МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

ЛЕСИК РОМАН ОЛЕГОВИЧ

Допускається до захисту:
в.о. завідувача кафедри
інформаційних технологій
канд. техн. наук, доцент
_____ О. В. Зелінська
«____»_____ 20__ р.

РОЗРОБКА ГОЛОСОВОГО АСИСТЕНТУ З ВИКОРИСТАННЯМ NODE.JS ТА REACT

Спеціальність 122 Комп’ютерні науки

Кваліфікаційна (бакалаврська) робота

Керівник:
І. В. Богач, доцент кафедри
інформаційних технологій,
к.т.н., доцент
_____

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

Вінниця – 2024

АНОТАЦІЯ

Лесик Р.О. Розробка голосового асистенту з використанням Node.js та React. Спеціальність 122 «Комп'ютерні науки», освітня програма «Комп'ютерні науки». Донецький національний університет імені Василя Стуса, Вінниця 2024.

У кваліфікаційній (бакалаврській) роботі досліджено та проаналізовано поняття голосового асистенту, пояснений основний принцип роботи та його створення. За допомогою таких технологій як: API, HTML 5, CSS 3 / SCSS, JavaScript, Node.js було розроблено програмне забезпечення голосового асистенту мета якого – допомога людині у повсякденних справах.

Ключові слова: додаток, голосовий асистент, Node.js, React.

85 ст. 20 рис., 2 дод., 29 джерел.

ABSTRACT

Lesyk R.O. Development of a Voice Assistant Using Node.js and React. Specialty 122 «Computer Science», educational program «Computer Science». Vasyl Stus Donetsk National University, Vinnytsia 2024.

In the qualification (bachelor's) thesis, the concept of a voice assistant is researched and analyzed, and the main principle of its operation and creation is explained. Using technologies such as API, HTML 5, CSS 3 / SCSS, JavaScript, and Node.js, software for a voice assistant was developed with the goal of assisting individuals in their daily tasks.

Keywords: application, voice assistant, Node.js, React.

ЗМІСТ

ВСТУП

Актуальність теми дослідження. Актуальність теми дослідження полягає в тому, що в сучасному світі голосові асистенти стають все більш популярними. З кожним роком зростає кількість користувачів, які віддають перевагу голосовим командам для взаємодії з пристроями та сервісами. Це зумовлено тим, що голосові асистенти можуть значно спростити виконання багатьох повсякденних завдань, таких як пошук інформації в Інтернеті, керування розумним будинком, організація розкладу та багато іншого [1-5].

Розробка зручного додатку стає викликом через велику кількість конкурентів, однак потрібно постійно вдосконалювати існуючі аналоги, які повинні поєднувати простоту та зручність використання. Ключовим аспектом є інтуїтивно зрозумілий інтерфейс, який дозволяє користувачам швидко та ефективно взаємодіяти з асистентом без тривалого навчання.

Тому, розробка голосового асистента є актуальним для покращення життя багатьох людей. Голосові асистенти можуть допомогти людям з обмеженими можливостями, забезпечуючи їм доступ до інформації та сервісів без необхідності використання клавіатури або миші. Вони також можуть стати незамінними помічниками для зайнятих професіоналів, надаючи можливість швидко отримувати необхідні дані або виконувати завдання за допомогою простих голосових команд. Отже, впровадження та вдосконалення голосових асистентів відкриває нові перспективи для технологічного розвитку та покращення якості життя.

Мета роботи полягає у створенні клієнт-серверного застосунку голосового асистенту, що буде допомагати людині спростити рутинні справи та зможе мати можливість більш простого додавання нового функціоналу в порівнянні з іншими аналогами.

Для досягнення поставленої мети потрібно вирішити наступні **завдання:**

1. Провести аналіз предметної області: визначити підстави для створення додатку; визначити вимоги до додатку.

2. Проаналізувати ринок голосових асистентів та обрати найуживаніші запити.

3. Проаналізувати ринок інструментів розробки та обрати відповідні до вимог;

4. Розробити функціонуючу систему на основі обраних технологій

5. Протестувати розроблений програмний продукт

Об'єкт дослідження - процеси проектування та реалізації голосових асистентів.

Предмет дослідження - програмні засоби для розробки програмних продуктів, голосових асистентів.

Теоретичне та практичне значення отриманих результатів полягає у розробці голосового асистенту з використанням Node.js та React, що допомагає людині спростити рутинні справи, просте у користуванні і має можливість додавання нового функціоналу.

Отримані результати можуть бути використані для подальшого розвитку голосових асистентів та клієнтських частин інших веб-застосунків .

Апробація результатів дослідження

Результати роботи було представлено на Всеукраїнській науково-практичній Інтернет-конференції студентів, аспірантів та молодих науковців «Молодь в науці: Дослідження, проблеми, перспективи (МН-2024)». [9]

Структура кваліфікаційної (бакалаврської) роботи

Робота складається зі вступу, трьох розділів, логічного висновку, списку використаних джерел з 29 джерел включаючи інтернет-ресурси, офіційну документацію обраних технологій, довідників та посібників та 20 рисунків. Загальний обсяг основної частини роботи - 64 сторінки.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Поняття голосового асистенту

Голосовий асистент - це програмне забезпечення або пристрій, який використовує голосове введення користувача для виконання різноманітних завдань та надання інформації. Цей інтерфейс дозволяє людям взаємодіяти з комп'ютером, смартфоном, планшетом або іншими електронними пристроями за допомогою свого голосу.

Голосові асистенти зазвичай використовують технології розпізнавання мови та штучного інтелекту для розуміння та обробки команд користувачів. Вони можуть відповідати на запитання, надавати інформацію про погоду, новини, розклад транспорту, а також виконувати функції керування пристроями в домашньому середовищі, такими як включення світла чи регулювання температури. Такі системи забезпечують користувачів можливістю здійснювати голосові дзвінки, надсилати повідомлення, встановлювати нагадування та таймери, керувати музикою та іншими мультимедійними функціями.

Популярні голосові асистенти включають такі системи, як Siri від Apple, Google Assistant, Amazon Alexa, Microsoft Cortana та інші. Вони стають все більш поширеними і надають користувачам зручний спосіб взаємодіяти з технологіями у повсякденному житті.[14] Наприклад, Siri інтегрована в операційну систему iOS, що дозволяє користувачам iPhone виконувати різні завдання за допомогою голосових команд. Google Assistant пропонує широку інтеграцію з сервісами Google, такими як пошук, календар та Google Maps. Amazon Alexa відома своєю здатністю керувати розумним домом за допомогою пристроїв Echo.

Використання голосових асистентів постійно розширюється завдяки розвитку технологій обробки природної мови (NLP) та машинного навчання. Це дозволяє асистентам більш точно розпізнавати голосові команди, навіть у шумному середовищі, та адаптуватися до індивідуальних потреб користувачів. Крім того, зростає роль голосових асистентів у забезпеченні доступності

технологій для людей з обмеженими можливостями, надаючи їм можливість здійснювати взаємодію з електронними пристроями без необхідності використання фізичних інтерфейсів.

Таким чином, голосові асистенти стають важливою складовою сучасного технологічного середовища, сприяючи зручності та ефективності взаємодії користувачів з різноманітними цифровими сервісами та пристроями.

1.2 Історія створення голосових асистентів

Голосові асистенти були вигадані для полегшення взаємодії людей з електронними пристроями та комп'ютерами. Основна мета полягає у створенні зручного та ефективного інтерфейсу, який дозволяє користувачам використовувати свій голос для керування пристроями та отримання необхідної інформації.

Основні причини виникнення голосових асистентів:

Зручність використання: Голосовий інтерфейс є одним з найбільш природних для людей способів взаємодії. Він дозволяє користувачам використовувати свій голос для виконання різноманітних завдань без необхідності вводити текст або використовувати складні інтерфейси. Ця простота робить технології більш доступними для широкого кола користувачів.

Підвищення продуктивності: Голосові асистенти можуть допомогти користувачам ефективно виконувати завдання, швидко отримувати потрібну інформацію та керувати пристроями, що дозволяє їм бути більш продуктивними. Наприклад, замість ручного введення запиту в пошукову систему, користувач може просто озвучити його, що економить час і зусилля.

Доступність: Голосові асистенти можуть стати незамінними для людей з обмеженнями в русі, зорі або іншими фізичними обмеженнями, оскільки вони дозволяють взаємодіяти з технологіями без необхідності використання рук. Це робить технології більш інклюзивними та відкриває нові можливості для людей з інвалідністю.

Технологічний прогрес: Розвиток технологій розпізнавання мови та штучного інтелекту дозволяє створювати все більш потужні та інтелектуальні голосові асистенти, які можуть краще розуміти та відповідати на запити користувачів. Це підвищує точність та ефективність взаємодії, роблячи асистентів ще більш корисними у повсякденному житті.

Інтеграція з різноманітними сервісами: Голосові асистенти можуть бути інтегровані з широким спектром сервісів і пристроїв, включаючи розумні будинки, автомобілі, мобільні додатки та багато іншого. Це створює єдину екосистему, де користувачі можуть керувати різними аспектами свого життя за допомогою одного інтерфейсу.

Персоналізація: Голосові асистенти можуть навчатися від користувачів, адаптуючись до їхніх звичок та уподобань. Це робить взаємодію більш персоналізованою та інтуїтивно зрозумілою, підвищуючи задоволеність користувачів.

Отже, голосові асистенти були створені для забезпечення зручного та ефективного способу взаємодії людей з електронними пристроями та комп'ютерами, підвищення продуктивності та доступності технологій для всіх користувачів. [15] Їх розвиток базується на поєднанні новітніх досягнень у сфері розпізнавання мови та штучного інтелекту, що дозволяє створювати інтерактивні, адаптивні та інтелектуальні системи. Це відкриває нові перспективи для їх використання у різних сферах життя, від побутових завдань до професійної діяльності, роблячи технології більш доступними та корисними для всіх категорій користувачів.

1.3 Тенденції росту кількості голосових помічників

Існує велика кількість помічників для різних типів запитів, починаючи від введення тексту у Google до оплати покупки у магазині.(рис.1.3). Дослідження показують, що користувачі голосових асистентів використовують їх для

широкого спектру задач. Наприклад, 91% користувачів ставлять запитання, 89.5% слухають музику, а 85.2% дивляться погоду (рис. 1.1).

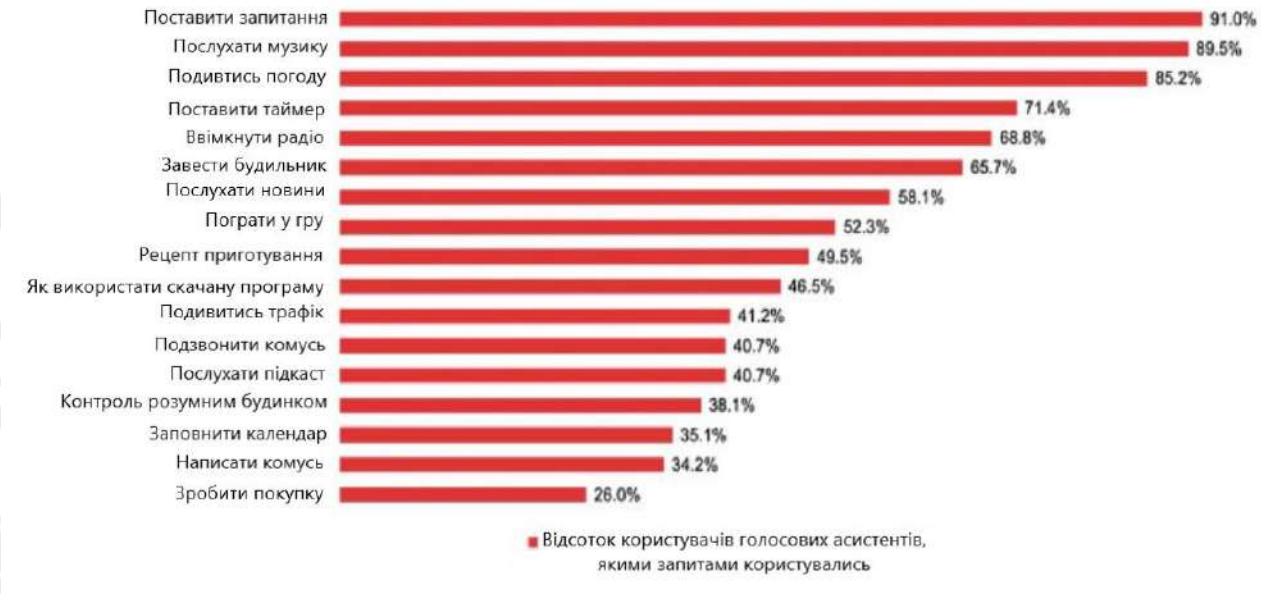

Рисунок 1.1 – Запити користувачів ГА у відсотках

У зв’язку з поширенням коронавірусу в 2020 році попит на пристрої, які дозволяють управляти "розумними" будинками за допомогою голосу, зросла внаслідок пандемії на 30 відсотків.

Згідно з дослідженням американської компанії ABI Research у питанні споживчого ринку, зростаючий попит пов'язаний із страхом людей перед коронавірусом, який передається через поверхні, а не тільки від людини до людини.

Дослідники Лондонського університету "Метрополітен" стверджують, що побоювання людей не безпідставні. Публічні термінали з тачскрином (рис. 1.2) є носієм великої кількості мікробів та бактерій.

Рисунок 1.2 – Приклад терміналу з тачскрином.

Наприклад, на екранах терміналів для реєстрації в аеропорту знаходиться приблизно 40 тисяч одиниць мікробів на кожному квадратному сантиметрі. Компанії в усьому світі, а також в Україні розробляють голосові технології для вирішення проблеми фізичного контакту з публічними пристроями[27].

Голосові асистенти стають все більш популярними через їх здатність забезпечувати безконтактну взаємодію, що є особливо важливим в умовах пандемії. Це зменшує ризик передачі вірусів і забезпечує безпечнішу альтернативу традиційним методам взаємодії з пристроями.

1.4 Порівняння з існуючими аналогами

З розвитком технологій голосові помічники стають невід'ємною частиною нашого повсякденного життя. Серед найпопулярніших на ринку - Google Assistant та Siri. Обидва пропонують широкий спектр функцій, але обидва мають свої відмінності, тому слід їх розглянути.

Користувацький інтерфейс. Інтерфейс користувача відіграє ключову роль у взаємодії між користувачем і віртуальним помічником. Siri та Google Assistant

мають різні підходи до реалізації інтерфейсу, кожен з яких має свої переваги і недоліки.

Siri пропонує простий та інтуїтивно зрозумілий інтерфейс. Мінімалістичний дизайн робить його зручним для використання та навігації. Великий акцент робиться на голосових командах, що забезпечує узгоджений користувацький досвід на різних пристроях Apple.

Серед функцій Siri – наявність ярликів та швидких дій, які дозволяють користувачам легко виконувати повсякденні завдання. Однак, для повного використання всіх можливостей інтерфейсу, може знадобитися певний час на налаштування.

Google Assistant, навпаки, надає більш візуально багатий та інтерактивний інтерфейс, який можна налаштувати відповідно до вподобань користувача. Він пропонує ряд віджетів та карток, які адаптуються до контексту використання, відображаючи релевантну інформацію та дії.

Асистент від Google також підтримує різні методи введення: голосові команди, жести, малювання на екрані та навіть використання міміки для взаємодії. Така гнучкість дозволяє користувачам вибирати найбільш зручний спосіб взаємодії, але може бути складною для новачків через велику кількість опцій.

Отже, Siri забезпечує простоту та зручність у використанні, підходячи тим, хто цінує мінімалізм і прямолінійність. Google Assistant пропонує більш гнучкий і налаштовуваний інтерфейс, ідеальний для тих, хто хоче максимізувати функціональність та індивідуалізацію. Вибір між ними залежить від особистих уподобань та потреб у користуванні.

Крім того, варто враховувати і різницю в підходах до захисту приватності даних користувачів. Siri, як продукт Apple, відома своєю високою рівнем захисту особистих даних. Компанія активно працює над забезпеченням конфіденційності користувачів та вживає заходів для мінімізації обробки особистої інформації. Наприклад, обробка запитів користувачів може відбуватися без прив'язки до конкретного облікового запису, що забезпечує анонімність.

