти контекст діалогу [12]. Серед таких платформ Google Dialogflow, Wit.ai (Facebook), IBM Watson, Microsoft Luis, Kore.ai та інші.

Всі ці платформи об’єднують принципи, за якими працюють системи. Вони мають розпізнавати намір (на англ. intent) користувача, при умові попереднього тренування за допомогою висловів (на англ. utterance) і виділяти з запиту сутності (на англ. entities), при цьому зберігаючи контекст розмови між запитами. Виділення сутностей, таких як, наприклад, дата, час або місце, дозволяє оперувати ними як змінними і реалізовувати необхідні дії [13].

2 ВИБІР МЕТОДУ РІШЕННЯ

2.1 Методи розробки клієнтської частини

Мова програмування JavaScript передбачає гнучкий підхід до розробки, не стримуючи розробників в одній парадигмі. Але слід зазначити, що є загальноприйняті стандарти написання вихідного коду програмного забезпечення на мові JavaScript, які передбачають використання стандарту ECMAScript 6 та більш сучасних.

Зі стандартом ECMAScript 6 прийшло досить багато нововведень. Одними з самих помітних є нові ключові слова let/const для оголошення змінних та шаблонні рядки.

На табл. 2.1.1 можна побачити порівняння ключових слів для оголошення змінних.

	var	let	const
Обмежується областю блоку коду	ні	так	так
Піднімається	так	ні	ні
Прив'язується до this	Тільки до глобального об'єкту	ні	ні
Дозволена передекларація	так	ні	ні
Дозволена переініціалізація	так	так	ні

Таблиця 2.1.1 – Порівняння ключових слів var, let і const

Явище піднімання (з англ. hoisting) являє собою підняття оголошення змінної в самий початок функції незалежно від місця оголошення і притаманно лише ключовому слову var.

Ключове слово const забороняє змінювати значення змінної після декларації, але не забороняє проводити операції над змістом об’єкту, тому як в змінних в мові JavaScript об’єкти зберігаються лише по посиланню, а не по значенню.

2.2 Методи та інструменти розробки серверу

В якості веб-серверу було обрано Node.js з фреймворком Express та пакетним менеджером npm. Для обміну повідомленнями між клієнтською та серверною частинами використовувалась бібліотека Socket.IO.

Node.js – це програмна платформа, яка працює на JavaScript двигуні V8, який використовується в веб-браузерах для трансляції JavaScript в машинний код. Серед переваг, які надає Node.js:

- низьке використання фізичних ресурсів серверу через специфічний асинхронний та багатопотоковий (тільки ввід-вивід) дизайн платформи;
- велика база зовнішніх бібліотек та готових модулів;
- відносна легкість у побудуванні веб-застосунку з допомогою готових фреймворків;
- єдина мова програмування (JavaScript) як для клієнтської, так і серверної частин застосунку [14].

Через те, що за стандартом Node.js має лише базові інструменти, доцільніше використовувати фреймворк для веб-застосунків – це додає потрібний рівень абстракції, що зекономить час, не потребуючи додаткових зусиль на розробку. Де-факто стандартом такого фреймворку є Express [15]. Він достатньо мінімалістичний і гнучкий, при цьому надає всі необхідні методи та інструменти для створення веб-застосунку в цьому випадку.

Express додає ряд корисних зручностей для HTTP-серверу Node.js, абстрагуючись далеко від його складності. Наприклад, надсилання одного

файлу JPEG досить складне у вихідному файлі Node.js (особливо якщо брати в увагу продуктивність); Express зменшує його до одного рядка [16]. Це дозволяє переформатувати одну функцію обробки монолітного запиту на багато менших обробників запитів, які обробляють лише певні шматочки. Це більш гнучкий та більш модульний підхід.

Для управління пакетами (модулями) і залежностями, Node packet manager (npm) - найпопулярніше рішення і має найбільшу базу пакетів [17]. Слід зазначити що у npm є аналог – пакетний менеджер yarn, який також використовує базу пакетів npm, але при цьому має дещо інший підхід до керування пакетами та залежностями. Yarn більш новий і не має такої великої спільноти і документації як у npm.

Socket.IO – подієво-орієнтована JavaScript-бібліотека, що використовується в веб-застосунках для обміну повідомленнями (даними) в реальному часі за допомогою протоколу WebSocket (хоча лише ним не обмежується) [18]. Socket.IO працює майже однаково як в клієнтському, так і в серверному (Node.js) середовищах.

WebSocket – протокол зв’язку, що працює поверх TCP-з’єднання і призначений для обміну повідомленнями між браузером і веб-сервером в режимі реального часу. Хоча WebSocket розроблений в першу чергу для роботи в веб-середовищі, він може використовуватись в будь яких клієнтських та серверних застосунках [19].

2.3 Платформа обробки природної мови

В якості компоненту штучного інтелекту застосунку було обрано платформу сприйняття природної мови Google Dialogflow, через важливі переваги, які вона надає:

- широкий безкоштовний тариф для розробників програмного забезпечення;
- кількість та якість документації;

- готові до використання агенти (див. далі), що позитивно вплинуло на швидкість розробки.

