

Міністерство освіти і науки України

Луцький національний технічний університет

(повне найменування закладу вищої освіти)

Факультет комп'ютерних та інформаційних технологій

(повне найменування факультету)

Кафедра комп'ютерної інженерії та охоронних систем

(повне найменування кафедри)

КВАЛІФІКАЦІЙНА РОБОТА
ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»
ГОЛОСОВИЙ АСИСТЕНТ ІЗ ФУНКЦІЯМИ РОЗУМНОГО
ДОМУ
VOICE ASSISTANT WITH SMART HOME FEATURES

спеціальність 123 Комп'ютерна інженерія

(шифр і назва спеціальності)

освітня програма Комп'ютерна інженерія

(назва освітньої програми)

Виконав: здобувач вищої освіти
групи КІ-42
Паламарчук Владислав
Олександрович

(підпис)

Керівник:
к.т.н., доцент
Бортник Катерина Яківна

(підпис)

Кваліфікаційну роботу
допущено до захисту
«_____» червня _____ 2026 р.
Гарант освітньої програми:
к.т.н., доцент
Лавренчук Світлана Василівна

(підпис)

Луцьк – 2026 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних та інформаційних технологій

Кафедра комп'ютерної інженерії та безпеки

Ступінь вищої освіти: бакалавр

Галузь знань: 12 Інформаційні технології

Спеціальність: 123 Комп'ютерна інженерія

Освітня програма: «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ

Завідувач кафедри

доц. Т. Терлецький

« 23 » 12 2025 р.

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Паламарчуку Владиславу Олександровичу

(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи Голосовий асистент із функціями розумного дому

Керівник роботи к.т.н., доцент Бортник Катерина Яківна

затверджені наказом закладу вищої освіти від «20» грудня 2025 року № 536/01-02

2. Строк подання здобувачем вищої освіти кваліфікаційної роботи 01.06.2026 р.

3. Вихідні дані до роботи Джерелом розробки є науково-технічна література та публікації в періодичних виданнях з даного питання, опубліковані зарубіжні та вітчизняні роботи в даній області, різні інтернет-ресурси технічного спрямування

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити):

Вступ

Дослідження предметної області та існуючих рішень

Вибір апаратного та програмного забезпечення

Проектування, реалізація та тестування системи

Висновки

5. Перелік графічного (ілюстративного) матеріалу:

Схеми роботи проекту

Схеми розподілу функцій

Скріншоти логування сервера

Структурна схема

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
<i>Дослідження предметної області та існуючих рішень</i>	<i>Бортник К. Я., доцент</i>		
<i>Вибір апаратного та програмного забезпечення</i>	<i>Бортник К. Я., доцент</i>		
<i>Проектування, реалізація та тестування системи</i>	<i>Бортник К. Я., доцент</i>		
<i>Нормоконтроль</i>	<i>Багнюк Н. В., доцент</i>		
<i>Гарант ОП</i>	<i>Лавренчук С. В., доцент</i>		
<i>Показник запозичень тексту</i>		%	
<i>Академічна доброчесність</i>	<i>Міскевич О. І., ст. викладач</i>		

7. Дата видачі завдання 23.12.2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	<i>Огляд літератури із досліджуваної проблеми, аналіз предметної області та наявних рішень</i>	до 10.02.2026 р.	
2.	<i>Дослідження предметної області та існуючих рішень</i>	до 02.03.2026 р.	
3.	<i>Вибір апаратного та програмного забезпечення</i>	до 02.04.2026 р.	
4.	<i>Проектування, реалізація та тестування системи</i>	до 10.04.2026 р.	
5.	<i>Представлення остаточного варіанту кваліфікаційної роботи керівникові</i>	до 01.05.2026 р.	
6.	<i>Нормоконтроль</i>	до 23.05.2026 р.	
7.	<i>Інструментальна перевірка на академічний плагіат</i>	до 26.05.2026 р.	
8.	<i>Здача кваліфікаційної роботи та всіх супровідних документів на кафедру</i>	до 28.06.2026 р.	