У випадку з Google Assistant, зазвичай використовується активна обробка даних, що може включати аналіз поведінки користувача для покращення рекомендацій та персоналізованої інформації. Це може викликати обурення серед тих, хто стежить за своєю приватністю, хоча Google також надає можливість управління рівнем доступу до даних та видаляння їх.

Також варто звернути увагу на екосистему та інтеграцію з іншими сервісами. Siri має глибоку інтеграцію з екосистемою Apple, що дозволяє користувачам зручно використовувати його на пристроях Apple, таких як iPhone, iPad, а також взаємодіяти з додатками та сервісами компанії. З іншого боку, Google Assistant має широкі можливості інтеграції з продуктами Google, такими як Gmail, Календар та Хром, що робить його важливим інструментом для тих, хто активно використовує сервіси Google.

Отже, при виборі між Siri та Google Assistant важливо враховувати не лише їхні функціональні можливості та інтерфейс, але й підходи до захисту даних, екосистему та інтеграцію з іншими сервісами. Користувачам варто зважити усі ці аспекти, щоб знайти найбільш підходящий варіант для своїх потреб та вимог.

Мовна підтримка. Мовна підтримка є критично важливим аспектом віртуальних цифрових помічників (VDA), оскільки визначає їхню доступність і корисність для користувачів з різних регіонів та мов.

Siri підтримує понад 20 мов, включаючи англійську, китайську, французьку, німецьку та іспанську. Завдяки цьому Siri охоплює більшість основних мов, якими розмовляють у всьому світі, роблячи його корисним інструментом для багатьох користувачів. Проте, для менш поширених мов або діалектів мовна підтримка Siri може бути обмеженою.

Google Assistant підтримує понад 30 мов, включаючи англійську, іспанську, французьку, німецьку, японську та китайську. Завдяки широкій мовній підтримці, Google Assistant є більш універсальним і корисним для користувачів, які говорять менш поширеними мовами або діалектами.

Крім голосового введення, Google Assistant також підтримує читання і написання кількома мовами. Це робить його відмінним інструментом для перекладу та спілкування, допомагаючи долати мовні бар'єри.

Отже, обидва помічники пропонують відмінну мовну підтримку для основних світових мов, але Google Assistant має перевагу в підтримці більшої кількості мов та діалектів. Це робить його більш підходящим для користувачів, які шукають багатомовний інструмент для повсякденного використання та перекладу. Siri, з іншого боку, забезпечує високу якість підтримки для популярних мов, що може бути достатнім для багатьох користувачів.

Розпізнавання голосу. Розпізнавання голосу є ключовою функцією віртуальних цифрових помічників (VDA), і як Siri, так і Google Assistant використовують передові технології для розуміння голосових команд користувача.

Siri застосовує обробку природної мови (NLP) та алгоритми машинного навчання для розпізнавання голосу користувача та відповідей на його команди. Коли користувач говорить із Siri, аудіовхід спочатку обробляється пристроєм та перетворюється на цифрові дані. Потім Siri аналізує ці дані за допомогою NLP, щоб визначити наміри користувача на основі слів і фраз, що використовуються.

Алгоритми машинного навчання, якими користується Siri, дозволяють поступово підвищувати точність розпізнавання голосу, оскільки система навчається на взаємодіях з користувачем. Це забезпечує більш персоналізований і точний користувацький досвід з часом.

Google Assistant використовує передові алгоритми штучного інтелекту для обробки введення природною мовою та надання точних відповідей. Система розпізнавання мовлення Google Assistant є дуже складною, використовуючи алгоритми глибокого навчання для аналізу мовних шаблонів і визначення намірів користувача. Ця система може розрізняти різні акценти та діалекти, що робить її більш точною та інклюзивною для користувачів з різних регіонів.

Окрім розпізнавання мовлення, Google Assistant також здатний враховувати контекст запиту, що дозволяє надавати більш релевантні відповіді. Це робить взаємодію з Google Assistant ще більш природною та ефективною.

Отже, і Siri, і Google Assistant мають потужні можливості розпізнавання голосу, але Google Assistant може мати перевагу завдяки своїм більш складним алгоритмам глибокого навчання та здатності обробляти різні акценти і діалекти. Siri, у свою чергу, також надає високий рівень точності завдяки використанню NLP і машинного навчання, що дозволяє створити персоналізований користувацький досвід. Обидва помічники підтримують кілька мов, але Google Assistant пропонує більш широку мовну підтримку, що робить його більш доступним для глобальної аудиторії.

Стороння інтеграція. Інтеграція зі сторонніми додатками та сервісами є важливим аспектом віртуальних цифрових помічників (VDA), оскільки дозволяє користувачам безперешкодно отримувати доступ до різноманітних функцій та керувати ними.

Siri має потужну екосистему сторонніх програм, таких як банківські сервіси, сервіси спільного використання поїздок та обмін повідомленнями, які можна інтегрувати з VDA. Siri також добре інтегрується з HomeKit від Apple, який підтримує різні пристрої розумного будинку, дозволяючи користувачам керувати побутовою технікою за допомогою голосових команд.

Ця інтеграція робить Siri зручною для користувачів, які вже перебувають в екосистемі Apple, оскільки забезпечує безперебійну взаємодію між різними пристроями та додатками.

Google Assistant пропонує більш відкриту екосистему, яка забезпечує широку інтеграцію з різноманітними додатками та сервісами. Користувачі можуть підключати Google Assistant до різних сторонніх програм, таких як сервіси потокової передачі музики, програми доставки їжі та системи домашньої автоматизації.

Google Assistant також інтегрується з екосистемою Google Nest, яка включає різні пристрої розумного будинку від багатьох виробників. Це дозволяє

користувачам керувати своїми пристроями за допомогою голосових команд, незалежно від бренду або виробника.

Отже, Siri надає надійну інтеграцію зі сторонніми додатками, особливо добре працюючи в межах екосистеми Apple. Це робить його ідеальним вибором для користувачів, які вже використовують пристрої Apple і хочуть максимально ефективно їх використовувати. Google Assistant, з іншого боку, пропонує більшу гнучкість завдяки своїй відкритій екосистемі, що дозволяє інтегруватися з більш широким спектром додатків і сервісів. Це робить його більш універсальним і підходящим для користувачів, які шукають широку функціональність та сумісність з різними пристроями.

1.5 Висновок до розділу

Обидва голосові помічники, Siri та Google Assistant, постійно вдосконалюються, пропонуючи унікальні переваги кожен по-своєму. Вибір між ними залежить від ваших індивідуальних потреб та вподобань. Siri відмінно інтегрується в екосистему Apple, забезпечуючи простий та інтуїтивно зрозумілий інтерфейс, тоді як Google Assistant пропонує більшу гнучкість та ширшу підтримку мов і додатків. [10] Обидва голосові помічники, Siri та Google Assistant, не тільки забезпечують базові функції відтворення музики, планування подій, виконання пошуку в Інтернеті та інші, але й постійно вдосконалюються, щоб забезпечити користувачам ще більше функцій та зручності.

Ciri відмінно інтегрується в екосистему Apple, що означає, що вона має прямий доступ до інших сервісів та пристроїв від Apple, таких як iPhone, iPad, Apple Watch та інші. Це дозволяє їй забезпечувати глибшу інтеракцію з користувачем та забезпечувати персоналізований досвід. Наприклад, ви можете використовувати Siri для нагадувань, керування медіа-програвачем або навіть керування освітленням у вашому будинку за допомогою пристроїв HomeKit.

З іншого боку, Google Assistant відкритий для різних пристроїв та платформ, не обмежуючись лише у своїй власній екосистемі. Він забезпечує

більшу гнучкість, оскільки може працювати на пристроях з різними операційними системами, включаючи Android, iOS та різні версії веб-переглядачів. Це робить його вибором для тих, хто користується різними пристроями і платформами або шукає більш універсальне рішення.

Що стосується мовної підтримки, як ми вже згадували, Google Assistant має перевагу завдяки своїй широкій підтримці понад 30 мов. Це робить його більш універсальним і доступним для користувачів з різних країн та мовних спільнот.

Таким чином, вибір між цими двома голосовими помічниками залежить від ваших індивідуальних потреб та вподобань. Якщо ви в основному користуєтеся пристроями від Apple та цінуєте глибоку інтеграцію, то Siri може бути кращим варіантом для вас. Але якщо вам потрібна більша гнучкість і широкі можливості, то Google Assistant може відповісти на ваші потреби краще.

Потрібно розробити голосовий асистент, щоб він мав простий та зрозумілий інтерфейс, історію запитів та був зорієнтований на найчастіше уживані запити користувачів. На відміну від Siri та Google Assistant додаток повинен мати можливість вдосконалюватись не лише розробником, а й самим користувачем, щоб дало змогу постійно підлаштовувати додаток під власні потреби, стаючи максимально зручним та унікальним.

РОЗДІЛ 2. ВИБІР ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА ІНСТРУМЕНТІВ ДЛЯ РОЗРОБКИ ГОЛОСОВОГО АСИСТЕНТУ

2.1 Основні поняття клієнт-серверної архітектури

Під час розробки проекту було використано клієнт-серверну архітектуру (рис 2.1). У ролі сервера виступав Node.js файл, а в якості клієнта я використовував фреймворк React, а потім за допомогою webpack скомпілював проект у єдине ціле.

Клієнт-серверна архітектура додатку полягає у розділенні програми на дві основні частини: клієнтську і серверну. Клієнтська частина відповідає за інтерфейс користувача, тоді як серверна забезпечує потрібну функціональність та управління даними.

Цей тип архітектури є стандартним для веб-додатків і багатьох інших програм, які потребують обміну даними між користувачами. Додатки, які використовують цю архітектуру, часто використовують різні протоколи передачі даних, такі як HTTP, FTP, SMTP та інші.

Клієнтська частина програми забезпечує взаємодію з користувачем, а серверна - обробку запитів. Сервер може бути веб-сервером, базою даних або іншим сервером, який виконує функції та надсилає результати клієнту.У цій архітектурі можуть також бути додаткові компоненти, такі як мережеві пристрої, сервіси зберігання даних тощо.

Взаємодія між клієнтом і сервером здійснюється через протоколи передачі даних, де клієнт надсилає запити, а сервер оброблює їх та надсилає результати назад.

Клієнт-серверна архітектура має переваги, такі як ефективне використання ресурсів та безпека даних, але також має недоліки, такі як залежність від доступності сервера та проблеми з масштабованістю. [8]

Рисунок 2.1 – Приклад клієнт-серверної архітектури

2.2 Використання Rest API

Задля поєднання клієнта та сервера було вирішено відійти від застосування звичайного патерну REST API. Натомість, я застосував технологію сокет з’єднань, адже це дало мені можливість розвернути сервер доволі швидко та встановити зв’язок між клієнтом та сервером у реальному часі. Завдяки цьому я зміг досягти того, щоб голосовий асистент мав змогу виконувати певні команди на ПК користувача.

Socket з'єднання (англ. socket connection) - це механізм, який дозволяє двом процесам обмінюватися даними через мережу. Він базується на використанні ідентифікаторів (адрес) та протоколів, що визначають формат даних, їх послідовність і спосіб передачі через мережу.

Socket з'єднання використовується в клієнт-серверних застосунках, де клієнтський процес і серверний процес спілкуються один з одним за допомогою мережі. Для цього клієнтський процес створює сокет (socket), який ідентифікує кінцеву точку з'єднання, а серверний процес очікує з'єднання з вказаного сокету.

Існує два типи сокетів: сокети інтернет-домену (Internet domain sockets) та сокети UNIX-домену (Unix domain sockets).

Сокети інтернет-домену використовуються для з'єднання між процесами на різних комп'ютерах через мережу Інтернет або локальну мережу. Для цього використовуються протоколи, такі як TCP/IP, UDP, ICMP тощо. Сокети інтернет-домену ідентифікуються за допомогою IP-адреси та номера порту.

Сокети UNIX-домену використовуються для з'єднання між процесами на одному комп'ютері. Вони ідентифікуються за допомогою файлових дескрипторів.

Для встановлення з'єднання між сокетами використовуються функції стандартної бібліотеки мов програмування, такі як socket(), bind(), listen(), accept(), connect() тощо.

Socket з'єднання може бути забезпечене безпекою за допомогою шифрування та автентифікації, наприклад, за допомогою протоколу SSL / TLS.

Socket з'єднання є важливим механізмом для розробки клієнт-серверних додатків, таких як веб-сервери, поштові сервери, месенджери тощо. Вони дозволяють взаємодіяти з клієнтами через мережу і передавати дані, які можуть бути текстовими повідомленнями, файлами, фотографіями, відео тощо.

Після встановлення з'єднання між сокетами, процеси можуть обмінюватися даними. Для цього використовуються різні методи передачі даних, такі як read() та write() для звичайних сокетів, або send() та recv() для сокетів, що використовують протокол TCP/IP. Крім того, можуть бути використані різні механізми для відправки та отримання повідомлень, наприклад, JSON, XML, protobuf тощо.

Socket з'єднання може бути використане як у синхронному, так і у асинхронному режимі. У синхронному режимі процес блокується, поки не буде отримано відповідь на запит. У асинхронному режимі процес може продовжувати виконуватися, поки не буде отримано відповідь на запит. [18]

Одним з найбільш поширених використань socket з'єднання є HTTP-запити, які використовуються для передачі даних між веб-браузером та веб-сервером. HTTP-запити передаються через TCP/IP-сокети, що забезпечує надійну та безпечну передачу даних через мережу.

Узагальнюючи, socket з'єднання - це важливий механізм для забезпечення взаємодії між процесами через мережу. Використання сокетів дозволяє передавати дані між процесами незалежно від їх місцезнаходження та операційної системи, забезпечуючи при цьому надійність та безпеку передачі даних. [18, 29]

2.3 Отримання та обробка голосового тексту

Отримання тексту після обробки голосу в браузерах відбувається за допомогою технології Speech Recognition API, яка є частиною стандарту Web Speech API.

Спочатку браузер записує голосовий сигнал користувача за допомогою мікрофону. Потім, за допомогою механізмів цифрової обробки сигналів, які вбудовані в браузер, з записаного звуку виокремлюється голосовий потік, який потім аналізується за допомогою розпізнавання мови.

Speech Recognition API підтримує розпізнавання голосу на різних мовах, таких як англійська, французька, іспанська тощо. Після обробки голосу, API повертає розпізнаний текст у вигляді рядка.

Для отримання тексту після обробки голосу в браузерах, можна використовувати JavaScript-код, який використовує Speech Recognition API. Наприклад, можна створити об'єкт SpeechRecognition, який буде відповідати за розпізнавання мови, і визначити обробники подій для обробки результатів розпізнавання (рисунок 2.2)

Отже, після запуску розпізнавання мови, Speech Recognition API оброблює голосовий сигнал, виділяє з нього голосовий потік і перетворює його на текст. Потім API повертає отриманий текст, який можна використовувати для подальшої обробки, наприклад, для пошуку в Інтернеті, для керування веб-додатками та інших завдань.

```javascript
// Створюємо об'єкт розпізнавання мови
const recognition = new SpeechRecognition();

// Встановлюємо мову розпізнавання
recognition.lang = 'en-US';

// Додаємо обробник події для отримання тексту після розпізнавання
recognition.onresult = (event) => {
  const text = event.results[0][0].transcript;
  console.log(text);
};

// Запускаємо розпізнавання мови
recognition.start();
```

Рисунок 2.2 - Приклад застосування Speech Recognition API

Застосування Speech Recognition API може бути корисним у випадках, коли користувачі не можуть вводити текст за допомогою клавіатури, наприклад, коли вони користуються мобільними пристроями або планшетами. Крім того, розпізнавання мови може бути використане для покращення доступності веб-додатків для людей з обмеженнями.