Агент (в контексті Dialogflow) – модуль обробки природної мови, що сприймає людське мовлення та представляє його у вигляді структурованого набору даних для подальшого використання в застосунках.

Intent відображає намір кінцевого користувача за один хід розмови [13]. Коли користувач висловлює запит, Dialogflow підбирає намір, найбільш близький до користувацького запиту.

Контексти Dialogflow близькі до таких, що наявні в природній мові. Наприклад, коли людина каже «вони білі», треба розуміти контекст, щоби визначити на кого вказується у вислові. Таким же чином працює Dialogflow – для того щоб точніше визначати намір користувача, потрібно мати контекст.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Визначення вимог та проєктування

На початку роботи було визначено перелік функціональних вимог, яким повинний відповідати веб-застосунок:

1. Головна сторінка повинна мати інтерактивну кнопку початку мовлення.
2. Після завершення мовлення користувачем, сказаний текст повинен з’явитися у вигляді тексту на сторінці.
3. В результаті бот повинен відповісти на запит користувача.
4. Формат змісту розмови – загальний та повсякденний.

В процесі проєктування було визначено схему, яка демонструє архітектуру веб-застосунку (рис. 3.1.1).

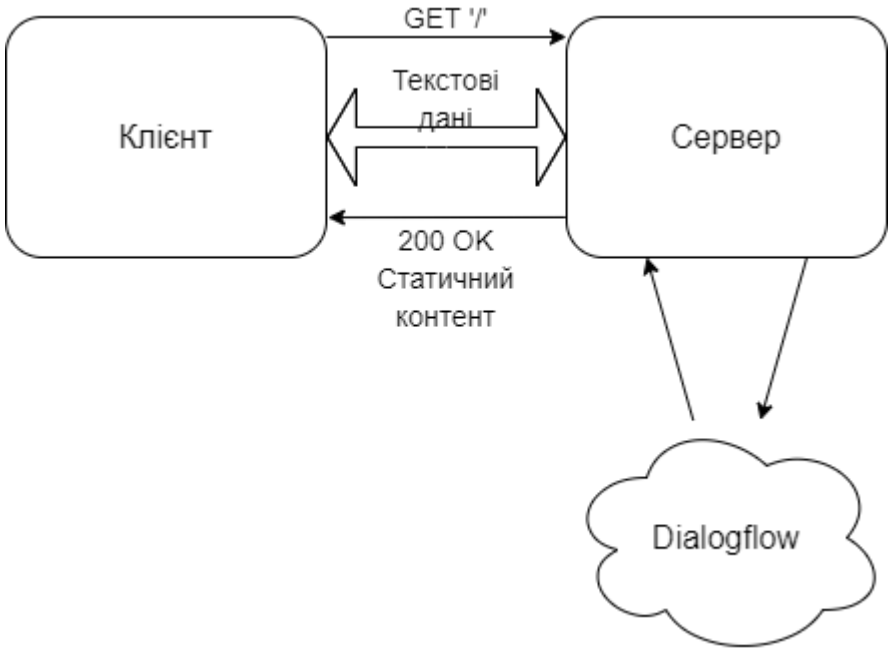

Рисунок 3.1.1 – Схематичне відображення архітектури веб-застосунку.

Початкова структура директорій і файлів продемонстрована на рис. 3.1.2.

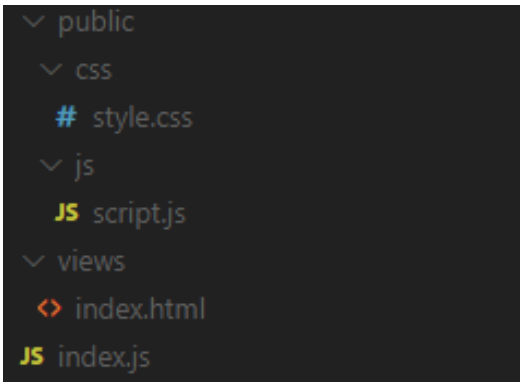

Рисунок 3.1.2 – початкова структура проєкту.

3.2 Ініціалізація проєкту

Перед початком безпосередньої розробки була потреба ініціювати npm проєкт, виконавши наступну команду в корені проєкту:

$ npm init -y

В результаті був створений файл package.json, який містить метадані проєкту (рис. 3.2.1), де:

- “name” – ім’я проєкту (пакету);
- “version” – версія пакету;
- “description” – опис пакету;
- “main” – основний файл пакету, який виступає точкою входу;
- “scripts” – набір сценаріїв терміналу, які будуть використовуватись на протязі життєвого циклу проєкту;
- “keywords” – масив ключових слів пакету;
- “author” – автори пакету;
- “license” – назва ліцензії, за якою пакет розповсюджується.