Здобувач вищої освіти

(підпис)

Владислав ПАЛАМАРЧУК

(прізвище, ініціали)

Керівник кваліфікаційної роботи

(підпис)

Катерина БОРТНИК

(прізвище, ініціали)

АНОТАЦІЯ

Паламарчук В. О. Голосовий асистент із функціями розумного дому.
Рукопис.

Кваліфікаційна робота бакалавра ОП «Комп'ютерна інженерія» спеціальності 123 Комп'ютерна інженерія. Луцький національний технічний університет. Луцьк, 2026.

Кваліфікаційна робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків.

Перший розділ присвячено дослідженню предметної області та аналізу існуючих рішень. У ньому розглянуто особливості голосових асистентів, принципи побудови систем розумного дому, методи розпізнавання, обробки та синтезу мовлення, а також сучасні голосові асистенти й платформи розумного дому.

У другому розділі здійснено вибір та обґрунтування апаратного і програмного забезпечення. Розглянуто використання Raspberry Pi 5 як центрального серверного вузла, ESP32-S3 для збору та передавання аудіоданих, мікрофона INMP441 і підсилювача MAX98357A. Також проаналізовано програмні засоби Flask, Vosk, WakeNet, Piper TTS, MQTT, HTTP та Groq API.

Третій розділ присвячено проєктуванню, реалізації та тестуванню системи голосового асистента. Описано структурну архітектуру системи, алгоритм обміну даними, серверну частину, обробку голосових команд, синтез мовлення та керування пристроями розумного дому через MQTT і HTTP.

Ключові слова: голосовий асистент, розумний дім, ESP32-S3, Raspberry Pi, INMP441, MAX98357A, Flask, Vosk, Piper TTS, MQTT, HTTP.

ANNOTATION

Palamarchuk V. Voice Assistant with Smart Home Features. Manuscript.

Bachelor's qualification thesis of the Educational Program "Computer Engineering", specialty 123 Computer Engineering. Lutsk National Technical University. Lutsk, 2026.

The qualification thesis consists of an introduction, three chapters, conclusions, a list of references, and appendices.

The first chapter is devoted to the study of the subject area and the analysis of existing solutions. It examines the features of voice assistants, the principles of smart home system design, methods of speech recognition, processing and synthesis, as well as modern voice assistants and smart home platforms.

The second chapter presents the selection and justification of the hardware and software. It considers the use of the Raspberry Pi 5 as the central server node, the ESP32-S3 for collecting and transmitting audio data, the INMP441 microphone, and the MAX98357A amplifier. The chapter also analyzes software tools such as Flask, Vosk, WakeNet, Piper TTS, MQTT, HTTP, and Groq API.

The third chapter is devoted to the design, implementation, and testing of the voice assistant system. It describes the structural architecture of the system, the data exchange algorithm, the server side, voice command processing, speech synthesis, and smart home device control via MQTT and HTTP.

Keywords: voice assistant, smart home, ESP32-S3, Raspberry Pi, INMP441, MAX98357A, Flask, Vosk, Piper TTS, MQTT, HTTP.

ЗМІСТ

АНОТАЦІЯ	4
ВСТУП	8
РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ	10
1.1 Призначення, функції та особливості голосових асистентів	10
1.2 Принципи побудови та компоненти систем розумного дому	11
1.3 Методи розпізнавання, обробки та синтезу мовлення	13
1.4 Аналіз існуючих асистентів і платформ розумного дому	15
1.5 Формування вимог до розроблюваної системи	16
РОЗДІЛ 2 ВИБІР АПАРАТНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ	18
2.1 Обґрунтування вибору одноплатного комп'ютера Raspberry Pi	18
2.2 Використання мікроконтролера ESP32 для збору та передавання аудіоданих	20
2.3 Аналіз компонентів аудіозахоплення та відтворення	22
2.4 Вибір програмних засобів для активації, розпізнавання та обробки голосових команд	24
2.4.1 Whisper vs Vosk	25
2.4.2 Виявлення wake word	26
2.4.3 Класифікація намірів	26
2.4.4 Відповіді на запитання від LLM через Groq API	27
2.4.5 Синтез мовлення	28
2.5. Вибір засобів інтеграції з пристроями розумного дому	29
2.5.1 Протоколи взаємодії	29
2.5.2 Веб-панель адміністрування IoT	30
РОЗДІЛ 3 ПРОЄКТУВАННЯ, РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ ..	32
3.1 Розробка структурної архітектури системи	32
3.2 Проєктування загального алгоритму роботи системи	36
3.3 Реалізація периферійного модуля на ESP32-S3	41