Проте варто зазначити, що розпізнавання мови є складним завданням, і точність розпізнавання може залежати від декількох факторів, таких як шум у приміщенні, якість мікрофону та мовлення користувача. Тому, для досягнення кращої точності розпізнавання, можуть застосовуватися додаткові техніки обробки сигналів та машинного навчання. [2, 25, 26]

2.4 Вибір технології розробки голосового асистенту

2.4.1 Мова розмітки веб сторінок : HTML 5

HTML 5 (Hypertext Markup Language, version 5) - це остання версія мови розмітки гіпертексту, яка відкриває нові можливості для веб-розробників та забезпечує покращену функціональність та ефективність створення веб-сторінок та веб-додатків. Вона є значним кроком уперед у порівнянні з попередніми версіями, пропонуючи вдосконалені можливості та нові функції для веб-розробників.

HTML 5 пропонує ряд нових технологій та покращень, які спрощують розробку веб-сторінок і додають їм більшу функціональність. Ці технології включають в себе підтримку відео та аудіо без використання додаткових плагінів, розширені можливості роботи з графікою, нові елементи форм та API для роботи з локальним сховищем та геолокацією.

Однією з ключових переваг HTML 5 є його здатність працювати на різних пристроях та платформах, що робить його ідеальним вибором для розробки веб-додатків, які мають бути доступні на різних пристроях із різними розмірами екрану та характеристиками. Це дозволяє створювати адаптивні веб-сайти, які оптимально відображаються на різних пристроях, включаючи комп'ютери, планшети та смартфони.

Загалом, HTML 5 відкриває широкі можливості для створення сучасних та інноваційних веб-додатків, які задовольняють потреби користувачів та відповідають сучасним вимогам веб-розробки.

HTML 5 містить багато нових елементів та атрибутів, які дозволяють розробникам створювати більші, більш складні та більш інтерактивні веб-сторінки.

Основні нововведення в HTML 5 включають:

1. Покращення роботи з мультимедіа. HTML 5 надає можливість вбудовувати відео та аудіо без додаткових плагінів, таких як Flash. Відео та аудіо

можуть бути відтворені на сторінці безпосередньо у браузері. Це полегшує відтворення мультимедійного вмісту на веб-сторінках та забезпечує кращу сумісність з різними пристроями та браузерами.

2. Нові елементи. HTML 5 містить нові елементи, такі як header, footer, nav, article та section, які дозволяють розробникам створювати більш структуровані та семантично коректні веб-сторінки. Це полегшує розуміння структури документа та поліпшує доступність та SEO оптимізацію.

3. Розробка веб-додатків. HTML 5 надає можливість розробки веб-додатків, які можуть працювати офлайн, тобто без доступу до Інтернету. Для цього використовується механізм Application Cache, який дозволяє зберігати ресурси веб-додатка локально на пристрої користувача.

4. Покращення форм. HTML 5 містить нові елементи та атрибути, які дозволяють розробникам створювати більші та більш складні форми. Крім того, в HTML 5 додано нові типи полів для введення даних, такі як email, url та date, що полегшує валідацію та обробку введених даних.

5. Підтримка графіки. HTML 5 містить нові елементи та атрибути, які дозволяють розробникам створювати графічні ефекти та анімацію без використання додаткових технологій, таких як Flash або JavaScript. Це дозволяє створювати більш привабливі та інтерактивні веб-сторінки без зайвого навантаження на пристрої користувача.

HTML 5 представляє значний прогрес у сфері веб-розробки, надаючи розробникам широкий спектр нових можливостей та інструментів для створення сучасних та інноваційних веб-додатків та веб-сторінок. Ця версія мови розмітки не лише полегшує вбудовування мультимедійного вмісту, роботу з формами та створення адаптивних та семантично коректних структур, але й дозволяє розробникам створювати веб-додатки, які можуть працювати офлайн, та застосовувати графічні ефекти без використання додаткових технологій. Загалом, HTML 5 відкриває нові можливості для творчості та інновацій у веб-

розробці, що допомагає відповідати зростаючим потребам користувачів та вимогам сучасного Інтернету.

2.4.2 Мова для описання стилю сторінок: CSS 3 / SCSS

Завдяки цьому з’являється стилізація нашого веб додатку.

CSS (Cascading Style Sheets) - це мова, яка використовується для опису вигляду та стилю веб-сторінок. CSS розділяє стиль веб-сторінки від її структури та змісту, що дозволяє розробникам змінювати стиль сторінки, не змінюючи її вмісту.

CSS дозволяє використовувати різні властивості, такі як розмір, колір, шрифт, положення, фон і багато іншого. Він також має поняття класів та ідентифікаторів, які дозволяють створювати відмінні стилі для окремих елементів веб-сторінки.

SCSS (Sass CSS) - це препроцесор CSS, який дозволяє розробникам писати CSS код з використанням змінних, функцій, міксинів та інших функцій, які не підтримуються в стандартному CSS.

SCSS базується на синтаксисі Sass, який використовує меншу кількість символів для опису CSS правил, що робить код більш зрозумілим та легко зчитуваним. SCSS також дозволяє використовувати вкладені стилі, що полегшує створення та організацію CSS коду.

Основні переваги використання SCSS у порівнянні зі стандартним CSS включають:

1. Використання змінних, що дозволяє швидко змінювати значення стилів.
2. Використання міксинів, що дозволяє повторно використовувати стилі з одного елементу на інший.
3. Використання вкладених стилів, що полегшує організацію та структурування CSS коду.
4. Підтримка операторів, функцій та умовних виразів, що дозволяє створювати більш динамічні та складні CSS правила.

2.4.3 Мова програмування: JavaScript ES6 / TypeScript

Завдяки цьому з’являється інтерактивність додатку.

JavaScript та TypeScript - це дві мови програмування, що активно використовуються для розробки веб-додатків та взаємодії з користувачем на веб-сторінках.

JavaScript, створений у 1995 році, використовується для створення динамічних веб-сторінок та додатків. Він дозволяє змінювати вміст та стиль веб-сторінок, додавати ефекти та взаємодіяти з сервером без перезавантаження сторінки.

TypeScript є розширенням JavaScript, що додає до нього підтримку статичного типізації та об'єктно-орієнтованого програмування. Використовується для розробки великих та складних проектів, де потрібна висока робоча стійкість та гнучкість. Він дозволяє розробникам збільшити безпеку та чіткість коду, а також спрощує його підтримку та розширення.

Основні переваги JavaScript включають:

1. Широке застосування у веб-розробці.
2. Легкість вивчення та використання.
3. Багато фреймворків та бібліотек для спрощення веб-розробки.
4. Динамічність та можливість взаємодії з користувачем.
5. Наявність великої кількості ресурсів та документації для допомоги в розробці.

Основні переваги TypeScript включають:

1. Статична типізація, що дозволяє виявляти помилки на етапі компіляції.
2. Підтримка об'єктно-орієнтованого програмування та наслідування класів.
3. Більш безпечний та прогнозовуваний код завдяки перевірці типів.
4. Можливість використання новітніх функцій JavaScript за допомогою компілятора TypeScript.

5. Наявність великої кількості бібліотек та фреймворків, що спрощують розробку.

Завдяки CSS (Cascading Style Sheets) розробники можуть ефективно відокремлювати вигляд веб-сторінок від їхньої структури та змісту, що сприяє простоті та гнучкості у редагуванні стилю сторінки. Використання SCSS (Sass CSS) додає ще більше можливостей, забезпечуючи зручний та потужний інструментарій для роботи з CSS кодом. З використанням змінних, міксинів, вкладених стилів та інших функцій SCSS стає важливим інструментом для створення складних та динамічних веб-додатків, що відповідають сучасним вимогам веб-дизайну та розробки. Відтак, об'єднуючи CSS та SCSS, розробники можуть ефективно стилізувати та налаштовувати вигляд своїх веб-додатків, роблячи їх привабливими та користувацьки зручними.

2.4.4 Фреймворк React

React - це важлива JavaScript бібліотека, яка забезпечує створення користувацьких інтерфейсів для веб-сайтів. Розроблена Facebook у 2011 році, React став вельми популярним інструментом для розробки веб-додатків.

Його основна концепція полягає в розбитті інтерфейсу на невеликі незалежні компоненти, які можна повторно використовувати. Кожен компонент має власний стан та поведінку, що робить код більш зрозумілим і легким у розвитку та підтримці.

Одним з головних переваг React є використання Virtual DOM, що забезпечує ефективну роботу з інтерфейсом. Він дозволяє реалізувати швидку відповідь користувачу та оптимізує роботу з DOM. Такий підхід зменшує витрати ресурсів та забезпечує плавну роботу додатків.

React здатний працювати разом з різноманітними іншими бібліотеками та фреймворками, що відкриває безмежні можливості для розробки динамічних веб-додатків. Він також має широку спільноту розробників, яка постійно розвивається та вдосконалюється.

Основні переваги React включають:

1. Можливість розбити інтерфейс на компоненти, що дозволяє повторно використовувати код та спрощує розробку.

2. Використання Virtual DOM, що забезпечує швидку та ефективну роботу з інтерфейсом.

3. Декларативний підхід до програмування, що спрощує розробку та забезпечує більш простий та зрозумілий код.

4. Широкий вибір бібліотек та фреймворків, що підтримують React та спрощують розробку.

5. Підтримка серверного рендерингу, що дозволяє забезпечити кращу оптимізацію та підвищити. [24]

React є потужним інструментом для створення користувацьких інтерфейсів веб-сайтів, що забезпечує простоту та ефективність розробки. Завдяки своїй основній концепції компонентного підходу та використанню Virtual DOM, React дозволяє розробникам створювати динамічні та ефективні веб-додатки з швидкою реакцією на дії користувачів. Його декларативний підхід до програмування та широкий вибір бібліотек і фреймворків, що підтримують React, роблять його одним із найпопулярніших інструментів для веб-розробки. Він надає можливість розробникам створювати високоякісні веб-додатки з плавною роботою та великою швидкодією, що відповідає вимогам сучасного веб-середовища.

2.4.5 Серверна частина Node.JS

Саме в цій частині прописується логіка серверної частини, яка оброблює голосовий текст користувача й виконує певні дії.

Node.js - це платформа, яка дозволяє виконувати JavaScript код на стороні сервера. Вона базується на движку V8, який використовується в браузері Google Chrome. Node.js дозволяє розробникам створювати серверні додатки та робити їх більш ефективними та швидкими.

Однією з ключових особливостей Node.js є його асинхронна модель програмування. Це означає, що Node.js може оброблювати багато запитів одночасно без блокування потоків виконання. Замість цього, Node.js використовує один потік та використовує non-blocking I/O операції, які забезпечують швидку відповідь та ефективне використання ресурсів. [20, 21]

Основні переваги Node.js:

1. Швидкість та ефективність - Node.js дозволяє швидко обробляти багато запитів одночасно, що забезпечує більш ефективну роботу додатків.
2. Асинхронність - Node.js дозволяє обробляти запити без блокування потоків, що забезпечує більш швидку відповідь та ефективне використання ресурсів.
3. JavaScript - Node.js дозволяє використовувати JavaScript на стороні сервера, що забезпечує зручність та зрозумілість розробки.
4. Розширення та пакети - Node.js має велику кількість розширень та пакетів, які дозволяють розробникам створювати більш складні та функціональні додатки.
5. Серверний рендеринг - Node.js дозволяє виконувати серверний рендеринг, що забезпечує більш оптимальну роботу додатків та підвищує їхню продуктивність[20, 21, 22, 23].

Node.js є потужною платформою для виконання JavaScript коду на стороні сервера, що дозволяє розробникам створювати ефективні та швидкодіючі серверні додатки. Його асинхронна модель програмування дозволяє обробляти

багато запитів одночасно без блокування потоків виконання, що забезпечує швидку відповідь та ефективне використання ресурсів сервера. З використанням JavaScript на стороні сервера та широким спектром розширень та пакетів, Node.js надає розробникам зручний та потужний інструментарій для створення складних та функціональних додатків. Його можливості включають не лише швидкість та ефективність обробки запитів, але й серверний рендеринг, що підвищує продуктивність та оптимізує роботу веб-додатків. [22, 23]

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ДОДАТКУ

3.1 Структура веб-додатку

Детальна схема застосунку наведена на рисунку 3.1.

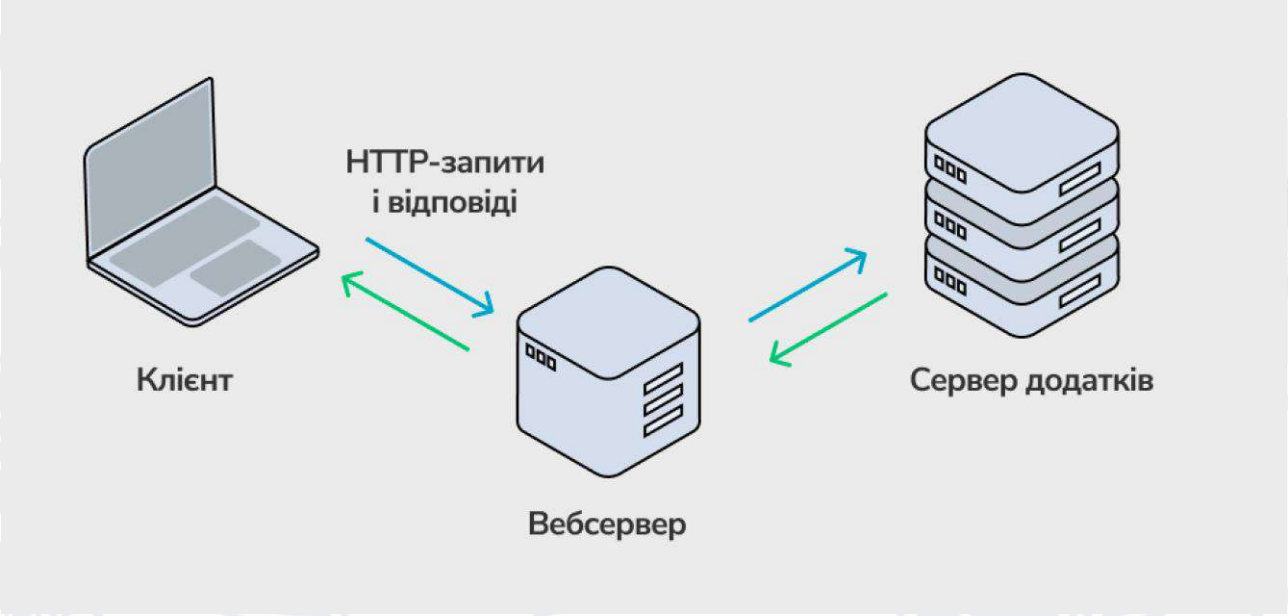

Рисунок 3.1 – Схема структури додатку

Клієнт ініціює запит до сервера, який може бути веб-сервером або API-сервером. На блок-схемі це показано як стрілка, що йде від клієнта до сервера. Після цього сервер перевіряє отриманий запит і визначає, чи може обробити його самостійно.

Якщо для обробки запиту необхідно звернутися до інших додатків або сервісів, сервер виконує внутрішні запити до цих систем. Це показано на блок-схемі стрілками, які йдуть від сервера до серверів додатків.

Після того, як сервер отримує відповіді від інших додатків або сервісів, він обробляє отримані дані, розраховує результат і формує відповідь клієнту. Це зображено на блок-схемі як стрілка, що йде від серверів додатків до сервера, а потім від сервера до клієнта.

Таким чином, блок-схема показує взаємодію між клієнтом, сервером та іншими серверами додатків у процесі обробки запитів та надсилання відповідей. [19].

3.1.1 Діаграма алгоритму роботи програми

Діаграма алгоритму роботи програми наведена на рисунку 3.2.