Більшість цих даних не має потреби редагувати, тому як проєкт не планується для публікації в базі пакетів npm.

```
C:\Users\danyl\Desktop\бакалавр\chatbot-test>npm init -y
Wrote to C:\Users\danyl\Desktop\бакалавр\chatbot-test\package.json:

{
  "name": "chatbot-test",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

Рисунок 3.2.1 – результат виконання npm init.

Наступним кроком було додано всі потрібні бібліотеки (залежності) до проєкту, за допомогою команди:

$ npm install express socket.io @google-cloud/dialogflow dialogflow-fulfillment –save

```
C:\Users\danyl\Desktop\бакалавр\chatbot-test>npm install express socket.io @google-cloud/dialogflow
dialogflow-fulfillment --save

> protobufjs@6.11.2 postinstall C:\Users\danyl\Desktop\бакалавр\chatbot-test\node_modules\protobufjs

> node scripts/postinstall

npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN notsup Unsupported engine for dialogflow-fulfillment@0.6.1: wanted: {"node":"6"} (current:
{"node":"14.16.1","npm":"6.14.12"})
npm WARN notsup Not compatible with your version of node/npm: dialogflow-fulfillment@0.6.1
npm WARN dialogflow-fulfillment@0.6.1 requires a peer of actions-on-google@^2.4.1 but none is instal
led. You must install peer dependencies yourself.
npm WARN chatbot-test@1.0.0 No description
npm WARN chatbot-test@1.0.0 No repository field.

+ socket.io@4.1.2
+ dialogflow-fulfillment@0.6.1
+ @google-cloud/dialogflow@3.5.0
+ express@4.17.1
added 149 packages from 143 contributors and audited 149 packages in 12.202s

4 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

Рисунок 3.2.2 – Результат виконання команди npm install

Далі потрібно було прописати сценарій запуску веб-серверу у файлі package.json. Для цього до об’єкту scripts додано значення “start”: “node index.js”.

Рисунок 3.2.3 – Фінальний вигляд файлу package.json.

Отже, ініціалізація та конфігурація проєкту була виконана успішно.

3.3 Реалізація веб-серверу

При розробці серверної частини завдання полягало в тому, щоб отримати в результаті веб-сервер, який буде одночасно відправляти користувачам за запитом індексну сторінку, відповідати на події, що були викликані користувачем та відправляти запити до стороннього сервісу, який представляв компонент штучного інтелекту.

Отже, на початку було визначено об’єкти, з якими буде працювати веб-сервер (рис. 3.3.1), де:

```
JS index.js > ...
1   const express = require('express');
2   const app = express();
3   const dialogflow = require('@google-cloud/dialogflow');
4   const sessionClient = new dialogflow.SessionsClient();
5
6   app.use(express.static(__dirname + '/views'));
7   app.use(express.static(__dirname + '/public'));
8
9   const server = app.listen(80);
10  const { Server } = require('socket.io');
11  const io = new Server(server);
12
```

Рисунок 3.3.1 – основні об’єкти веб-серверу.

- app - представляє Express-застосунок і містить відповідні методи;
- app.use(express.static(…)) - шляхи до статичних файлів застосунку, які необхідно надавати клієнту;
- io - сервер socket, який відповідає за роботу з повідомленнями;
- sessionClient - клієнт сесії Dialogflow, який надає можливість працювати з відповідним сервісом.

Так як розробка виконувалась в середовищі ОС Windows 10 на локальному комп’ютері, виникла потреба визначити додаткові константи, необхідні для роботи клієнту сесії Dialogflow, а саме (рис. 3.3.2):

- projectId – внутрішній ідентифікатор проєкту Google Cloud, який дозволяє звертатися до модулю Dialogflow;
- langCode – код мови, на якій виконується звернення;
- GOOGLE_APPLICATION_CREDENTIALS – змінна середовища, яка визначає шлях до .json файлу з аутентифікаційними даними Google Cloud.

```
12
13  const projectId = 'newagent-gusc';
14  const langCode = 'en';
15  process.env
16  .GOOGLE_APPLICATION_CREDENTIALS = "C:\\Users\\danyl\\Desktop\\newagent-gusc-dce419ea77c6.json"
17
```

Рисунок 3.3.2 – параметри звернень та середовища.

Наступним кроком був визначений шлях для індексної сторінки, яка повинна надсилатись користувачу за запитом (рис. 3.3.3), де res – об’єкт відповіді серверу, sendFile() – метод, що надсилає користувачу сторінку чи будь який інший документ.

```
17
18  app.get('/', (req, res) => {
19    res.sendFile('index.html');
20  });
21
```

Рисунок 3.3.3 – функція, яка визначає відповідь на запит шляху.

Останнім етапом реалізації серверної частини було визначення функції, яка отримуватиме запити користувача та визначення функцій, які надсилають запити через посередництво клієнта сесії Dialogflow.