3.4 Реалізація серверної частини на Raspberry Pi	48
3.5 Реалізація керування пристроями розумного дому	52
3.6 Тестування системи та оцінка результатів	57
ВИСНОВКИ	61
ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ	63

ВСТУП

Актуальність теми. Розвиток технологій штучного інтелекту та інтернету речей відкриває нові можливості для створення голосових асистентів з певними елементами розумного дому.

Сучасні комерційні рішення, такі як Amazon Alexa, Google Assistant та Apple Siri, мають низку недоліків, серед яких залежність від хмарних сервісів, обмежена підтримка української мови, закритість програмної архітектури та недостатня гнучкість під час адаптації до індивідуальних потреб користувача. У зв'язку з цим особливо актуальною є розробка власного голосового асистента на базі доступних апаратних платформ, який забезпечуватиме локальну або гібридну обробку команд, підтримку української мови, керування пристроями та можливість подальшого розширення функціоналу. Такий підхід дає змогу підвищити автономність системи, покращити конфіденційність даних користувача та зменшити залежність від сторонніх сервісів.

Метою роботи є розробка голосового асистента із функціями розумного дому на базі мікроконтролера ESP32 та мікрокомп'ютера Raspberry Pi, який забезпечуватиме обробку голосових команд, підтримку української мови, керування IoT-речами та надання інформаційних відповідей користувачу.

Об'єкт дослідження – системи голосового керування пристроями розумного будинку на базі мікроконтролерів та методи обробки природної мови.

Предмет дослідження – програмно-апаратна реалізація голосового асистента із функціями розумного дому на базі ESP32 і Raspberry Pi з підтримкою української мови та інтеграцією з IoT-пристроями.

Завдання, які необхідно виконати:

- реалізувати механізм приймання, передавання та обробки голосових команд користувача;
- розробити архітектуру голосового асистента з розподілом функцій між ESP32, Raspberry Pi та модулями керування пристроями;

- дослідити сучасні підходи до побудови голосових асистентів і наявні рішення для систем розумного дому;
- візуалізувати архітектуру системи, алгоритми її роботи та взаємодію між компонентами;
- спроектувати модуль керування пристроями на основі голосових команд;
- запропонувати підхід до підвищення автономності, гнучкості та зручності використання системи в побутових умовах.

РОЗДІЛ 1

ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ

У сучасних умовах розвитку інформаційних технологій особливого поширення набувають системи, що поєднують можливості штучного інтелекту, обробки мовлення та Інтернету речей. Одним із найбільш перспективних напрямів є створення голосових асистентів, інтегрованих із системами розумного дому. Це забезпечує зручну взаємодію користувача та пристроїв, автоматизує звичайні рутинні дії, полегшує побут. Ефективність цих систем залежить в більшості від архітектури, обробки голосових команд та можливості інтеграції нових пристроїв.

1.1 Призначення, функції та особливості голосових асистентів

Насамперед голосовий помічник – це роботизований продукт, який здатний виконувати певні задачі на основі власних алгоритмів, інформації, отриманої від користувача та інформації, доступної в глобальній мережі. Комерційні платформи на кшталт Alexa, Google Assistant та Siri орієнтовані саме на таку модель взаємодії: користувач звертається до системи голосом, після чого вона виконує інформаційні, організаційні або керувальні функції [1].

До основних функцій голосових асистентів належать:

- розпізнавання мовлення користувача;
- пошук та надання інформації;
- керування програмами і сервісами;
- виконання команд автоматизації;
- взаємодія з пристроями розумного дому;
- формування голосової відповіді.