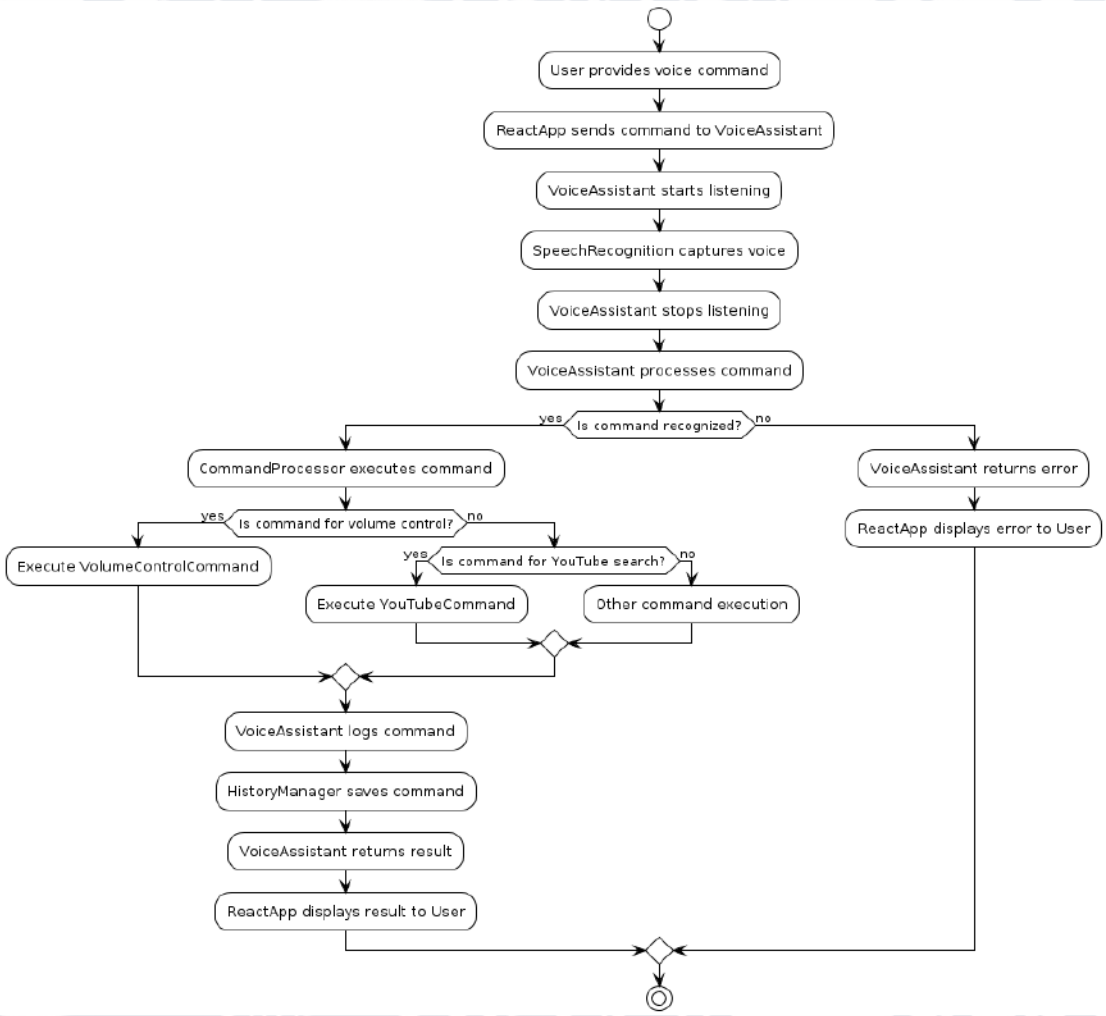

Рисунок 3.2 – Діаграма алгоритму роботи програми

Опис діаграми алгоритму роботи програми

1. Start: Початок алгоритму роботи програми.
2. User provides voice command: Користувач надає голосову команду через інтерфейс ReactApp.

3. ReactApp sends command to VoiceAssistant: ReactApp передає команду до VoiceAssistant.

4. VoiceAssistant starts listening: VoiceAssistant починає прослуховування голосової команди.

5. SpeechRecognition captures voice: SpeechRecognition захоплює голосову команду.

6. VoiceAssistant stops listening: VoiceAssistant зупиняє прослуховування.

7. VoiceAssistant processes command: VoiceAssistant обробляє команду.

8. Is command recognized?: Перевірка, чи була команда розпізнана.

- Yes:
- CommandProcessor executes command: CommandProcessor виконує команду.
- Is command for volume control?: Перевірка, чи є команда для управління гучністю.
- Yes:
- Execute VolumeControlCommand: Виконання команди управління гучністю.
- No:
- Is command for YouTube search?: Перевірка, чи є команда для пошуку на YouTube.
- Yes:
- Execute YouTubeCommand: Виконання команди пошуку на YouTube.
- No:
- Other command execution: Виконання іншої команди.
- VoiceAssistant logs command: VoiceAssistant зберігає команду в лог.
- HistoryManager saves command: HistoryManager зберігає команду.
- VoiceAssistant returns result: VoiceAssistant повертає результат.
- ReactApp displays result to User: ReactApp відображає результат користувачу.

- No:
- VoiceAssistant returns error: VoiceAssistant повертає помилку.
- ReactApp displays error to User: ReactApp відображає помилку користувачу.

9. End: Кінець алгоритму роботи програми.

Ця діаграма алгоритму роботи програми ілюструє основні етапи обробки голосових команд у системі голосового асистента, від початку прослуховування до виконання команд і збереження їх в історії або обробки помилок.

3.1.2 Опис діаграми класів додатку

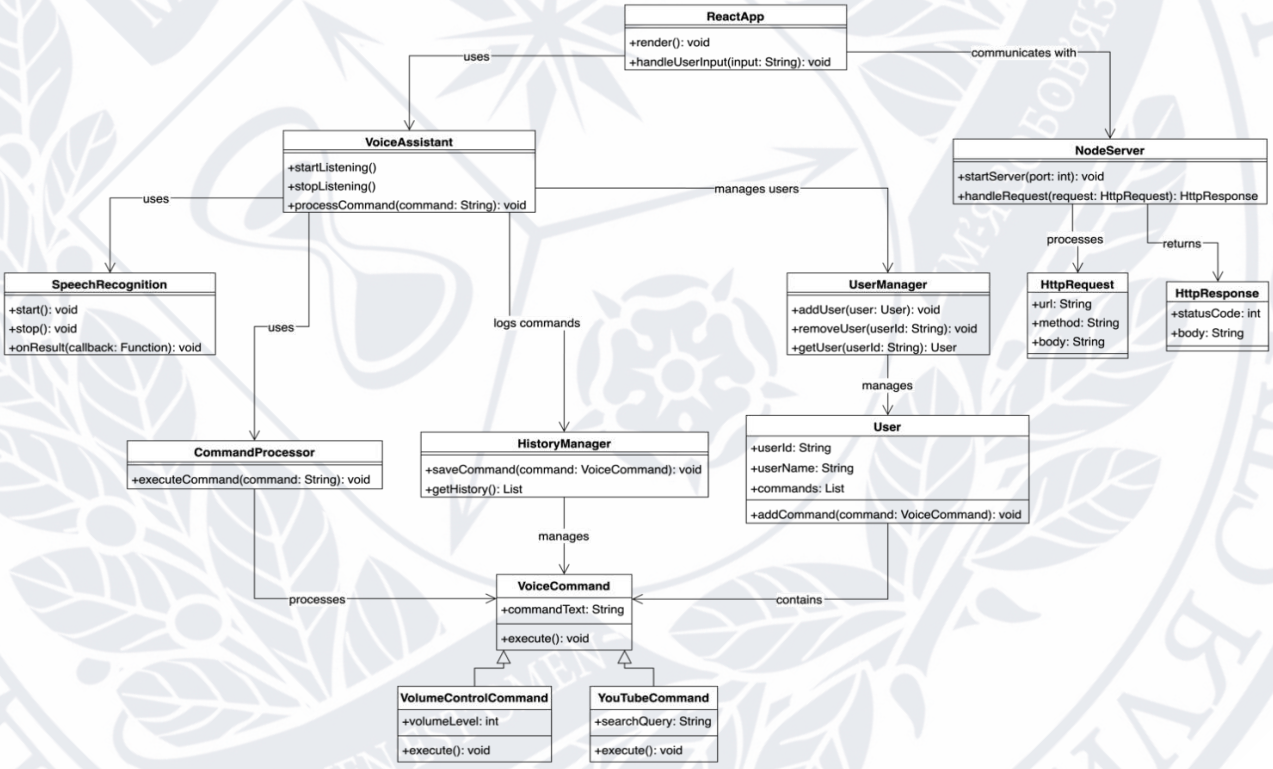

Рисунок 3.3 – Діаграма класів

Діаграма класів зображує основні компоненти та їх взаємодію в системі голосового асистента, розробленого за допомогою Node.js та React. Класи діаграми включають як клієнтську, так і серверну частини додатку, а також механізми обробки голосових команд та управління користувачами. Нижче

наводиться опис кожного з класів, представлених на діаграмі, та їх основні функції.

Класи та їх взаємодія:

1. VoiceAssistant

- Методи:
- +startListening(): запускає прослуховування голосових команд.
- +stopListening(): зупиняє прослуховування голосових команд.
- +processCommand(command: String): void: обробляє отриману голосову команду.
- Взаємодія:
- Використовує SpeechRecognition для розпізнавання голосу.
- Використовує CommandProcessor для виконання команд.
- Логує команди за допомогою HistoryManager.
- Управляє користувачами через UserManager.

2. SpeechRecognition

- Методи:
- +start(): void: запускає процес розпізнавання мови.
- +stop(): void: зупиняє процес розпізнавання мови.
- +onResult(callback: Function): void: встановлює функцію зворотного виклику для обробки результатів розпізнавання.

3. NodeServer

- Методи:
- +startServer(port: int): void: запускає сервер на вказаному порту.
- +handleRequest(request: HttpRequest): HttpResponse: обробляє HTTP запити та повертає відповідь.

4. ReactApp

- Методи:
- +render(): void: рендерить інтерфейс користувача.
- +handleUserInput(input: String): void: обробляє введення користувача.

5. CommandProcessor

- Методи:
- +executeCommand(command: String): void: виконує отриману команду.

6. VoiceCommand

- Властивості:
- +commandText: String: текст команди.
- Методи:
- +execute(): void: виконує команду.

7. VolumeControlCommand (наслідує VoiceCommand)

- Властивості:
- +volumeLevel: int: рівень гучності.
- Методи:
- +execute(): void: виконує команду зміни гучності.

8. YouTubeCommand (наслідує VoiceCommand)

- Властивості:
- +searchQuery: String: пошуковий запит.
- Методи:
- +execute(): void: виконує команду пошуку на YouTube.

9. HistoryManager

- Методи:
- +saveCommand(command: VoiceCommand): void: зберігає команду в історію.
- +getHistory(): List<VoiceCommand>: повертає історію команд.

10. UserManager

- Методи:
- +addUser(user: User): void: додає нового користувача.
- +removeUser(userId: String): void: видаляє користувача за ідентифікатором.

- +getUser(userId: String): User: повертає інформацію про користувача.

11. User

- Властивості:
- +userId: String: ідентифікатор користувача.
- +userName: String: ім'я користувача.
- +commands: List<VoiceCommand>: список команд користувача.
- Методи:
- +addCommand(command: VoiceCommand): void: додає команду до списку команд користувача.

12. HttpRequest

- Властивості:
- +url: String: URL запиту.
- +method: String: метод запиту (GET, POST тощо).
- +body: String: тіло запиту.

13. HttpResponse

- Властивості:
- +statusCode: int: статусний код відповіді.
- +body: String: тіло відповіді.

Взаємозв'язки між класами:

- VoiceAssistant використовує SpeechRecognition для розпізнавання голосу, CommandProcessor для виконання команд, HistoryManager для збереження команд в історію та UserManager для управління користувачами.
- ReactApp взаємодіє з VoiceAssistant для обробки команд користувача та з NodeServer для обробки HTTP запитів.
- CommandProcessor використовує VoiceCommand та його похідні класи (VolumeControlCommand, YouTubeCommand) для виконання відповідних команд.
- HistoryManager зберігає та повертає команди VoiceCommand.

- UserManager керує об'єктами User, які містять список команд VoiceCommand.
- NodeServer обробляє HttpRequest та повертає HttpResponse.

Ця діаграма класів відображає структурну архітектуру голосового асистента й ілюструє, як різні компоненти системи взаємодіють між собою для забезпечення функціональності додатку.

3.1.3 Опис компонентів програми

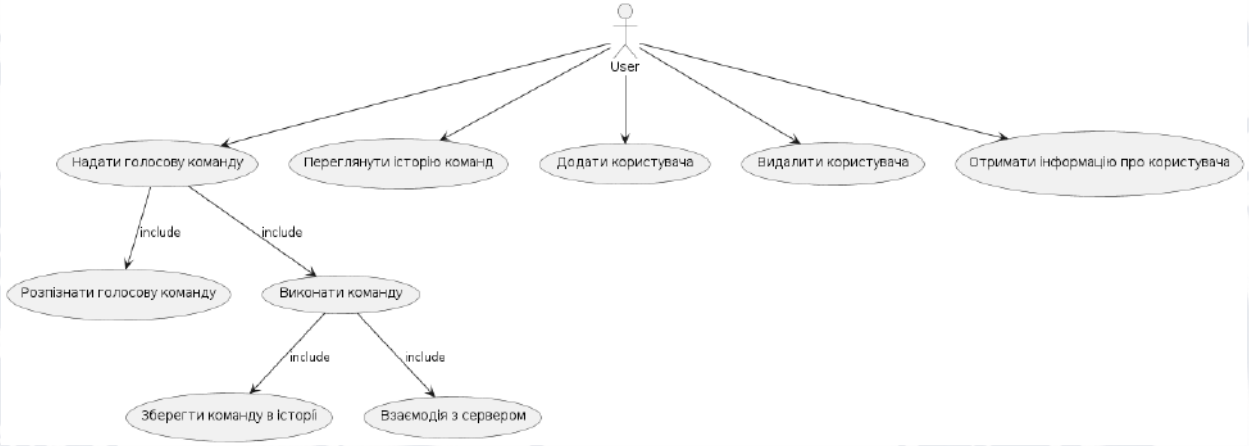

Рисунок 3.4 – Діаграма компонентів

Опис компонентів:

1. User Interface
   - Інтерфейс користувача для взаємодії з голосовим асистентом.
2. ReactApp
   - Відповідає за рендеринг інтерфейсу користувача та обробку введення користувача.
   - Передає голосові команди до VoiceAssistant.
   - Комунікує з NodeServer для обробки запитів.
3. VoiceAssistant
   - Обробляє голосові команди, передані з ReactApp.
   - Використовує SpeechRecognition для розпізнавання голосових команд.
   - Використовує CommandProcessor для виконання команд.

- Логує команди за допомогою HistoryManager.
- Управляє користувачами через UserManager.

4. SpeechRecognition
- Відповідає за розпізнавання голосових команд.

5. CommandProcessor
- Обробляє команди VoiceCommand та його похідні класи (VolumeControlCommand, YouTubeCommand).

6. HistoryManager
- Зберігає та повертає історію команд.

7. UserManager
- Управляє користувачами та їх командами.

8. NodeServer
- Обробляє HTTP запити від ReactApp та повертає відповіді.

9. VolumeControlCommand
- Виконує команди зміни гучності.

10. YouTubeCommand
- Виконує команди пошуку на YouTube.

Взаємозв'язки між компонентами:

- User Interface взаємодіє з ReactApp, що відповідає за рендеринг інтерфейсу та обробку введення користувача.
- ReactApp передає голосові команди до VoiceAssistant та комунікує з NodeServer для обробки запитів.
- VoiceAssistant обробляє голосові команди, використовуючи SpeechRecognition для розпізнавання голосу, CommandProcessor для виконання команд, HistoryManager для збереження команд в історію, та UserManager для управління користувачами.
- CommandProcessor обробляє команди VoiceCommand та його похідні класи (VolumeControlCommand, YouTubeCommand).
- NodeServer обробляє HTTP запити від ReactApp та повертає відповіді.

Ця діаграма компонентів відображає основні компоненти системи голосового асистента та їх взаємодію, показуючи, як клієнтські і серверні частини додатку працюють разом для забезпечення повної функціональності системи.

3.2 Макет дизайну веб-додатку

Дизайн застосунку доволі мінімалістичний. І він дуже добре підходить для додатку, який присвячений голосовому асистенту. Він дозволить користувачеві зосередитися на важливих функціях та завданнях, які можна виконувати за допомогою асистента, не відволікаючись на зайві деталі або елементи інтерфейсу.

Збоку наявне бічна панель історії запитів. Користувач може швидко переглядати свої попередні запити або виконані дії, що спростить йому роботу з асистентом. Такий підхід до дизайну дозволяє зробити додаток більш зручним та доступним для використання, а також додає йому сучасний та стильний вигляд (рис 3.5).