При з’єднанні з користувачем (функція io.on(‘connection’, …)), сервер очікує повідомлення (функція socket.on(‘chat message’, …)), і у разі відповіді – виконує запит з текстом повідомлення за допомогою функції executeQueries() (рис. 3.3.4).

```
21
22  io.on('connection', function(socket) {
23    socket.on('chat message', (text) => {
24      console.log(`User Message: ${text}`);
25
26      let exQ = executeQueries(projectId, socket.id, [text], langCode);
27
28      exQ.then(text => {
29        socket.emit('bot reply', text);
30      });
31
32      exQ.catch(error => {
33        socket.emit('bot reply', 'Sorry, could you repeat?');
34      });
35    });
36  });
37
```

Рисунок 3.3.4 – функція-обробник, яка реагує на з’єднання з користувачем і подальші повідомлення.

У разі успішної відповіді викликається функція emit(), яка надсилає користувачеві текст відповіді. У разі помилки, наприклад, втраченому зв’язку, або будь якій іншій непередбачуваній ситуації, надсилається загальна відповідь з текстом «Sorry, could you repeat?».

Функції, що працюють с запитами, виглядають наступним чином (рисунки 3.3.5 та 3.3.6): функція executeQueries() отримує в якості аргументів ідентифікатор проєкту, ідентифікатор сесії, масив з рядками текстів запитів і код мови, після чого кожен текст запиту передається функції detectIntent(), яка приймає в якості аргументів ідентифікатор проєкту, ідентифікатор сесії, текст запиту, контекст і код мови та повертає результат виконання запиту. Для зручності відстежування процесу роботи веб-серверу всі запити виводяться в термінал за допомогою функції console.log().

```
async function executeQueries(
  projectId,
  sessionId,
  queries,
  langCode
) {
  let context; // збереження контексту між запитами
  let intentResponse;
  for (let query of queries) {
    try {
      console.log(`Sending Query: ${query}`);
      intentResponse = await detectIntent(
        projectId,
        sessionId,
        query,
        context,
        langCode
      );
      console.log(`Fullfilment: ${
        intentResponse.queryResult.fulfillmentText
      }`);
      // Використаємо контекст для наступних запитів
      context = intentResponse.queryResult.outputContexts;
      return intentResponse.queryResult.fulfillmentText;
    } catch (error) {
      console.log(error);
      return error;
    }
  }
}
```

Рисунок 3.3.5 – функція, що виконує запити.

```
async function detectIntent(
  projectId,
  sessionId,
  query,
  contexts,
  langCode
) {
  // Шлях до ідентифікації агента, якому належить намір (intent)
  const sessionPath = sessionClient.projectAgentSessionPath(
    projectId,
    sessionId
  );

  const request = {
    session: sessionPath,
    queryInput: {
      text: {
        text: query,
        languageCode: langCode,
      },
    },
  };

  if (contexts && contexts.length > 0) {
    request.queryParams = {
      contexts: contexts,
    };
  }

  const responses = await sessionClient.detectIntent(request);
  return responses[0];
}
```

Рисунок 3.3.6 – функція, що виявляє наміри, виходячи с запиту та контексту.

3.4 Реалізація клієнтської частини

Клієнтська частина була реалізована в три етапи: розмітка сторінки, опис стилів і розробка функціоналу.

Розмітка сторінки складається з:

- тегу <head>, який включає теги з шляхами до стороннього сервісу піктограм Font Awesome і шляхом до файлу опису стилів style.css (рисунок 2.4.1)
- тегу <body>, який складається з елементу <button>, що відповідає за кнопку початку мовлення, блоку , що містить текст, який озвучив

користувач та текст відповіді боту, а також тегів <script>, що відповідають за включення до веб-сторінки функціоналу, визначеного мовою програмування JavaScript (рис. 3.4.2).

```
<!DOCTYPE html>
<html>
  <head>
    <script src="https://kit.fontawesome.com/0a54f4cbb9.js" crossorigin="anonymous"></script>
    <link rel="stylesheet" href="css/style.css">
  </head>
```

Рисунок 3.4.1 – зміст тегу <head>.

```
  <body>
    <button>
      <i class="fa fa-microphone"></i>
    </button>
    <div>
        <p>You said: <em class="output-you">...</em></p>
        <p>Bot replied: <em class="output-bot">...</em></p>
    </div>

    <script src="/socket.io/socket.io.js"></script>
    <script src="js/script.js"></script>
  </body>
```

Рисунок 3.4.2 – зміст тегу <body>.

Опис стилів включає в себе всі відображені елементи і детально наведений в Додатку.

Останнім етапом в розробці клієнтської частини була реалізація функціональної складової. Вихідний код знаходиться в файлі script.js і складається з:

- директиви ‘use strict’, яка переводить інтерпретацію в строгий режим;
- визначення об’єкту socket, який містить необхідні клієнтські функції для обміну повідомленнями з сервером через бібліотеку socket.io;
- визначення об’єктів елементів класів .output-you і .output-bot, які представляють текстовий зміст мовного виразу користувача і відповідь на запит боту відповідно;
- об’єкт recognition класу SpeechRecognition, що містить функції, які обробляють мовлення користувача;

- визначення властивостей об’єкту recognition.