З практичної сторони асистент є інтелектуальним інтерфейсом між людиною і пристроями, також перевагою є те, що немає необхідності

працювати з меню та складними налаштуваннями, оскільки достатньо голосової команди.

У загальному голосовий помічник містить певні обов'язкові компоненти:

- активація системи за ключовим словом або кнопкою;
- запис голосового фрагмента;
- перетворення аудіосигналу в текст;
- аналіз тексту та визначення команди;
- виконання дії або формування відповіді;
- відтворення результату через синтез мовлення.

Така структура є доцільною оскільки легка в реалізації, розподіленні між компонентами та забезпечує виконання всіх необхідних функцій. До прикладу легкий мікроконтролер може виконувати захоплення аудіо з мікрофону та озвучувати результат, а основний мозок бути окремо на сервері для обробки складних команд.

Сучасні голосові асистенти можуть працювати за хмарною, локальною або гібридною моделлю. Хмарні передбачають передавання і обробку повного циклу аудіо-запису на сервері. Локальний підхід, навпаки, дозволяє обробляти команди безпосередньо на пристрої або в межах локальної мережі, що підвищує автономність і контроль над даними. Гібридна модель передбачає розподілення функцій між вище зазначеними моделями [2].

Для розробки власного асистента найбільш доцільною є використання саме з змішаної моделі, оскільки це дозволяє підлаштувати систему під свої власні потреби, але й не обмежує її в використанні більш складної зовнішньої обробки для інформаційних запитів.

1.2 Принципи побудови та компоненти систем розумного дому

Головною особливістю розумного дому є здатність об'єднувати різні пристрої в єдину систему, у якій окремі компоненти можуть обмінюватися даними та реагувати на події в режимі реального часу.

Архітектура розумного дому традиційно складається з трьох рівнів (рисунок 1.1): фізичний (пристрої IoT: сенсори, активатори, контролери), мережевий/комунікаційний (протоколи та мережі передачі даних), сервісний та логічний рівень (контролери, платформи, VA) з базами даних та логікою автоматизацій.

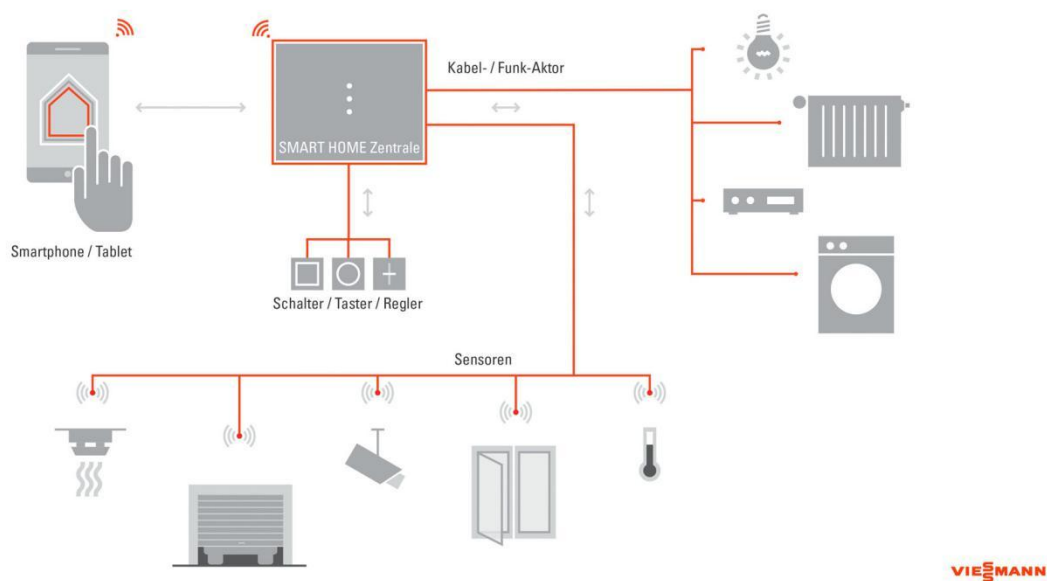


Рисунок 1.1 – Архітектура розумного дому [3]

Основними принципами побудови систем розумного дому є модульність, масштабованість, сумісність та автоматизація. Модульність означає, що система складається з окремих вузлів, які можуть додаватися або замінюватися без повної перебудови архітектури.