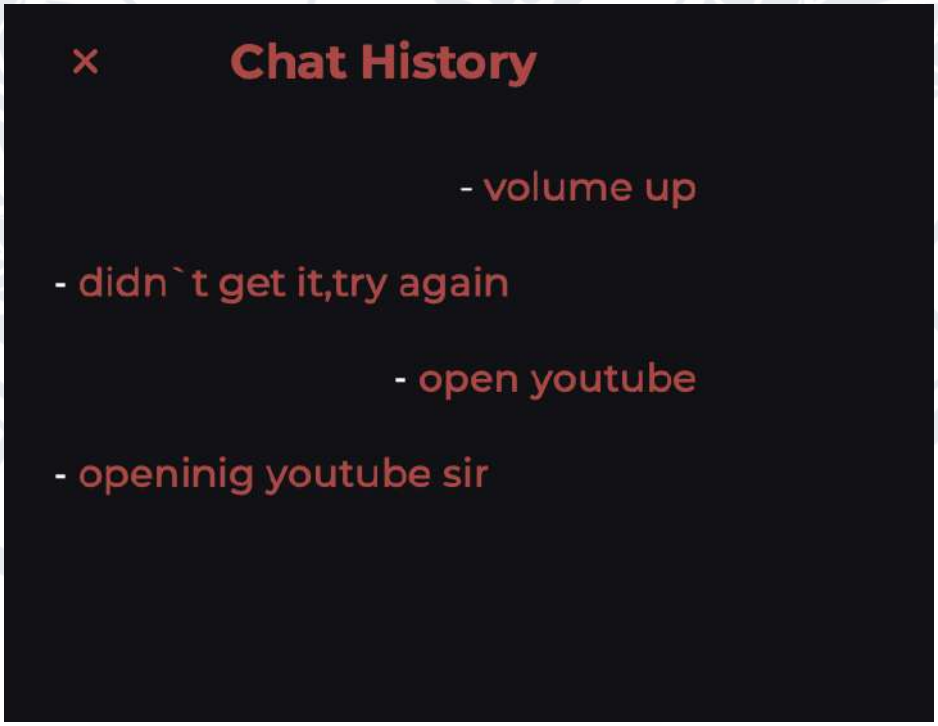

Рисунок 3.5 – Бокова панель

В центрі присутня кнопка для виклику асистента. Центрально розміщена кнопка виклику асистента робить її легкою для знаходження і натискання, навіть для новачків. Крім того, це створить точку фокусу в інтерфейсі, яка приверне увагу користувачів і підкреслить важливість асистента в додатку. Такий центральний елемент може стати ключовим з точки зору зручності використання та ергономіки інтерфейсу, що додасть йому естетику та привабливість (рис. 3.6).

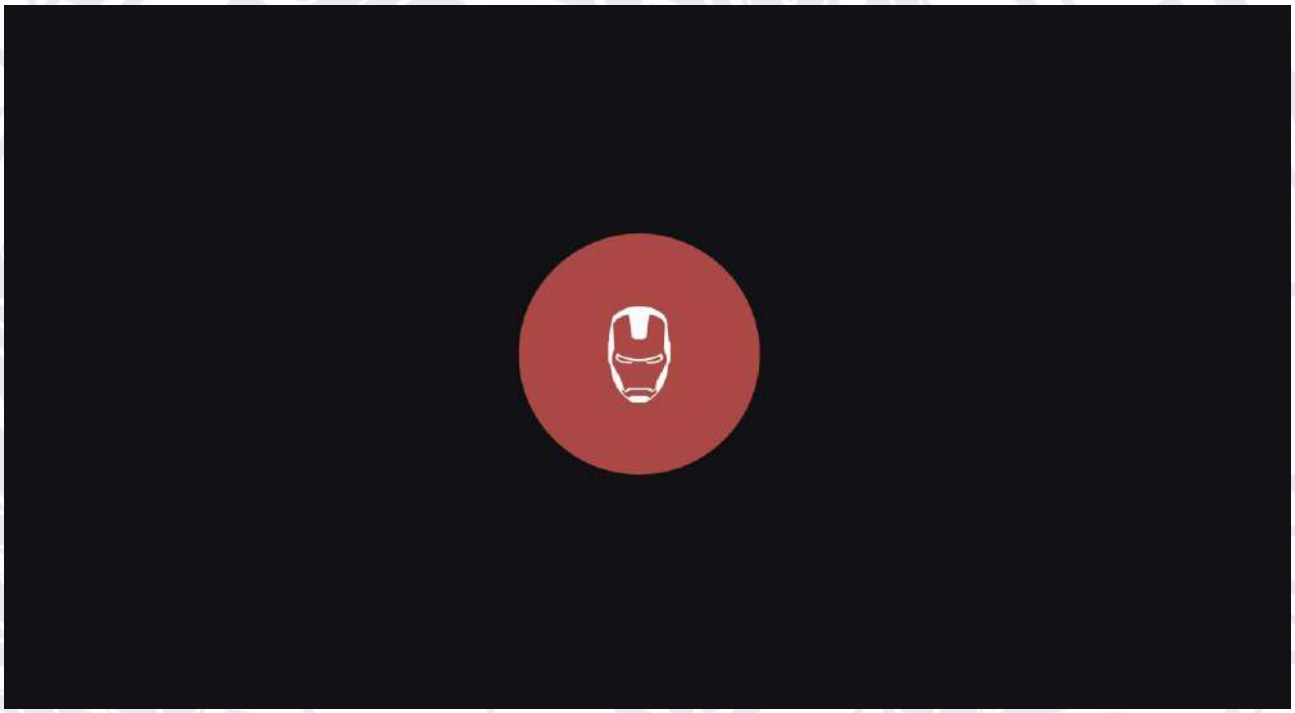

Рисунок 3.6 – Кнопка для виклику асистента

Додаток включає функціонал для голосових команд, таких як збільшення гучності або відкриття YouTube. Якщо команда не розпізнана, асистент інформує користувача про необхідність спробувати ще раз. Це забезпечує зворотний зв'язок, що допомагає користувачам навчатися взаємодіяти з асистентом більш ефективно.

Такий підхід до дизайну та функціональності дозволяє створити інтуїтивно зрозумілий інтерфейс, де основні функції легко доступні, а користувач отримує швидкий зворотний зв'язок щодо своїх дій. Це робить використання голосового асистента приємним і продуктивним досвідом.

3.3 Серверна частина додатку

Серверна частина написана на Node.js з використанням таких бібліотек, як Experss та Puppeteer.

Node.js - це потужна технологія, яка дозволяє писати серверну частину додатків на JavaScript. Використання Express, одного з найпопулярніших фреймворків для Node.js, спрощує створення серверних додатків швидко та ефективно. Puppeteer - це інструмент для керування веб-браузером через програмний інтерфейс, що дозволяє автоматизувати взаємодію з веб-сторінками, виконувати дії, які зазвичай робляться людиною, такі як натискання кнопок або заповнення форм.

Давайте розглянемо код більш детально.

Ми спочатку створюємо екземпляр додатку Express за допомогою express(). Потім ми створюємо HTTP сервер, який слухає на порту 9999 і приймає наш додаток Express як обробник запитів. Це дозволяє нашому серверу слухати та відповідати на HTTP запити.

Далі ми створюємо інстанцію Socket.IO сервера, який також використовує наш HTTP сервер для з'єднання. Ми налаштовуємо CORS параметри для дозволу запитів з деяких джерел.

Middleware cors дозволяє нашому серверу приймати CORS запити, а express.json() дозволяє парсити JSON дані, що надходять у запиті.

Нарешті, ми встановлюємо обробник маршруту для кореневого шляху. При отриманні GET запиту до кореневого шляху, сервер відправляє рядок "hello" у форматі JSON (рис 3.7).

```
const app = express();
const server = http.Server(app).listen(9999,()=>console.log('server has been started'));
const io = new Server(server,{
  cors:{
    origin: "http://localhost:3000",
    allowedHeaders: ["my-custom-header"],
    credentials: true
  }
});
const corsOptions ={
   origin:'*',
   credentials:true,
   optionSuccessStatus:200,
}

app.use(cors(corsOptions));
app.use(express.json());

app.get('/',(req,res)=>{
    res.send(JSON.stringify('hello'))
})
```

Рисунок 3.7 – Створення серверу

Далі ми маємо асинхронну функцію, яка встановлює з'єднання з вже запущеним браузером Puppeteer за допомогою WebSocket і створює нову сторінку. Вона викликається одразу після запуску програми.

Після цього ми встановлюємо обробники подій для з'єднання через Socket.IO. При отриманні певних подій від клієнта, ми виконуємо певні дії на стороні сервера.

Наприклад, коли клієнт надсилає запит на збільшення гучності, ми встановлюємо нове значення гучності за допомогою бібліотеки loudness. Якщо клієнт надішле запит на вимкнення звуку, ми також викликаємо відповідний метод.

Крім того, ми можемо відкривати YouTube або виконувати пошук на YouTube за допомогою Puppeteer, переходячи на відповідні сторінки згідно з отриманими даними від клієнта (рис 3.8).

```
let browser = null;
let page = null;

(async()=>{
  browser = await puppeteer.connect({browserWSEndpoint:'ws://127.0.0.1:9222/devtools/browser/d40b7fa8-d083-
477b-b895-019efed13f70',defaultViewport:null}).catch((e)=>e);
  console.log(browser);
  page = await browser.newPage();
})();

io.on('connection',async (socket)=>{

  socket.on("make-volume-more",(data)=>{
    const { volume } = data;
    loudness.setVolume(volume);
  })
  socket.on("make-the-volume-mute",()=>{
    loudness.setMuted();
  })
  socket.on("open-youtube",async ()=>{
    await page.goto('https://www.youtube.com/', { waitUntil: 'networkidle2' });
  })
  socket.on("search-in-youtube",async(data)=>{
    const {searchText} = data;
    await page.goto( `https://www.youtube.com/results?search_query=${searchText
    .split(" ")
    .join("+")}`)

  })
})
```

Рисунок 3.8 – Робота сокету

Коли клієнт надсилає запит на збільшення гучності, ми встановлюємо нове значення гучності за допомогою бібліотеки loudness. Якщо клієнт надішле запит на вимкнення звуку, ми також викликаємо відповідний метод.

Далі описується асинхронна функція, яка встановлює з'єднання з вже запущеним браузером Puppeteer за допомогою WebSocket і створює нову сторінку. Ця функція викликається одразу після запуску програми. Після успішного підключення створюється нова сторінка у браузері.

Встановлюються обробники подій для з'єднання через Socket.IO. При отриманні певних подій від клієнта на стороні сервера виконуються відповідні дії. Наприклад, коли клієнт надсилає запит на збільшення гучності,

встановлюється нове значення гучності за допомогою бібліотеки loudness. У випадку, якщо клієнт надсилає запит на вимкнення звуку, викликається відповідний метод. Крім того, можливо відкривати YouTube або виконувати пошук на YouTube за допомогою Puppeteer, переходячи на відповідні сторінки згідно з отриманими даними від клієнта.

Опис JSON-діаграми потоку даних:

1. User: Користувач, який взаємодіє з інтерфейсом ReactApp.
2. ReactApp: Клієнтська частина, яка обробляє введення користувача і взаємодіє з серверними компонентами через JSON-повідомлення.
3. VoiceAssistant: Компонент, який обробляє голосові команди, передає їх для виконання і повертає результати у вигляді JSON.
4. CommandProcessor: Компонент, який обробляє команди і взаємодіє з сервером через JSON.
5. NodeServer: Сервер, який обробляє команди, отримані від CommandProcessor, і повертає результати у вигляді JSON.
6. HistoryManager: Компонент, який зберігає та повертає історію команд у вигляді JSON.
7. UserManager: Компонент, який обробляє дані користувача і повертає результати у вигляді JSON.

Взаємозв'язки між компонентами:

1. Надання голосової команди:

- Користувач надає голосову команду через інтерфейс ReactApp.
- ReactApp надсилає JSON-повідомлення з голосовою командою до VoiceAssistant.
- VoiceAssistant обробляє команду і надсилає JSON до CommandProcessor.
- CommandProcessor надсилає JSON з командою до NodeServer для обробки.

- NodeServer повертає результат у вигляді JSON до CommandProcessor.
- CommandProcessor повертає результат у вигляді JSON до VoiceAssistant.
- VoiceAssistant повертає результат у вигляді JSON до ReactApp.
- ReactApp відображає результат користувачу у вигляді JSON.

2. Перегляд історії команд:

- Користувач запитує перегляд історії команд через ReactApp.
- ReactApp надсилає запит у вигляді JSON до VoiceAssistant.
- VoiceAssistant надсилає запит у вигляді JSON до HistoryManager.
- HistoryManager повертає історію у вигляді JSON до VoiceAssistant.
- VoiceAssistant повертає історію у вигляді JSON до ReactApp.
- ReactApp відображає історію команд користувачу у вигляді JSON.

3. Управління користувачами:

- Користувач управляє своїм обліковим записом через ReactApp.
- ReactApp надсилає запит у вигляді JSON до VoiceAssistant.
- VoiceAssistant надсилає запит у вигляді JSON до UserManager.
- UserManager обробляє дані користувача і повертає результат у вигляді JSON до VoiceAssistant.
- VoiceAssistant повертає результат у вигляді JSON до ReactApp.
- ReactApp відображає результат користувачу у вигляді JSON.

Ця JSON-діаграма потоку даних ілюструє основні етапи обробки голосових команд, взаємодію між клієнтськими і серверними компонентами через JSON-повідомлення, а також управління користувачами і збереження історії команд у системі голосового асистента.

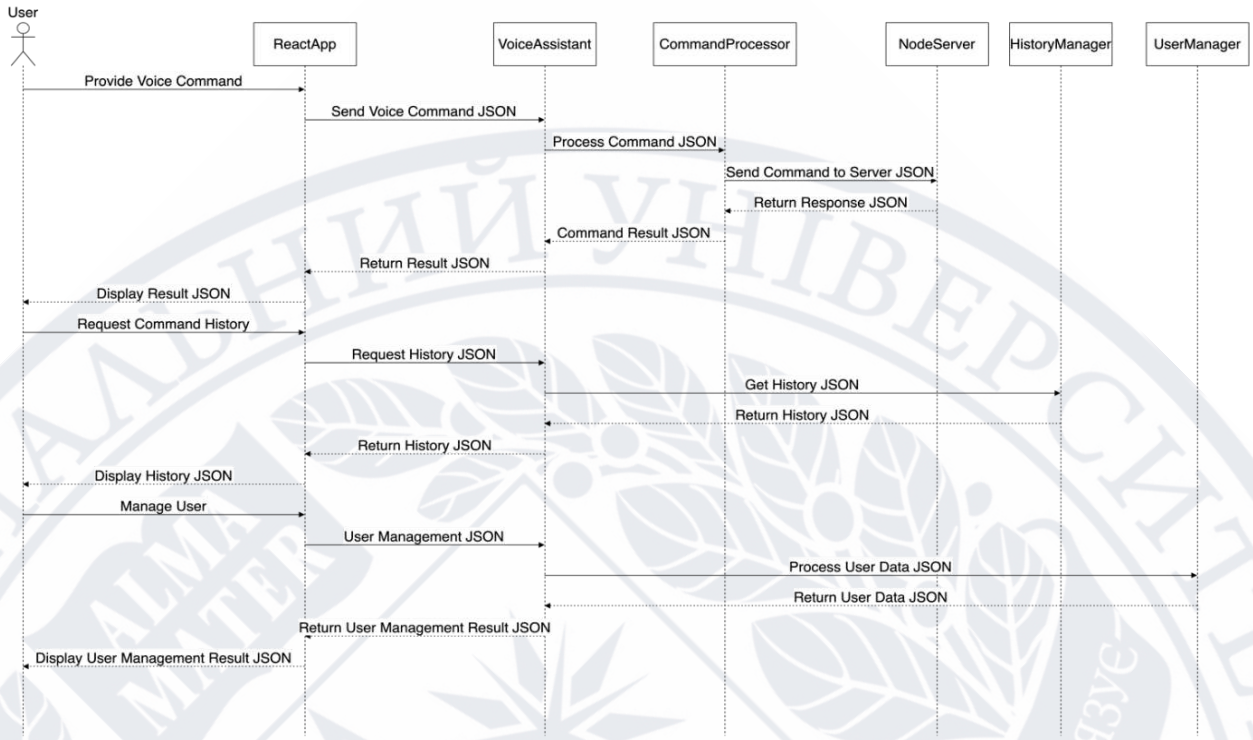

Рисунок 3.9 – JSON діаграма потоку даних

3.4 Клієнтська частина та приклад роботи

Основний принцип роботи додатку – це обробка голосового вводу користувача й в разі відповідності тексту до того випадку, який ми описуємо в коді виконується певна дія. Наприклад користувач може попросити голосового асистента виконати наступні дії:

- Зробити рівень гучності на пристрої користувача гучніше або навпаки тихіше або вимкнути його

-Зробити запит у гугл (відкрити сторінки із певним запитом)

-Зробити запит у YouTube (відкрити сторінку із певним запитом)

-Зробити запит до Штучного Інтелекту ChatGPT та надати інформацію щодо його відповіді.