Зазначені властивості (рис. 2.4.3) потрібні для коректної конфігурації поведінки об’єкту і відповідають за:

- lang – мову, яка обробляється (англійська, Сполучені Штати);
- interimResults – повернення проміжного результату, коли мовлення ще не закінчилось (встановлено false);
- maxAlternatives – максимальну кількість альтернатив сприйняття того чи іншого виразу (встановлено 1).

```
8   const recognition = new SpeechRecognition();
9
10  recognition.lang = 'en-US';
11  recognition.interimResults = false;
12  recognition.maxAlternatives = 1;
```

Рисунок 3.4.3 – об’єкт recognition та його властивості.

Функціонування елементів забезпечується прив’язкою функцій до відповідних подій. Прив’язка здійснюється викликом функції addEventListener() об’єкту, який викликає подію. Ця функція приймає в якості аргументу назву події і функцію, яка виконається, якщо подія відбудеться.

На рис. 3.4.4 продемонстровано фрагмент коду, де при події «click» (коли користувач натискає на елемент) елементу button викликається функція recognition.start(), що починає процес розпізнавання мовлення.

```
13
14  document.querySelector('button').addEventListener('click', () => {
15    recognition.start();
16  });
17
```

Рисунок 3.4.4 – прив’язка виклику функції до події.

Після закінчення мовлення користувача, в об’єкті recognition відбувається подія «result», обробник якої транслює повідомлення «chat message» з текстом результату розпізнавання мовлення (рис. 3.4.5).

```
recognition.addEventListener('result', (event) => {
  console.log('DEBUG: Result detected');

  let last = event.results.length - 1;
  let text = event.results[last][0].transcript;

  outputYou.textContent = text;
  console.log('DEBUG: Confidence = ' + event.results[0][0].confidence);

  socket.emit('chat message', text); // передаємо повідомлення серверу
});
```

Рисунок 3.4.5 – функція-обробник події «result».

Озвучування тексту відбувалося за допомогою виклику функції synthVoice() (рис. 3.4.6). При виклику створюється об’єкт вислову класу SpeechSynthesisUtterance, якому у властивості .text присвоюється відповідний текстовий зміст (відповідь боту), після чого об’єкт передається функції speak() глобального об’єкту window.speechSyntesis.

```
function synthVoice(text) {
    const utterance = new SpeechSynthesisUtterance();
    utterance.text = text;
    window.speechSynthesis.speak(utterance);
}
```

Рисунок 3.4.6 – функція, що викликає програвання синтезованого мовлення.

3.5 Тестування

Після всіх етапів планування та реалізації, останнім етапом було тестування готового веб-застосунку. Для цих цілей веб-сервер був запущений на локальному комп’ютері. Веб-сторінка перевірялась у веб-браузері Google Chrome версії 90.0.4430.212.

Результати тестування продемонстровані на рисунках 3.5.1 – 3.5.3.

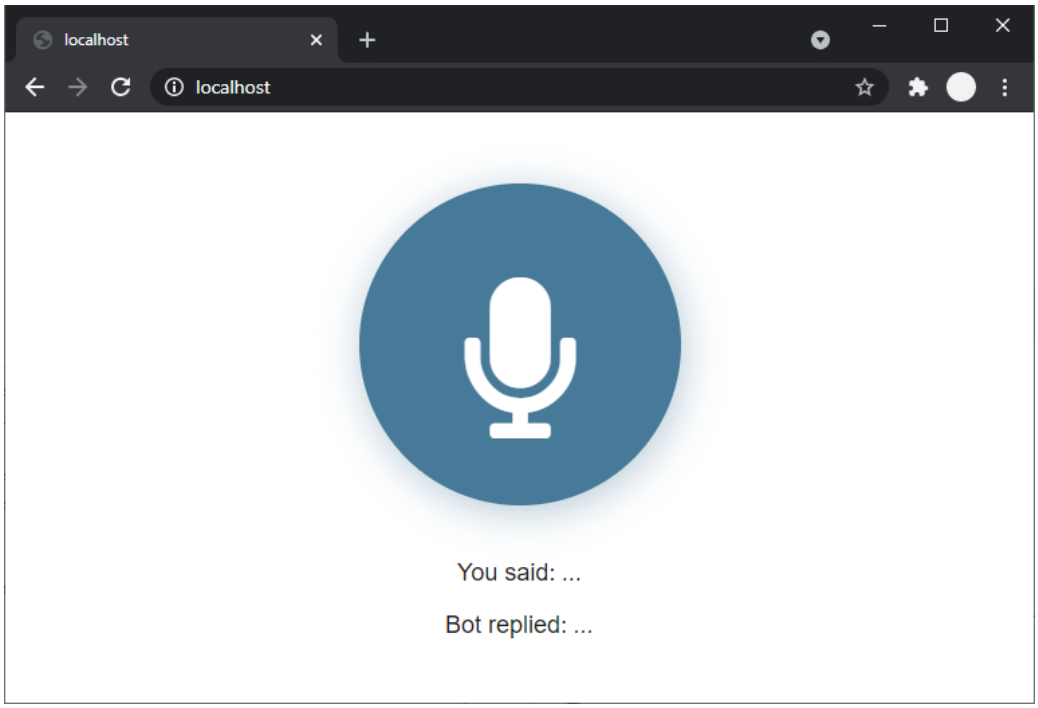

Рисунок 3.5.1 – Вид головної сторінки додатку.