Масштабованість дозволяє поступово розширювати систему новими пристроями. Сумісність забезпечує роботу обладнання різних виробників у межах одного середовища. Автоматизація передбачає виконання дій за наперед визначеними сценаріями або у відповідь на події.

Для забезпечення цих принципів необхідно використовувати уніфіковані та загальнодоступні методи з'єднання й обміну даними. Такими шляхами зв'язку є Wi-Fi, Bluetooth, Zigbee, Z-Wave, Ethernet, MQTT та інші.

Для розроблюваної системи доцільним є використання Wi-Fi як основного способу передавання даних між периферійним мікроконтролером і основним пристроєм обробки інформації, оскільки він забезпечує достатню пропускну здатність для надсилання аудіофрагментів, простоту реалізації та сумісність із Raspberry Pi.

Ознакою розумного дому є підтримка сценаріїв автоматизації. Наприклад, система може вмикати освітлення за командою користувача, регулювати температуру, активувати сигналізацію або виконувати послідовність дій одним запитом. У цьому контексті голосове керування не лише дублює ручне керування, а й пришвидшує доступ до функцій системи, оскільки дозволяє передавати складні команди у звичній мовній формі.

1.3 Методи розпізнавання, обробки та синтезу мовлення

Одним з способів звернення до системи є активація по ключовому слову. Метод є енергоефективним та зводить кількість помилкових спрацювань до мінімуму саме тому більшість голосових помічників використовує саме його.

Система аналізує короткі аудіофрагменти на збіг із командою активації та після його виявлення переходить до повноцінної роботи (рисунк 1.2). Практично це можна розділити на два рівні: виявлення ключа та подальшої обробки аудіоряду на отримання команд [4]. Такий метод є оптимальним для реалізації асистента оскільки частина з отриманням звуку буде реалізована на ESP32, а складна обробка винесена на Raspberry Pi.

ASR використовує концепції обробки природної мови та машинного навчання для своєї розробки. Розпізнання включає в себе такі етапи розробки як запис звуку, попередня обробка аудіо, витяг функції, акустичне моделювання, схематизація мови та декодування [2].

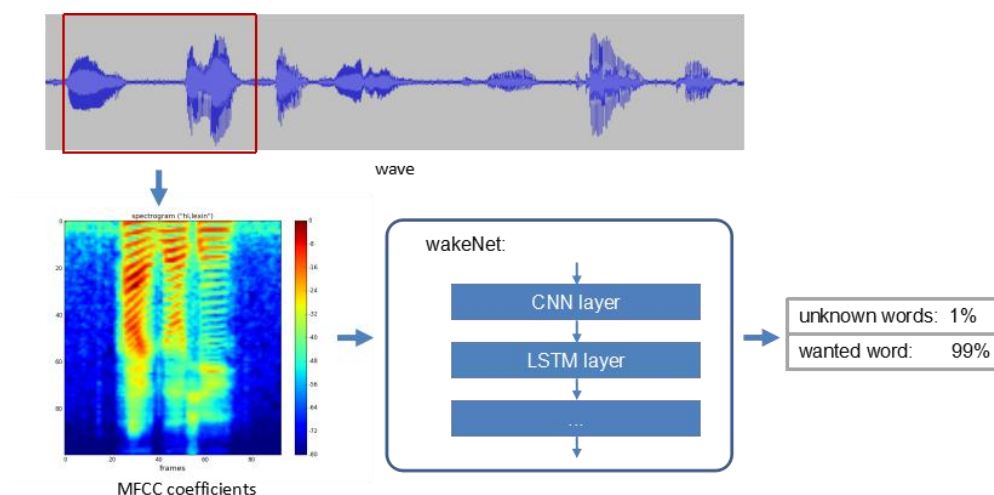


Рисунок 1.2 – Схема активації голосового асистента за ключовим словом [4]

Процес є складним, довгим та ресурсозатратним у випадку розробки його з нуля, проте існують готові рішення такі як Whisper та Vosk (таблиця 1.1). Основна різниця між ними в тому, що перший метод – це хмарний інструмент, який надає дуже точне перетворення аудіо в текст, другий же є локальним, але якість перетворення на порядок нижча, проте достатня для голосового асистента.