Давайте тепер розглянемо код та структуру клієнтської сторони. Код написаний на React Js.

Почнемо зі структури папок. В нас є головна папка “src”. Вміст якої включає в себе папки з компонентами, шрифтами та допоміжними файлами (рис 3.10).

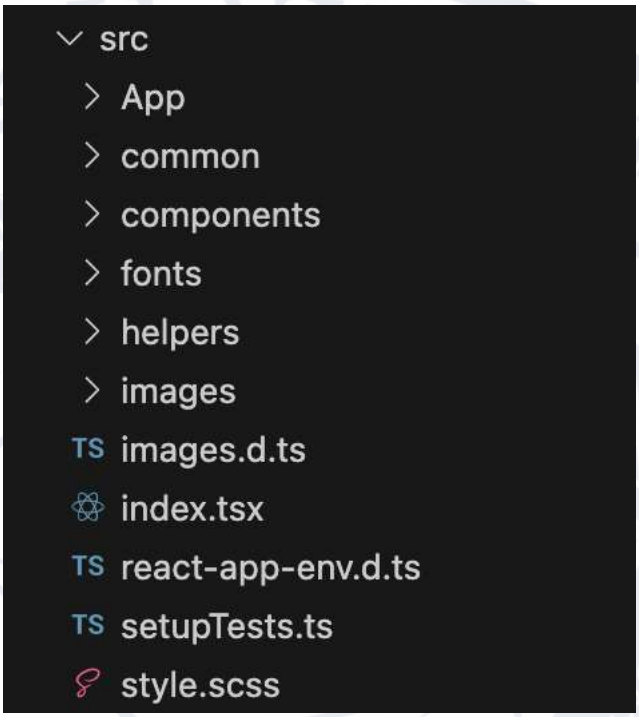

Рисунок 3.10 – Структура проекту

Далі перейдемо до коду. В папці “App” є головний файл для відмалювання додатку. В ньому ви викликаємо копноненту в якій знаходиться кнопка для виклику та вся логіка стосовно неї (рис 3.11).

```
import MainCircle from "@src/components/MainCircle";
import React from "react";

const App: React.FC = () => {
  return (
    <div>
      <MainCircle />
    </div>
  );
};

export default App;
```

Рисунок 3.11 – Головна компонента

Далі розберемо компоненту MainCircle.

```
const Recognition = window.SpeechRecognition || window.webkitSpeechRecognition;
const myRecord = new Recognition();
myRecord.lang = "en-US";

myRecord.continuous = true;

const MainCircle: React.FC = () => {
  const [isRecoStart, setIsRecoStart] = useState(false);
  const [chatMessages, setChatMessages] = useState<IChatMessage[]>([]);

  const startRecordHandler = () => {
    setIsRecoStart(true);
    myRecord.start();
  };

  const endRecordHandler = () => {
    setIsRecoStart(false);
    myRecord.stop();
  };

  myRecord.onresult = (e) => {
    getResultOfRecord({
      e,
      setChatMessages,
    });
  };
```

Рисунок 3.12 – Перша частина коду компоненти MainCircle

У фрагменті коду (рис 3.12) ми використовуємо бібліотеку для розпізнавання мови (SpeechRecognition) на клієнтській стороні (веб-браузер). Ми перевіряємо, яку саме реалізацію SpeechRecognition підтримує браузер, оскільки вона може відрізнятися залежно від браузера (window.SpeechRecognition або window.webkitSpeechRecognition).

Створюємо новий екземпляр Recognition та встановлюємо його властивості. У нашому випадку, ми встановлюємо мову розпізнавання на "en-US" та включаємо режим continuous, щоб можна було записувати мову безперервно.

У компоненті MainCircle, який є основним компонентом інтерфейсу, ми використовуємо хуки стану (useState) для відстеження стану запису мови (isRecoStart) та масиву повідомлень чату (chatMessages).

Метод startRecordHandler встановлює isRecoStart на true і починає запис мови за допомогою методу start() об'єкта myRecord.

Метод endRecordHandler встановлює isRecoStart на false і зупиняє запис за допомогою методу stop().

Після завершення запису ми отримуємо результати розпізнавання у вигляді подій (myRecord.onresult) і обробляємо їх, викликаючи функцію getResultOfRecord, яка оновлює стан chatMessages - масив повідомлень чату.

У наступному фрагменті (рис. 3.13) коду ми визначаємо функцію changeRecordStart, яка викликає startRecordHandler, якщо запис не ведеться, або endRecordHandler, якщо запис уже ведеться.

За допомогою useEffect ми встановлюємо ефект, що відбувається при монтуванні компоненту. Ми зчитуємо дані з локального сховища (localStorage) за ключем "chatMessages" і встановлюємо їх у стан chatMessages. Це дозволяє зберігати дані між сесіями користувача та відновлювати їх при перезавантаженні сторінки.

Також у цьому ефекті ми викликаємо метод getVoices об'єкта speechSynthesis, щоб отримати доступні голоси для синтезу мови.

Інший useEffect викликається при зміні стану chatMessages і оновлює дані у локальному сховищі, щоб зберегти останні зміни.

У поверненому JSX ми рендеримо компоненти Circle і ChatComponent, передаючи їм відповідні дані і функції. Circle відповідає за інтерфейс запису мови, а ChatComponent - за відображення чату.

```
const changeRecordStart = () => {
    if (!isRecoStart) {
      startRecordHandler();
    } else {
      endRecordHandler();
    }
  };

  useEffect(() => {
    const chatMessagesFromLS = localStorage.getItem("chatMessages");
    if (chatMessagesFromLS) {
      setChatMessages(JSON.parse(chatMessagesFromLS));
    }
    setTimeout(() => {
      speechSynthesis.getVoices();
    }, 50);
  }, []);

  useEffect(() => {
    localStorage.setItem("chatMessages", JSON.stringify(chatMessages));
  }, [chatMessages]);

  return (
    <div className="main-bg">
      <Circle changeRecordStart={changeRecordStart} />
      <ChatComponent
        chatMessages={chatMessages}
        setChatMessages={setChatMessages}
      />
    </div>
  );
};
```

Рисунок 3.13 – Друга частина коду компоненти MainCircle

Компонента яку ми тільки що розглянути рендерить 2 дочірні компоненти Circle та ChatComponent. Давайде розгялнемо ще їх більш детально.

Почнемо з Circle. У фрагменті коду (рис. 3.14) ми оголошуємо інтерфейс IProps, який містить один властивість changeRecordStart, що є функцією без параметрів і без поверненого значення.

Потім ми використовуємо цей інтерфейс для визначення типів властивостей компонента Circle. Компонент Circle приймає один аргумент props, який має бути об'єктом з властивістю changeRecordStart типу () => void.

У JSX коді ми рендеримо div елемент, який представляє собою круглий інтерфейс запису мови. При кліку на цей елемент викликається функція changeRecordStart, передана з батьківського компонента, яка відповідає за початок або завершення запису мови.

```
interface IProps {
  changeRecordStart: () => void;
}

const Circle: React.FC<IProps> = ({ changeRecordStart }) => {
  return (
    <div className="main-bg-circle-wrapper">
      <div className="main-bg-circle" onClick={changeRecordStart}>
        <div className="main-bg-circle-shadow" />
        <LogoSvg />
      </div>
    </div>
  );
};
```

Рисунок 3.14 – Компонент Circle

Далі перейдемо до компоненти ChatComponent. У фрагменті коду (рис 3.15) ми оголошуємо інтерфейс IProps, який визначає типи властивостей, які приймає компонент ChatComponent. Цей інтерфейс містить дві властивості: chatMessages, яка є масивом об'єктів типу IChatMessage, і setChatMessages, яка є функцією React.Dispatch для оновлення стану chatMessages.

У компоненті ChatComponent ми використовуємо ці властивості для відображення списку повідомлень чату. При кліку на кнопку "clearChatHandler" здійснюється очищення чату за допомогою функції setChatMessages, яка встановлює новий порожній масив для chatMessages.

Далі ми відображаємо заголовок "Chat History" і список повідомлень чату, перебираючи елементи масиву chatMessages за допомогою методу map(). Кожен елемент чату представлений відповідним div елементом з класом "chat-wrapper__chat-item", де текст повідомлення відображається в тезі <p>. Клас "chat-wrapper__chat-item--right" використовується для вирівнювання

повідомлень праворуч, якщо вони належать користувачу (MessageOwnerEnum.ME).

```
interface IProps {
  chatMessages: IChatMessage[];
  setChatMessages: React.Dispatch<React.SetStateAction<IChatMessage[]>>;
}
const ChatComponent: React.FC<IProps> = ({ chatMessages, setChatMessages }) => {
  const clearChatHandler = () => {
    setChatMessages([]);
  };

  return (
    <div className="chat-wrapper">
      <div className="chat-wrapper__close-chat" onClick={clearChatHandler}>
        <CloseSVG />
      </div>
      <div className="chat-wrapper__title">Chat History</div>
      <div className="chat-wrapper__chat">
        {chatMessages.map((item) => (
          <div
            key={item.id}
            className={`chat-wrapper__chat-item ${
              item.owner === MessageOwnerEnum.ME
                ? "chat-wrapper__chat-item--right"
                : ""
            }`}
          >
            <p>{item.message}</p>
          </div>
        ))}
      </div>
    </div>
  );
};
```

Рисунок 3.15 – Компонента ChatComponent

І на прикінці розглянемо як виглядає додаток коли його запустити (рис. 3.16)

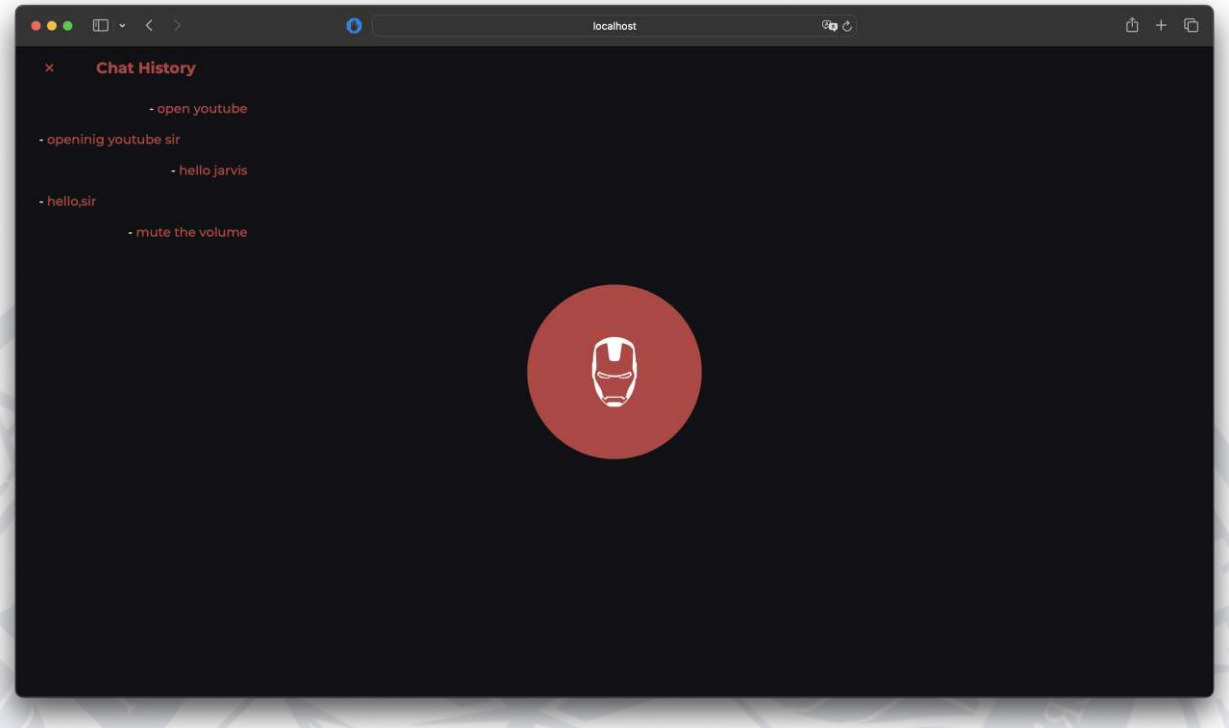

Рисунок 3.16 – Робота додатку

3.5 Тестування додатку

Тестування є важливим етапом розробки програмного забезпечення, що дозволяє виявити помилки, перевірити відповідність вимогам і забезпечити надійну та стабільну роботу додатку. В даному розділі розглядаються методи, які були використані для тестування голосового асистента, розробленого з використанням Node.js та React.

3.5.1 Модульне тестування

Модульне тестування є одним із найважливіших етапів тестування програмного забезпечення, що дозволяє перевірити функціональність окремих модулів та компонентів додатку в ізоляції. Метою модульного тестування є виявлення помилок на ранніх етапах розробки та забезпечення коректної роботи кожного окремого блоку коду. Для реалізації модульного тестування у нашому проекті було використано такі інструменти, як Mocha, Chai та Jest.

Інструменти для модульного тестування:

1. Mocha – це популярний фреймворк для тестування JavaScript, який дозволяє виконувати асинхронні тести та забезпечує гнучкі можливості налаштування.

2. Chai – це бібліотека для асерцій, що використовується разом із Mocha. Вона надає зручний синтаксис для перевірки очікуваних результатів тестів.

3. Jest – це всеосяжний фреймворк для тестування JavaScript, розроблений Facebook. Jest забезпечує простоту налаштування, автоматичне створення знімків стану (snapshot testing) та інтеграцію з різними інструментами для забезпечення якості коду.

Розглянемо приклад модульного тестування для функції обробки голосових команд у додатку. Основна функція, яку ми будемо тестувати, називається processCommand. Вона приймає команду у вигляді тексту і повертає відповідний результат.

Функція processCommand обробляє різні голосові команди, такі як збільшення гучності, вимкнення звуку, відкриття YouTube тощо. Ось її реалізація:

```
function processCommand(command) {
  switch (command) {
    case 'increase volume':
      return 'Volume increased';
    case 'mute':
      return 'Volume muted';
    default:
      return 'Command not recognized';
  }
}
```

Для перевірки правильності роботи функції processCommand було написано кілька тестів з використанням Mocha та Chai. Ось приклад модульного тесту:

```
const expect = require('chai').expect;
const { processCommand } = require('../src/voiceAssistant');
describe('Voice Assistant Command Processing', () => {
 it('should return correct response for volume up command', () => {
  const command = 'increase volume';
  const response = processCommand(command);
  expect(response).to.equal('Volume increased');
 });
 it('should return correct response for mute command', () => {
  const command = 'mute';
  const response = processCommand(command);
  expect(response).to.equal('Volume muted');
 });
 it('should return error for unrecognized command', () => {
  const command = 'play music';
  const response = processCommand(command);
  expect(response).to.equal('Command not recognized');
 });
});
```

Кожен тест виконує наступні дії:

1. Викликає функцію processCommand з певною командою.
2. Перевіряє, чи повертає функція очікуваний результат за допомогою методу expect з бібліотеки Chai.