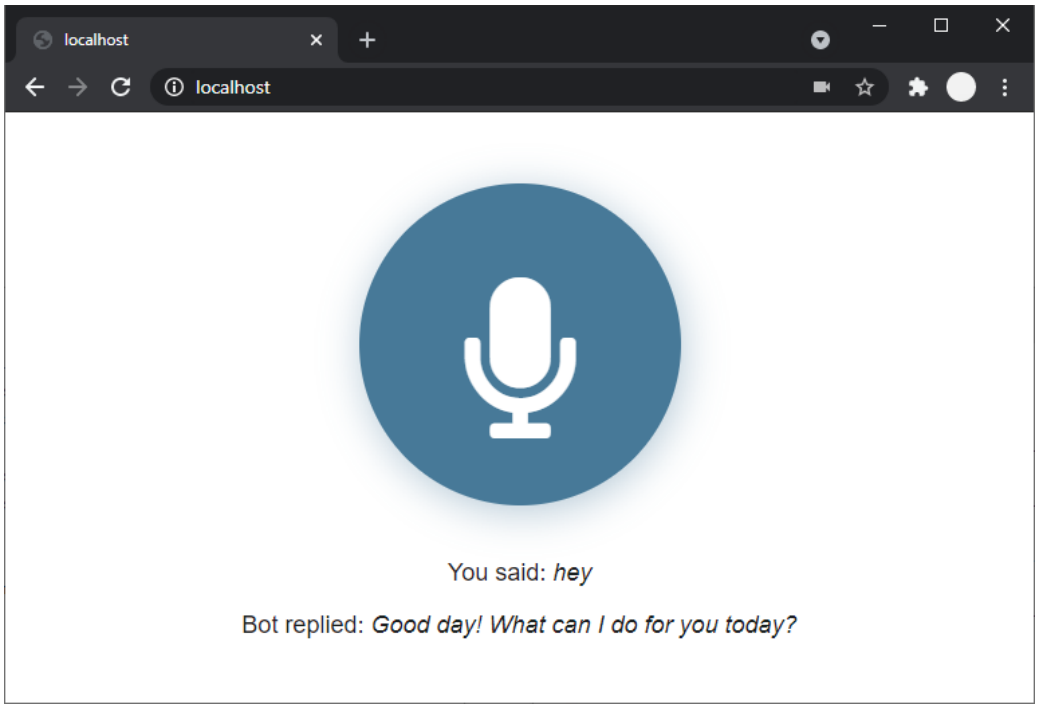

Рисунок 3.5.2 – Реакція боту на привітання.

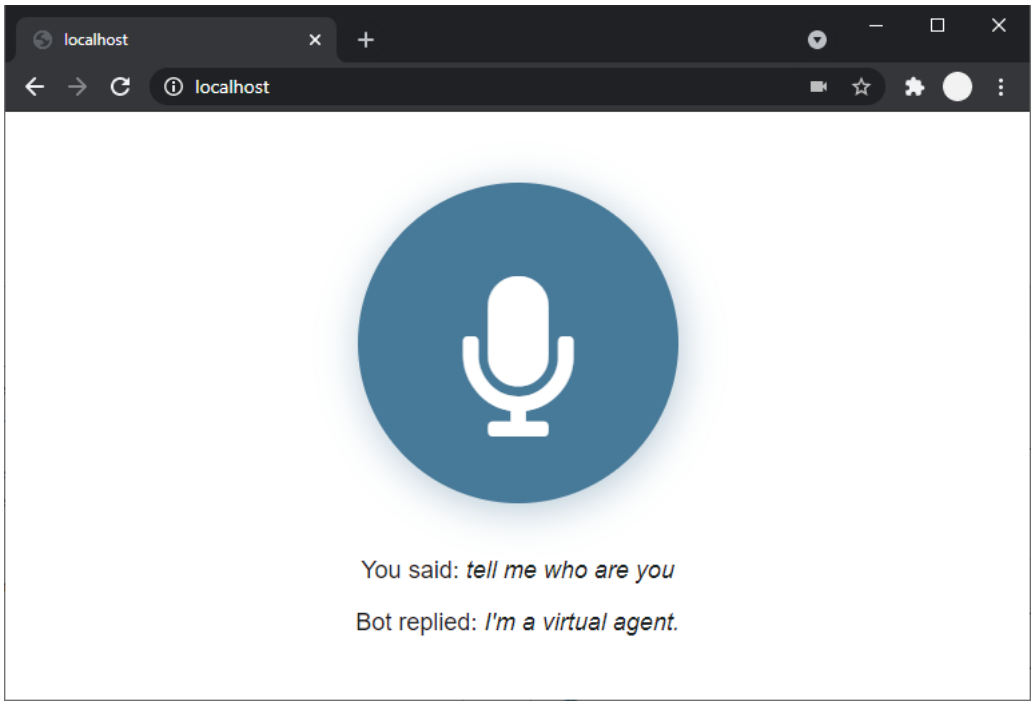

Рисунок 3.5.3 – Відповідь боту на запитання.