Таблиця 1.1 – Засоби розпізнавання та синтезу мовлення

Засіб	Призначення	Тип роботи	Переваги	Недоліки
WakeNet	активація	локальний	швидкість, енергоефективність	обмеження ключовими словами
Whisper	розпізнавання мовлення	локальний / хмарний	висока точність	вищі вимоги до ресурсів
Vosk	розпізнавання мовлення	локальний	офлайн робота, простота	нижча точність
Piper	синтез мовлення	локальний	швидкодія, автономність	залежність від якості моделі

Наступним етапом після розпізнавання тексту є обробка та визначення наміру. На цьому етапі реалізується аналіз змісту фрази, виділення ключових слів, визначення наміру та параметрів команди. Лише для системи розумного дому достатньо звичайних команд типу «вмикни світло», «яка температура»,

але для складних сценаріїв застосовуються моделі природної мови, проте вони в свою чергу потребують багато ресурсів та залежать від зовнішніх сервісів.

Завершальним етапом роботи голосового асистента є формування голосової відповіді. Це теж можна реалізувати як і локально так і через хмарні сервіси. Для проєкту краще обрати локальний підхід, оскільки він зменшує затримки та підвищує автономність системи. Piper описується як швидка локальна нейронна система text-to-speech, придатна для використання у вбудованих і домашніх рішеннях.

Таким чином, для голосового асистента із функціями розумного дому доцільно використовувати ключове слово для активації, локальне або напівлокальне розпізнання мовлення, автономну обробку типових команд та синтез мовлення.

1.4 Аналіз існуючих асистентів і платформ розумного дому

На даний момент є багато готових архітектурних рішень для голосових асистентів. Вони відрізняються за способом обробки даних, рівнем відкритості, можливостями інтеграції та залежністю від зовнішніх сервісів. Для вибору найбільш доцільного підходу до реалізації власної системи необхідно провести порівняльний аналіз найпоширеніших платформ.

Як видно з таблиці 1.2, комерційні рішення мають високий рівень готовності до використання, проте здебільшого залежать від хмарної інфраструктури та мають обмежені можливості адаптації до індивідуальних потреб користувача.

Таблиця 1.2 – Порівняльний аналіз голосових асистентів і платформ розумного дому

Назва платформи	Тип рішення	Переваги	Недоліки	Оцінка для проєкту
Amazon Alexa	хмарне	інтеграція, функціональність	залежність від хмари	середня

Продовження таблиці 1.2

Назва платформи	Тип рішення	Переваги	Недоліки	Оцінка для проекту
Google Assistant / Google Home	хмарне	точність, поширеність	закритість, хмара	середня
Apple Siri / HomeKit	закрите	стабільність, безпека	обмежена сумісність	низька
Home Assistant	відкрите	локальність, гнучкість	складність налаштування	висока

Серед розглянутих рішень найбільш придатним до реалізації є платформа Home Assistant, оскільки вона характеризується відкритістю, локальним керуванням та гнучкою інтеграцією з компонентами системи розумного дому.

1.5 Формування вимог до розроблюваної системи

На основі дослідження предметної області та аналізу існуючих рішень можна сформулювати основні вимоги до розроблюваного голосового асистента із функціями розумного дому.

Функціональні вимоги визначають основні можливості, які повинна забезпечувати розроблювана система для виконання свого призначення.