У першому тесті перевіряється команда збільшення гучності, і очікується, що функція поверне рядок "Volume increased". У другому тесті перевіряється команда вимкнення звуку, і очікується, що функція поверне рядок "Volume

muted". У третьому тесті перевіряється невідома команда, і очікується, що функція поверне рядок "Command not recognized".

Виконання цих тестів забезпечує впевненість у тому, що функція processCommand працює правильно для різних типів команд. У разі зміни реалізації функції або додавання нових команд, ці тести допоможуть швидко виявити помилки та забезпечити стабільність коду.

Модульне тестування є важливим інструментом для забезпечення якості програмного забезпечення. Використання Mocha, Chai та Jest дозволяє ефективно перевіряти функціональність окремих компонентів додатку, виявляти помилки на ранніх етапах розробки та забезпечувати стабільну роботу системи. Ретельне модульне тестування функції processCommand підтвердило її коректну роботу та готовність до інтеграції з іншими компонентами голосового асистента.

3.5.2 Інтеграційне тестування

Інтеграційне тестування є важливим етапом перевірки програмного забезпечення, яке дозволяє перевірити взаємодію між різними модулями та компонентами системи. Метою інтеграційного тестування є забезпечення коректної роботи всіх частин додатку разом. Для реалізації інтеграційного тестування у нашому проекті було використано такі інструменти, як Supertest та Mocha.

Інструменти для інтеграційного тестування

1. Supertest – це бібліотека для тестування HTTP-запитів у додатках Node.js. Вона дозволяє легко відправляти запити до серверу та перевіряти відповіді.

2. Mocha – це популярний фреймворк для тестування JavaScript, який дозволяє виконувати асинхронні тести та забезпечує гнучкі можливості налаштування.

Розглянемо приклад інтеграційного тестування для API нашого додатку. Основна мета – перевірити, що сервер коректно обробляє запити від клієнта та повертає правильні відповіді.

Розроблений сервер має простий API, який обробляє запити та повертає відповіді у форматі JSON. Для перевірки роботи API був написаний наступний тест:

```
const request = require('supertest');
const app = require('../src/server');
describe('API Integration Test', () => {
 it('should return hello message on GET /', (done) => {
  request(app)
   .get('/')
   .expect('Content-Type', /json/)
   .expect(200)
   .end((err, res) => {
    if (err) return done(err);
    expect(res.body).to.equal('hello');
    done();
   });
 });
});
```

Кожен тест виконує наступні дії:

1. Викликає метод request з бібліотеки Supertest, передаючи йому серверний додаток app.
2. Виконує HTTP-запит до сервера, у даному випадку GET-запит до кореневого маршруту /.
3. Перевіряє, чи відповідає сервер з правильним типом контенту (Content-Type: application/json) та статусом відповіді (200).
4. Перевіряє, чи повертає сервер правильний вміст у відповіді, у цьому випадку рядок "hello".

Виконання цього тесту забезпечує впевненість у тому, що сервер правильно обробляє запити та повертає очікувані відповіді. Якщо сервер не повертає очікуваний результат, тест буде провалений, що дозволяє виявити та виправити помилки на ранніх етапах розробки.

Інтеграційне тестування також може включати перевірку інших маршрутів та функцій API, таких як обробка POST-запитів, взаємодія з базою даних, авторизація користувачів тощо. Ось приклад тесту для перевірки POST-запиту:

```
describe('POST /api/command', () => {
  it('should process command and return response', (done) => {
    request(app)
      .post('/api/command')
      .send({ command: 'increase volume' })
      .expect('Content-Type', /json/)
      .expect(200)
      .end((err, res) => {
        if (err) return done(err);
        expect(res.body.response).to.equal('Volume increased');
        done();
      });
  });
});
```

Цей тест перевіряє, чи сервер правильно обробляє POST-запит, отримує команду від клієнта та повертає очікуваний результат.

Інтеграційне тестування є важливим інструментом для забезпечення якості програмного забезпечення. Використання Supertest та Mocha дозволяє ефективно перевіряти взаємодію між різними модулями та компонентами додатку, виявляти помилки на ранніх етапах розробки та забезпечувати стабільну роботу системи. Ретельне інтеграційне тестування API підтвердило його коректну роботу та готовність до використання в реальних умовах.

3.5.3 Юніт тестування

Юніт тестування є фундаментальною частиною процесу забезпечення якості програмного забезпечення. Воно дозволяє перевірити окремі частини коду в ізоляції, забезпечуючи правильну роботу кожного модуля. У проекті для реалізації юніт тестування використовувалися такі інструменти, як Jest, Mocha та Chai.

Інструменти для юніт тестування:

1. Jest – потужний фреймворк для тестування JavaScript, розроблений Facebook. Він забезпечує простоту налаштування, інтеграцію з різними інструментами та автоматичне створення знімків стану (snapshot testing).
2. Chai – бібліотека для асерцій, яка використовується разом з Mocha для перевірки очікуваних результатів тестів.

Основна мета юніт тестування полягає у перевірці правильності роботи окремих функцій, методів та класів. Це дозволяє виявити помилки на ранніх етапах розробки та забезпечити, щоб кожен модуль працював відповідно до специфікацій.

Розглянуто приклад юніт тестування для функції обробки голосових команд у додатку. Функція, яка тестується, називається processCommand. Вона приймає команду у вигляді тексту і повертає відповідний результат.

Функція processCommand обробляє різні голосові команди, такі як збільшення гучності, вимкнення звуку, відкриття YouTube тощо. Ось її реалізація:

```
function processCommand(command) {
  switch (command) {
    case 'increase volume':
      return 'Volume increased';
    case 'mute':
      return 'Volume muted';
    default:
```

```
    return 'Command not recognized';
  }
}
```

Для перевірки правильності роботи функції processCommand було написано кілька тестів з використанням Jest. Ось приклад юніт тесту:

```
const { processCommand } = require('../src/voiceAssistant');
describe('Voice Assistant Command Processing', () => {
  test('should return correct response for volume up command', () => {
    const command = 'increase volume';
    const response = processCommand(command);
    expect(response).toBe('Volume increased');
  });
  test('should return correct response for mute command', () => {
    const command = 'mute';
    const response = processCommand(command);
    expect(response).toBe('Volume muted');
  });
  test('should return error for unrecognized command', () => {
    const command = 'play music';
    const response = processCommand(command);
    expect(response).toBe('Command not recognized');
  });
});
```

Кожен тест виконує наступні дії:

1. Викликає функцію processCommand з певною командою.
2. Перевіряє, чи повертає функція очікуваний результат за допомогою методу expect з бібліотеки Jest.

У першому тесті перевіряється команда збільшення гучності, і очікується, що функція поверне рядок "Volume increased". У другому тесті перевіряється команда вимкнення звуку, і очікується, що функція поверне рядок "Volume

muted". У третьому тесті перевіряється невідома команда, і очікується, що функція поверне рядок "Command not recognized".

Виконання цих тестів забезпечує впевненість у тому, що функція processCommand працює правильно для різних типів команд. Якщо функція не повертає очікуваний результат, тест буде провалений, що дозволяє виявити та виправити помилки на ранніх етапах розробки.

Юніт тестування є важливим інструментом для забезпечення якості програмного забезпечення. Використання Jest, Mocha та Chai дозволяє ефективно перевіряти функціональність окремих компонентів додатку, виявляти помилки на ранніх етапах розробки та забезпечувати стабільну роботу системи. Ретельне юніт тестування функції processCommand підтвердило її коректну роботу та готовність до інтеграції з іншими компонентами голосового асистента.

3.6 Системні вимоги

Для коректної роботи голосового асистента, розробленого з використанням Node.js та React, необхідно забезпечити відповідність певним системним вимогам. Вони охоплюють як апаратні, так і програмні ресурси, які необхідні для успішного встановлення та функціонування додатку.

Щоб запускати голосовий асистент, рекомендується мати комп'ютер із мінімум двоядерним процесором з тактовою частотою 2.5 ГГц або вище, 4 ГБ оперативної пам'яті та щонайменше 500 МБ вільного місця на жорсткому диску для встановлення та роботи додатку. Також потрібна наявність звукової карти та мікрофону для обробки голосових команд.

Для коректної роботи додатку необхідно мати встановлену операційну систему Windows 10, macOS 10.15 або новішу, або Ubuntu 18.04 чи новішу. Node.js версії 14.0 або новішої забезпечує середовище для виконання серверного коду, а npm (Node Package Manager) версії 6.0 або новішої використовується для управління залежностями проекту. Також рекомендується використовувати останні версії браузерів Google Chrome, Firefox або Microsoft Edge для

забезпечення роботи клієнтської частини додатку. Git необхідний для клонування репозиторію з кодом проекту.

Для реалізації функціоналу голосового асистента використовуються додаткові бібліотеки та інструменти, такі як Puppeteer для автоматизації взаємодії з веб-сторінками, Socket.IO для забезпечення реального часу комунікації між клієнтом та сервером, Mocha та Chai для тестування, а також Speech Recognition API для обробки голосових команд у браузері.

Коректна робота додатку також залежить від наявності доступу до Інтернету, що необхідно для отримання оновлень, взаємодії з зовнішніми сервісами, такими як API погоди та YouTube, а також для підтримки зв'язку між клієнтською та серверною частинами додатку.

Дотримання вищевказаних системних вимог є необхідним для забезпечення коректної роботи голосового асистента. Це включає відповідні апаратні ресурси, встановлене програмне забезпечення та бібліотеки, а також доступ до Інтернету. Забезпечення відповідності цим вимогам дозволить користувачам успішно встановити, налаштувати та використовувати голосовий асистент, розроблений з використанням Node.js та React.

Забезпечення відповідності зазначеним системним вимогам є критичним кроком для успішної роботи голосового асистента. Недостатність ресурсів або неправильна конфігурація можуть призвести до непередбачуваної поведінки або навіть до повного виходу з ладу додатку. Саме тому важливо не лише виконати початкові вимоги, але й забезпечити регулярне оновлення програмного забезпечення та перевірку його належного функціонування.

При встановленні додатку слід звернути увагу на налаштування параметрів у файлі config.js. Зокрема, необхідно правильно вказати порт, на якому буде запущено сервер, а також налаштувати параметри підключення до бази даних, якщо вона використовується. Додаткова увага повинна бути приділена налаштуванню клієнтських параметрів, таких як URL серверу, до якого буде підключатися клієнтська частина.

Після успішного встановлення та налаштування голосового асистента, користувачі можуть розпочати його використання. Для цього потрібно відкрити браузер та ввести URL, на якому запущено додаток (наприклад, http://localhost:3000). На головній сторінці додатку буде представлений інтерфейс з кнопкою для активації голосового асистента. Натисканням на цю кнопку користувач активує голосового асистента, після чого можна вводити голосові команди, такі як "increase volume" або "open YouTube". Асистент обробить команду та виконає відповідну дію. Також у додатку передбачено відображення історії запитів, де можна переглядати попередні команди та результати їх виконання.

У додатку передбачено декілька розширених функцій, що робить його більш гнучким і корисним для користувачів. Наприклад, можливість відправки електронних листів шляхом голосових команд, що значно полегшує комунікацію. Для цього можна використовувати команду "send email to [email] with subject [subject] and message [message]". Ще однією корисною функцією є перевірка погоди. Для цього використовується команда "check weather in [city]", яка надає користувачеві інформацію про погоду у зазначеному місті. Ці функції роблять голосовий асистент універсальним інструментом для різноманітних завдань.

Важливо зазначити, що коректна робота додатку також залежить від стабільності Інтернет-з'єднання. Це необхідно для отримання оновлень, взаємодії з зовнішніми сервісами, такими як API погоди та YouTube, а також для підтримки зв'язку між клієнтською та серверною частинами додатку. У разі відсутності Інтернет-з'єднання певні функції додатку можуть бути недоступними.

Забезпечення відповідності всім зазначеним вимогам є ключовим для успішного використання голосового асистента. Це включає як належну апаратну конфігурацію, так і правильне налаштування програмного забезпечення. Регулярні оновлення та підтримка системи також є важливими для забезпечення її стабільної та ефективної роботи. Таким чином, користувачі зможуть отримати

максимальну користь від функціональності голосового асистента, розробленого з використанням Node.js та React.

3.7 Інструкція користувачеві

Для встановлення голосового асистента необхідно спершу скачати репозиторій з кодом. Це можна зробити, перейшовши на GitHub-сторінку проекту та скопіювавши URL репозиторію. Після цього у терміналі виконується команда git clone <URL репозиторію>, яка копіює проект на локальний комп'ютер. Далі потрібно перейти до директорії проекту за допомогою команди cd <ім'я директорії> і виконати команду npm install, щоб встановити всі необхідні залежності. Після цього для запуску серверної частини додатку виконується команда npm start.

Для налаштування додатку необхідно відкрити файл config.js та внести відповідні зміни. Зокрема, слід вказати порт, на якому буде запущено сервер, а також налаштувати параметри підключення до бази даних, якщо такі є. Крім того, потрібно вказати URL серверу, до якого буде підключатися клієнтська частина.

Після встановлення та налаштування додатку користувач може розпочати його використання. Для цього слід перейти до браузера та ввести URL, на якому запущено додаток (наприклад, http://localhost:3000). На головній сторінці додатку користувач бачить інтерфейс з кнопкою для виклику голосового асистента. Для активації голосового асистента необхідно натиснути на кнопку. Після активації слід вимовити голосову команду, наприклад, "increase volume" або "open YouTube". Асистент обробить команду та виконає відповідну дію. На боковій панелі доступна історія запитів, де можна переглядати попередні команди та результати їх виконання.

Додаток також підтримує декілька розширених функцій. Наприклад, для відправки електронних листів можна використовувати команду "send email to

[email] with subject [subject] and message [message]". Для перевірки погоди можна використовувати команду "check weather in [city]", яка надасть інформацію про погоду у зазначеному місті.

ВИСНОВКИ

Мета роботи виконана за рахунок створення клієнт-серверного застосунку голосового асистенту, що буде допомагати людині спростити рутинні справи та зможе мати можливість більш простого додавання нового функціоналу в порівнянні з іншими аналогами.

В процесі роботи було виконано наступні задачі: проведено аналіз предметної області, обрано програмні технології, розробленофункціонуючу систему на основі обраних технологій та протестовано розроблений програмний продукт

Підсумовуючи виконану бакалаврську роботу, можна виділити кілька основних висновків та рекомендацій, які стосуються процесу розробки голосового асистента з використанням клієнт-серверної архітектури на базі Node.js та React.

По-перше, важливо підкреслити актуальність теми. Розробка голосових асистентів є важливим напрямком сучасних технологій, що сприяє полегшенню повсякденних завдань та підвищенню зручності користувачів. Аналіз ринку показав зростаючий попит на такі рішення, зокрема серед людей з обмеженими можливостями та зайнятих професіоналів. Це підтверджує необхідність подальших досліджень та розробок у цій сфері.

По-друге, було проведено глибокий аналіз предметної області, який включав вивчення вимог до голосових асистентів та визначення підстав для створення нового додатку. Було визначено, що ключовими аспектами є інтуїтивно зрозумілий інтерфейс, швидка реакція на голосові команди та можливість легко додавати новий функціонал.

По-третє, обрані програмні технології, а саме Node.js та React, продемонстрували свою ефективність у розробці сучасних веб-додатків. Аналіз ринку інструментів розробки показав, що ці технології відповідають вимогам щодо створення масштабованих та зручних у використанні систем. Node.js

забезпечив швидку обробку серверних запитів, а React – динамічність та інтерактивність клієнтської частини.

Процес розробки включав декілька етапів: проектування архітектури системи, реалізацію функціональних модулів, інтеграцію компонентів та тестування готового продукту. Розроблена система продемонструвала високу продуктивність та стабільність роботи під час тестування, що підтверджує правильність обраних підходів та технологій.

Важливим аспектом стало тестування програмного продукту. Було проведено як модульне, так і інтеграційне тестування, що дозволило виявити та усунути недоліки на ранніх етапах розробки. Це забезпечило високу якість кінцевого продукту та його готовність до використання реальними користувачами.

Практичне значення отриманих результатів полягає в розробці функціонуючого голосового асистента, який спрощує рутинні завдання користувачів та забезпечує можливість додавання нового функціоналу. Цей продукт може бути використаний як основа для подальшого розвитку голосових асистентів, а також як приклад для розробки інших веб-додатків з використанням сучасних технологій.