ВИСНОВКИ

В даній випускній роботі було проаналізовано сучасний прикладний програмний інтерфейс синтезу та розпізнавання мовлення в веб-браузерах, його інтерфейси і функції, на прикладі реального веб-застосунку.

В ході роботи було проведено збір та структуризація матеріалів про сучасну розробку веб-застосунків, аналіз інструментів і готових рішень з областей клієнтських і серверних технологій.

Створено працюючий веб-застосунок чат-боту у вигляді розмовного інтерфейсу користувача з використанням прикладного програмного інтерфейсу синтезу та розпізнавання мовлення, платформи штучного інтелекту, а саме розпізнавання природньої мови і відповідей на запитання.

Наступним кроком у розвитку чат-боту має бути детальніша розробка сценаріїв використання, що визначить більш широкі функціональні можливості застосунку з метою вдосконалення користувацького досвіду.

СПИСОК ЛІТЕРАТУРИ

1. Shklar, Leon; Rosen, Richard; Dow Jones and Company. Web Application Architecture: Principles, protocols and practices. Chichester: John Wiley & Sons Ltd, 2003. 357 pp.
2. World Wide Web Consortium. Offline Web Applications // World Wide Web Consortium (W3C). 2008. URL: https://www.w3.org/TR/offline-webapps/
3. World Wide Web Consortium. HTML5 is a W3C Recommendation | W3C News // World Wide Web Consortium (W3C). 2014. URL: https://www.w3.org/blog/news/archives/4167
4. World Wide Web Consortium. HTML5 Differences from HTML4 // World Wide Web Consortium (W3C). 2014. URL: https://www.w3.org/TR/2014/NOTE-html5-diff-20141209/
5. World Wide Web Consortium. HTML & CSS // World Wide Web Consortium (W3C). 2020. URL: https://www.w3.org/standards/webdesign/htmlcss#whatcss
6. Ecma International. ECMAScript® 2022 Language Specification // TC39 – Specifying JavaScript. 2021. URL: https://tc39.es/ecma262/#sec-overview
7. DocForge. Web application framework - DocForge // DocForge. 2018. URL: https://web.archive.org/web/20151024013648/http://docforge.com/wiki/Web_application_framework
8. StackOverflow. StackOverflow Developer Survey 2019 // StackOverflow. 2019. URL: https://insights.stackoverflow.com/survey/2019#technology-_-web-frameworks
9. Plekhanova J, "Evaluating web development frameworks: Django, Ruby on Rails and CakePHP," Temple University, Philadelphia, Науково-дослідна робота ISSN, 2009.
10. Contributors to the Web Speech API Specification. Web Speech API // Web Speech API. 2020. URL: https://wicg.github.io/speech-api/

11. MDN Contributors. Using the Web Speech API - Web APIs | MDN [Електроний ресурс] // MDN Web Docs: [сайт]. [2021]. URL: https://developer.mozilla.org/en-US/docs/Web/API/Web_Speech_API/Using_the_Web_Speech_API
12. Goyal P., Pandey S., та Jain K. Deep Learning for Natural Language Processing: Creating Neural Networks with Python. Bangalore: Apress, Inc., 2018. 277 pp.
13. Google. Intents | Dialogflow ES | Google Cloud // Google Cloud. 2021. URL: https://cloud.google.com/dialogflow/es/docs/intents-overview
14. Brown E. Web Development with Node and Express. Sebastopol (California): O'Reilly Media, Inc., 2014. 306 pp.
15. Serby P. Case study: How & why to build a consumer app with Node.js // VentureBeat. 2012. URL: https://venturebeat.com/2012/01/07/building-consumer-apps-with-node/
16. Hahn E. Express in Action. New York: Manning Publications Co., 2016. 236 pp.
17. // npm Docs: [сайт]. [2021]. URL: https://docs.npmjs.com/about-npm
18. Krill P. Socket.IO JavaScript framework ready for real-time apps // InfoWorld. 2014. URL: https://www.infoworld.com/article/2607757/socket-io-javascript-framework-ready-for-real-time-apps.html
19. Шестаков В.С., Сагидуллин А.С., "ПРИМЕНЕНИЕ ТЕХНОЛОГИИ WEBSOCKET В WEB-ПРИЛОЖЕНИЯХ ТЕХНОЛОГИЧЕСКОГО НАЗНАЧЕНИЯ.," // Приборостроение, апрель 2015. С. 3.