Розроблювана система повинна:

- виконувати активацію за ключовим словом;
- приймати та записувати голосову команду користувача;
- передавати аудіодані на центральний вузол обробки;
- розпізнавати мовлення;
- визначати намір команди;

- керувати пристроями розумного дому;
- формувати голосову відповідь;
- підтримувати розширення функціоналу новими командами та пристроями.

Вимоги охоплюють основні етапи роботи голосового асистента та визначають перелік завдань, які система повинна виконувати в процесі взаємодії.

До основних нефункціональних вимог належать:

- робота в локальній мережі без критичної залежності від хмарних сервісів;
- достатня швидкодія для побутового використання;
- модульність програмної та апаратної архітектури;
- можливість інтеграції з мікроконтролерами та IoT-пристроями;
- стабільність роботи при багаторазовому виконанні команд;
- зручність масштабування системи.

Нефункціональні вимоги визначають основні якісні характеристики системи та забезпечують її придатність до тривалого, стабільного й зручного використання.

З урахуванням практичного спрямування система має мати розподілену архітектуру. Мікроконтролер ESP32 може взяти на себе активацію, захоплення аудіо та передачу, весь складніший функціонал типу обробки мовлення, логіки керування та взаємодії з пристроями реалізовуватиметься на Raspberry Pi.

В результаті було досліджено предметну область голосових асистентів і систем розумного дому. Визначено, що їх ефективність залежить від якості розпізнаного мовлення, правильності інтерпретації команд та швидкості формування відповіді. Розглянуто, принципи побудови системи розумного дому та їх основні компоненти. Проаналізовано основні методи розпізнавання, обробки та синтезу мовлення, а також існуючі платформи голосового керування й домашньої автоматизації.

РОЗДІЛ 2

ВИБІР АПАРАТНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Вибір апаратного та програмного забезпечення є ключовим елементом при розробці голосового асистента з елементами розумного дому, оскільки від цього напряму залежить ефективність, затримка роботи команд, масштабність та інші не менш значні показники роботи системи.

Як апаратну основу доцільно розглядати розподілену архітектуру, у якій периферійний вузол відповідає за активацію та збір аудіоданих, а центральний сервер за обробку мовлення, виконання команд та взаємодію з пристроями розумного дому.

2.1 Обґрунтування вибору одноплатного комп'ютера Raspberry Pi

Доцільно використовувати Raspberry Pi в гібридній структурі оскільки саме на нього покладаються найбільш ресурсомісткі функції. У розроблюваній системі серверний вузол виконує складні задачі такі як розпізнавання мовлення (STT), класифікацію намірів (NLU), синтез мовлення (TTS), взаємодію з LLM через API та веб-сервер для керування IoT-пристроями.

Зважаючи на функціональні вимоги серверна платформа повинна відповідати таким критеріям:

- достатня потужність для запуску моделей розпізнавання та синтезу мовлення;
- підтримка OS Linux та мови програмування Python;
- наявність Ethernet або Wi-Fi для локальної мережі та зв'язку з ESP32;
- компактність та можливість цілодобової роботи.

Для обґрунтування вибору серверного вузла доцільно порівняти Raspberry Pi 5 з попередньою моделлю Raspberry Pi 4, яка також часто використовується в IoT-проєктах.

Як видно в таблиці 2.1, п'ята версія має вищу продуктивність у порівнянні з попередньою, більший максимальний обсяг оперативної пам'яті, що важливо для подальшого масштабування, та кращий запас ресурсів для одночасного виконання кількох задач [5].

Таблиця 2.1 – Порівняльна характеристика Raspberry Pi [5]

Параметр	Raspberry Pi 5	Raspberry Pi 4
Процесор	Cortex-A76, 2.4 ГГц	Cortex-A72, 1.8 ГГц
Оперативна пам'ять	до 16 ГБ	до 8 ГБ
Продуктивність	вища	нижча
Мережеві можливості	Wi-Fi, Bluetooth, Ethernet	Wi-Fi, Bluetooth, Ethernet
Доцільність для STT/TTS	висока	середня
Перспектива розширення	висока	обмеженіша

Якщо розглядати ПК чи ноутбук як серверного вузла, то в порівнянні з мікрокомп'ютерами він значно поступається в енергоефективності та ціні. По швидкості роботи та обчислювальних потужностях він значно перевищує їх, але для цілодобової роботи не підходить. Також варто зазначити, що обчислювальних потужностей плати цілком достатньо для виконання всіх функціональних вимог.