Таким чином, виконана робота підтверджує доцільність та ефективність використання Node.js та React для створення голосових асистентів. Отримані результати можуть бути використані для подальших досліджень та розробок у сфері інтелектуальних систем взаємодії з користувачем. Розроблений продукт має потенціал для подальшого вдосконалення та розширення функціональності, що відкриває нові перспективи для його використання в різних сферах життя та бізнесу.

Виконана робота заклала міцний фундамент для подальших досліджень та розробок у галузі голосових асистентів, що сприятиме розвитку технологій та підвищенню якості життя користувачів.

СПИСОК ВИКОРИСТАНИХ ПОСИЛАНЬ

1. Безкоштовні веб-підручники для веб-розробки https://www.schoolsw3.com/index.php.

2. Голосове керування – Вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/Голосове_керування.

3. Голосові помічники: що це і навіщо вони HR [Ok Google, Alexa, Siri, Cortana] | HURMA. HURMA. URL: https://hurma.work/blog/voice-assistants-shho-cze-i-navishho-voni-hr-ok-google-alexa-siri-cortana/.

4. ДСТУ ISO/IEC/IEEE 16326:2015 Розроблення систем та програмного забезпечення. Процеси життєвого циклу. Керування проектами (ISO/IEC/IEEE 16326:2009, IDT) URL : http://online.budstandart.com/ua/catalog/docpage.html?id_doc=67052. (дата звернення 21.05.2024)

5. Інформація та документація. Бібліографічне посилання Загальні положення та правила складання ДСТУ 8302:2015 введ. 01.07.2016. К.: ДП «УкрНДНЦ», 2016. 16 с.

6. Інформація та документація. Звіти у сфері науки і техніки. Структура та правила оформлювання: ДСТУ 3008-2015 введ. 01.07.2017. К.: ДП «УкрНДНЦ», 2016. 26 с.

7. ICO. URL: https://www.ukrcsm.kiev.ua/index.php/en/services-ua/standard-ua/inter-org-standardua/inter-org-iso-ua. (дата звернення 21.05.2024)

8. КЛІЄНТ-СЕРВЕРНА АРХІТЕКТУРА URL: https://training.qatestlab.com/blog/technical-articles/client-server-architecture/.

9. Лесик Р.О., Шмалюк В.А., Богач І.В. Розробка голосового асистента з використанням Node.js та React . Матеріали Всеукраїнської науково-практичної інтернет-конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2024)» 2024», Вінниця, 2024. URL: https://conferences.vntu.edu.ua/index.php/mn/mn2024/paper/view/21763. (дата звернення: 25.05.2024).

10.Лопез Г., Куессада Л., Гуереро Л. А., Alexa проти Siri проти Cortana проти Google Assistant: порівняння користувацьких інтерфейсів на основі мовлення. 2017. 241-250.

11.МЕЖДУНАРОДНЫЙ СТАНДАРТ ISO 9000. Cert academy. 2015. URL: http://isomanagement.com/wp-content/uploads/2018/09/ISO-9000-2015.pdf. (дата звернення 21.05.2024)

12.Методичні рекомендації до виконання, оформлення та захисту кваліфікаційної (бакалаврської) роботи здобувачами спеціальності 122 «Комп'ютерні науки» освітньої програми «Комп'ютерні науки». О. В. Зелінська, Т. В. Січко, Н. А. Потапова, К. О. Якубич. Вінниця: ДонНУ імені Василя Стуса, 2024. 27 с.

13.Перелік національних стандартів для Перелік Національних стандартів України для створення, впровадження та супроводження автоматизованих і інформаційних систем http://nbuv.gov.ua/node/1469 (дата звернення 21.05.2024)

14.Поначугін О. В., Пічужкіна Д. Ю., Смєкалова К. С. Наука без меж. Голосовий помічник як технологія обробки даних. 2020. 96 с.

15.Ракша Лариса, 21.12.2021 Okay, Google: історія розвитку голосових асистентів URL: https://bazilik.media/okay-google-istoriia-rozvytku-holosovykh-asystentiv/.

16.Уніфікована система організаційно-розпорядчої документації. Вимоги до оформлювання документів. ДСТУ 4163-2003; чинний від 01.09.2003. URL: http://zakon3.rada.gov.ua/rada/show/v0055609-03. (дата звернення 21.05.2024)

17.Що таке стандарт ISO? URL: https://onmedu.edu.ua/shho-take-standart-iso-9001/. (дата звернення 21.05.2024)

18.Що таке API RESTful? URL: https://aws.amazon.com/ru/what-is/restful-api/.

19.Як створити веб-додаток: типи, переваги, принцип роботи URL: https://wezom.com.ua/ua/blog/kak-sozdat-veb-prilozhenie.

20.David Kopec. Classic Computer Science Problems in Swift. Essential techniques for practicing programmers. 2018 ISBN 9781617294891 224c

21. How to build a website using HTML and CSS URL: https://www.browserstack.com/guide/build-a-website-using-html-css.

22. How To Create a Web Server in Node.js with the HTTP Module URL: https://www.digitalocean.com/community/tutorials/how-to-create-a-web-server-in-node-js-with-the-http-module.

23. Morton M. Astrahan, Mike W. Blasgen, Donald D. Chamberlin, Kapali P. Eswaran, Jim Gray, Patricia P. Griffiths, W. Frank King III, Raymond A. Lorie, Paul, R. McJones, James W. Mehl, Gianfranco R. Putzolu, Irving L. Traiger, Bradford W. Wade, Vera Watson. System R: Relational Approach to Database Management. ACM, Transactions on Database Systems, 1(2), 1976, 97-137.

24. Nabendu Biswas Beginning React and Firebase. 1st Ed. ISBN 9781484278116 184c

25. SpeechRecognition. PyPI. URL: https://pypi.org/project/SpeechRecognition/.

26. Speech-to-Text: Automatic Speech Recognition | Google Cloud. Google Cloud. URL: https://cloud.google.com/speech-to-text.

27. Voice Assistants Are Becoming Less Accurate, But Google Assistant is Still the Smartest: Report – Voicebot.ai. Voicebot.ai. URL: https://voicebot.ai/2019/10/31/voice-assistants-are-becoming-less-accurate-but-google-assistant-is-still-the-smartest-report/.

28. What is a Data Flow Diagram. Lucidchart. URL: https://www.lucidchart.com/pages/data-flow-diagram.

29. WebSockets та Реалізація Двостороннього Зв'язку між Клієнтом і Сервером URL: https://it-rating.ua/websockets-ta-realizatsiya-dvostoronnogo-zvyazku-mij-klientom-i-serverom.

ДОДАТКИ

ДОДАТОК А

Лістинг основної частини програми

MainCircle.tsx

```
import React, { useEffect, useState } from "react";
import "./style.scss";
import { getResultOfRecord } from "@src/helpers/getResultOfRecord";
import { IChatMessage } from "@src/common/interfaces";
import Circle from "./Circle/Circle";
import ChatComponent from "./ChatComponent";

const Recognition = window.SpeechRecognition ||
window.webkitSpeechRecognition;
const myRecord = new Recognition();
myRecord.lang = "en-US";

myRecord.continuous = true;

const MainCircle: React.FC = () => {
 const [isRecoStart, setIsRecoStart] = useState(false);
 const [chatMessages, setChatMessages] = useState<IChatMessage[]>([]);

 const startRecordHandler = () => {
  setIsRecoStart(true);
  myRecord.start();
 };

 const endRecordHandler = () => {
  setIsRecoStart(false);
  myRecord.stop();
```

```
};

myRecord.onresult = (e) => {
  getResultOfRecord({
    e,
    setChatMessages,
  });
};

const changeRecordStart = () => {
  if (!isRecoStart) {
    startRecordHandler();
  } else {
    endRecordHandler();
  }
};

useEffect(() => {
  const chatMessagesFromLS = localStorage.getItem("chatMessages");
  if (chatMessagesFromLS) {
    setChatMessages(JSON.parse(chatMessagesFromLS));
  }
  setTimeout(() => {
    speechSynthesis.getVoices();
  }, 50);
}, []);

useEffect(() => {
  localStorage.setItem("chatMessages", JSON.stringify(chatMessages));
}, [chatMessages]);
```

```
    return (
      <div className="main-bg">
        <Circle changeRecordStart={changeRecordStart} />
        <ChatComponent
          chatMessages={chatMessages}
          setChatMessages={setChatMessages}
        />
      </div>
    );
  };

  export default MainCircle;
```

ChatComponent.tsx

```
  import { IChatMessage, MessageOwnerEnum } from
"@src/common/interfaces";
  import React from "react";
  import { ReactComponent as CloseSVG } from "@images/close.svg";

  interface IProps {
    chatMessages: IChatMessage[];
    setChatMessages: React.Dispatch<React.SetStateAction<IChatMessage[]>>;
  }
  const ChatComponent: React.FC<IProps> = ({ chatMessages,
setChatMessages }) => {
    const clearChatHandler = () => {
      setChatMessages([]);
    };
```

```
    return (
      <div className="chat-wrapper">
        <div className="chat-wrapper__close-chat"
onClick={clearChatHandler}>
          <CloseSVG />
        </div>
        <div className="chat-wrapper__title">Chat History</div>
        <div className="chat-wrapper__chat">
          {chatMessages.map((item) => (
            <div
              key={item.id}
              className={`chat-wrapper__chat-item ${
                item.owner === MessageOwnerEnum.ME
                  ? "chat-wrapper__chat-item--right"
                  : ""
              }`}
            >
              <p>{item.message}</p>
            </div>
          ))}
        </div>
      </div>
    );
  };

  export default ChatComponent;
```

style.scss

```
.main-bg-circle{
   width:220px;
```

```
    height: 220px;
    border-radius: 50%;
    background-color: rgb(184, 65, 65);
    cursor: pointer;
}

.main-bg-circle-wrapper{
    position: absolute;
    top:50%;
    left:50%;
    transform: translate(-50%,-50%);
    z-index: 10;

    svg{
        width: 40%;
        height: 40%;

        position: absolute;
        top:50%;
        left: 50%;
        transform: translate(-50%,-50%);
        g{
            fill:white;
        }
    }
}

.main-bg-circle-shadow {
    width: 100%;
    height: 100%;
```

```
    background-color: rgb(18, 18, 22);
    border:1px solid rgb(184, 65, 65);
    border-radius: 50%;
    position: relative;
    z-index: -1;

}

.chat-wrapper  {
    position: fixed;
    top:15px;
    left:15px;
    bottom:0;
    height: 100vh;
    width:300px;
    display: flex;
    flex-direction: column;
    align-items: stretch;
    padding-bottom: 20px;
}
.chat-wrapper__title{
    color:rgb(184, 65, 65);
    font-size: 1.2rem;
    text-align: center;
    align-self: center;
    font-family: "Montserrat-bold";
}

.chat-wrapper__close-chat{
    position: absolute;
```

```
    top:4px;
    left:20px;
    cursor: pointer;

    svg{
      width: 16px;
      height: 16px;
      fill:rgb(184, 65, 65);
    }
  }

  .chat-wrapper__chat {
    display: flex;
    flex-direction: column;
    align-items: stretch;
    margin-top: 20px;
    max-height: 100vh;
    overflow-y: scroll;
    padding:0 15px;
  }

  .chat-wrapper__chat-item{
    color:rgb(184, 65, 65);
    font-size: 1rem;
    position: relative;
    padding:10px 0 10px 5px;
    white-space: pre-wrap;
    box-sizing: border-box;
    font-family: "Montserrat-medium";
    p{
```

```
        padding-left: 5px;
    }
    &::before{
        content:"-";
        position: absolute;
        color:#fff;
        font-size: 0.9rem;
        top:50%;
        left:0;
        transform: translateY(-50%);
    }
}

.chat-wrapper__chat-item--right{
    align-self: flex-end;
    text-align: right;

}
```

index.module.js

```
import http from "http";
import express from "express";
import cors from "cors";
import puppeteer from "puppeteer";
import { Server } from "socket.io";
import fs from 'fs';
import pkg from "@deepgram/sdk";
import mic from "mic";

const { Deepgram} = pkg;
```

```
const deepgramApiKey = 'eb31540369ecc434a3f06af0f9c7f747e835dda7';
const pathToFile = './recorded_audio.wav';
const mimetype = 'audio/wav';
const deepgram = new Deepgram(deepgramApiKey);

const app = express();
const server = http.Server(app).listen(9999,()=>console.log('server has been started'));
const io = new Server(server,{
  cors:{
    origin: "http://localhost:3000",
    allowedHeaders: ["my-custom-header"],
    credentials: true
  }
});
const corsOptions ={
   origin:'*',
   credentials:true,
   optionSuccessStatus:200,
}

app.use(cors(corsOptions));
app.use(express.json());

app.get('/',(req,res)=>{
   res.send(JSON.stringify('hello'))
})

// const micInstance = mic({
```

```
//   rate: '16000',
//   channels: '1',
//   debug: true,
// });

// Створюємо запис до файлу
// const stream = fs.createWriteStream('record.wav');

// Починаємо запис
// micInstance.start();

// Додаємо обробники подій для отримання даних з мікрофона та їх запису до файлу
// micInstance.on('data', (data) => {
//   console.log('Recieved Input Stream: ' + data.length);
//   stream.write(data);
// });

// Зупиняємо запис через 10 секунд
// setTimeout(() => {
//   micInstance.stop();
//   stream.end();
// }, 10000);

let browser = null;
let page = null;

(async()=>{
```

```
        browser = await
puppeteer.connect({browserWSEndpoint:'ws://127.0.0.1:9222/devtools/browser/d40b
7fa8-d083-477b-b895-019efed13f70',defaultViewport:null}).catch((e)=>e);
        console.log(browser);
        page = await browser.newPage();
       })();

      io.on('connection',async (socket)=>{

        socket.on("make-volume-more",(data)=>{
          const { volume } = data;
          loudness.setVolume(volume);
        })
        socket.on("make-the-volume-mute",()=>{
          loudness.setMuted();
        })
        socket.on("open-youtube",async ()=>{
          await page.goto('https://www.youtube.com/', { waitUntil: 'networkidle2' });
        })
        socket.on("search-in-youtube",async(data)=>{
          const {searchText} = data;
          await page.goto(
`https://www.youtube.com/results?search_query=${searchText
          .split(" ")
          .join("+")}`)

        })
      })
```

ДОДАТОК Б

Декларація щодо унікальності текстів роботи та невикористання матеріалів інших авторів без посилань

ДЕКЛАРАЦІЯ

про дотримання академічної доброчесності

Я, _____

_____

Повністю вказується ПІБ та статус (посада для працівників, освітня (освітньо-наукова) програма – для здобувачів вищої освіти)

що нижче підписалась/підписався, розуміючи та підтримуючи загальновизнані засади справедливості, доброчесності та законності,

ЗОБОВ'ЯЗУЮСЬ:

дотримуватися принципів та правил академічної доброчесності, що визначені законодавством України, локальними нормативними актами Донецького національного університету імені Василя Стуса, положеннями, правилами, умовами, визначеними іншими суб'єктами, та не допускати їх порушення.

ПІДТВЕРДЖУЮ:

що мені відомі положення статті 42 Закону України «Про освіту»;

що у даній роботі не представляла/представляв чиїсь роботи повністю або частково як свої власні. Там, де я скористалася/скористався працею інших, я зробила/зробив відповідні посилання на джерела інформації;

що дана робота не передавалась іншим особам і подається вперше, не порушує авторських та суміжних прав закріплених статтями 21-25 Закону України «Про авторське право та суміжні права», а дані та інформація не отримувались в недозволений спосіб.

УСВІДОМЛЮЮ:

що ця робота може бути перевірена університетом на плагіат або інші порушення академічної доброчесності, в тому числі з використанням спеціалізованих сервісів;

що у разі порушення академічної доброчесності, до мене можуть бути застосовані процедури, передбачені законодавством України та Кодексом академічної доброчесності та корпоративної етики Донецького національного університету імені Василя Стуса, іншими локальними нормативними актами університету, та я можу бути притягнута/притягнутий до академічної відповідальності.

_____ _____

(дата) (підпис)