ДОДАТОК

1 Вихідний код файлу index.js

```
const express = require('express');
const app = express();
const dialogflow = require('@google-cloud/dialogflow');
const sessionClient = new dialogflow.SessionsClient();

app.use(express.static(__dirname + '/views'));
app.use(express.static(__dirname + '/public'));

const server = app.listen(80);
const { Server } = require('socket.io');
const io = new Server(server);

const projectId = 'newagent-gusc';
const langCode = 'en';
process.env
.GOOGLE_APPLICATION_CREDENTIALS =
"C:\\Users\\danyl\\Desktop\\newagent-gusc-dce419ea77c6.json"

app.get('/', (req, res) => {
  res.sendFile('index.html');
});

io.on('connection', function(socket) {
  socket.on('chat message', (text) => {
    console.log(`User Message: ${text}. Session ID: ${socket.id}`);

    let exQ = executeQueries(projectId, socket.id, [text], langCode);
```

```
    exQ.then(text => {
      socket.emit('bot reply', text);
    });

    exQ.catch(error => {
      socket.emit('bot reply', 'Sorry, could you repeat?');
    });
  });
});

async function detectIntent(
  projectId,
  sessionId,
  query,
  contexts,
  langCode
) {
  // Шлях до ідентифікації агента, якому належить намір (intent)
  const sessionPath = sessionClient.projectAgentSessionPath(
    projectId,
    sessionId
  );

  const request = {
    session: sessionPath,
    queryInput: {
      text: {
        text: query,
```

```
        languageCode: langCode,
      },
    },
  };

  if (contexts && contexts.length > 0) {
    request.queryParams = {
      contexts: contexts,
    };
  }

  const responses = await sessionClient.detectIntent(request);
  return responses[0];
}

async function executeQueries(
  projectId,
  sessionId,
  queries,
  langCode
) {
  let context; // збереження контексту між запитами
  let intentResponse;
  for (let query of queries) {
    try {
      console.log(`Sending Query: ${query}`);
      intentResponse = await detectIntent(
        projectId,
        sessionId,
```

```
        query,
        context,
        langCode
      );
      console.log(`Fullfilment: ${
        intentResponse.queryResult.fulfillmentText
      }`);
      // Використаємо контекст для наступних запитів
      context = intentResponse.queryResult.outputContexts;
      return intentResponse.queryResult.fulfillmentText;
    } catch (error) {
      console.log(error);
      return error;
    }
  }
}
```

2 Вихідний код index.html

```
<!DOCTYPE html>
<html>
  <head>
    <script src="https://kit.fontawesome.com/0a54f4cbb9.js"
crossorigin="anonymous"></script>
    <link rel="stylesheet" href="css/style.css">
  </head>
  <body>
    <button>
      <i class="fa fa-microphone"></i>
    </button>
```

```
    <div>
        <p>You said: <em class="output-you">...</em></p>
        <p>Bot replied: <em class="output-bot">...</em></p>
    </div>

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

3 Вихідний код style.css

```
html {
  height: 100%;
}

body {
  padding: 0;
  margin: 0;
  font-family: "Arial", sans-serif;
  font-weight: 300;
  font-size: 18px;
  color: #1B2021;
}

button {
  display: block;
  width: 240px;
  height: 240px;
  border: 1px;
```

```
  border-radius: 50%;
  padding: 0.75em 1em;
  margin: 4em auto 3em;
  text-align: center;
  background-color: #477998;
  box-shadow: 2px 5px 30px rgba(71, 121, 152, 0.4);
  will-change: transform, filter;
  transition: all 0.25s ease-out;
  color: #fff;
}

button:hover {
  transform: scale(0.95);
}

button:active {
  filter: brightness(0.85);
}

button:focus {
  outline: 0;
}

button .fa {
  font-size: 120px;
  line-height: 240px;
  margin: 0;
}
```

```
p {
  text-align: center;
}
```

4 Вихідний код script.js

```
'use strict';
const socket = io();

const outputYou = document.querySelector('.output-you');
const outputBot = document.querySelector('.output-bot');

const SpeechRecognition = window.SpeechRecognition ||
window.webkitSpeechRecognition;
const recognition = new SpeechRecognition();

recognition.lang = 'en-US';
recognition.interimResults = false;
recognition.maxAlternatives = 1;

document.querySelector('button').addEventListener('click', () => {
  recognition.start();
});

recognition.addEventListener('speechstart', () => {
  console.log('DEBUG: Speech start.');
});

recognition.addEventListener('result', (event) => {
  console.log('DEBUG: Result detected');
```

```
  let last = event.results.length - 1;
  let text = event.results[last][0].transcript;

  outputYou.textContent = text;
  console.log('DEBUG: Confidence = ' + event.results[0][0].confidence);

  socket.emit('chat message', text); // передаємо повідомлення серверу
});

recognition.addEventListener('speechend', () => {
  recognition.stop();
});

recognition.addEventListener('error', (event) => {
  outputBot.textContent = `Error! ${event.error}`;
});

function synthVoice(text) {
  const utterance = new SpeechSynthesisUtterance();
  utterance.text = text;
  window.speechSynthesis.speak(utterance);
}

socket.on('bot reply', function(replyText) {
  synthVoice(replyText);
  outputBot.textContent = replyText == '' ? '(No answer...)' : replyText;
});
```