Отже, для проєкту доцільніше використати Raspberry Pi 5, оскільки вона має значно більше переваг у порівнянні з 4тою версією та звичайним ПК.

Завдяки підтримці Linux (таблиця 2.2) ця платформа дозволяє використовувати Python, Flask, Vosk, Piper TTS, MQTT-клієнти та інші програмні засоби, які необхідні для проєкту.

Таблиця 2.2 – Основні характеристики Raspberry Pi 5 [6]

Характеристика	Значення
Процесор	Broadcom BCM2712
Архітектура	64-бітна Arm Cortex-A76
Частота	2.4 ГГц

Продовження таблиці 2.2

Характеристика	Значення
Кількість ядер	4
Оперативна пам'ять	до 16 ГБ
Бездротовий зв'язок	Wi-Fi, Bluetooth
Дротовий зв'язок	Gigabit Ethernet
Операційна система	Linux / Raspberry Pi OS
Призначення в проєкті	сервер, STT, TTS, логіка команд

2.2 Використання мікроконтролера ESP32 для збору та передавання аудіоданих

ESP32 виконує роль «розумного» периферійного вузла: він постійно прослуховує аудіопотік із мікрофона через I2S, виявляє ключове слово, записує аудіо-файл та передає його на Raspberry Pi через HTTP POST.

Такий розподіл функцій дозволяє зробити систему більш ефективною. Мікроконтролер працює безпосередньо з аудіомодулем і може бути розміщений у зручному місці приміщення, тоді як мікрокомп'ютер може виконувати роль основного сервера. ESP32 має вбудований Wi-Fi-модуль, що дозволяє обмінюватись даними з центральним вузлом. Проте варто зазначити, що різні версії мають свої особливості. З таблиці 2.3 видно, що кожна модель має свої слабкі та сильні сторони. Важливими аспектами в реалізації проєкту є підтримка I2S для цифрового аудіо, інтегрований Wi-Fi та підтримка технології слова-активації. ESP32-S3 містить всі ці компоненти.

Таблиця 2.3 – Порівняння версій ESP32 [7]

Платформа	Процесор	Периферія	Переваги	Обмеження
ESP32-S3	2 ядра, Xtensa LX7, до 240 МГц	I2S, I2C, SPI, UART, GPIO	аудіо, AIoT, wake word, PSRAM	складніше налаштування

Продовження таблиці 2.3

Платформа	Процесор	Периферія	Переваги	Обмеження
ESP32-C3	1 ядро, RISC-V, до 160 МГц	I2S, I2C, SPI, UART, GPIO	дешевий, енергоефективний, простий	менша продуктивність, менше GPIO
ESP32-C6	1 ядро, RISC-V	I2C, SPI, UART, GPIO	сучасні IoT- протоколи, Matter- ready	надлишковий для аудіо, слабший для STT/wake word
ESP32- WROOM- 32	2 ядра, Xtensa LX6, до 240 МГц	I2S, I2C, SPI, UART, GPIO	поширений, дешевий, багато прикладів	старіша архітектура, гірше для AIoT

Надважливою функцією ESP32 є підтримка технології wake word, це дозволяє не обробляти постійно весь звуковий потік, а переходити до запису лише після виявлення фрази. В ESP32-S3 доцільно використовувати WakeNet із складу ESP-SR. Ця технологія є рушієм для виявлення слова-активатора, містить використання MFCC для виділення мовних ознак та підтримує формат 16 кГц, моно та 16-біт [4]. Для проєкту мікроконтролер може постійно аналізувати короткі аудіофрагменти після чого переходити до основної роботи – запису та передавання файлу на сервер.

Функціонально всі мікроконтролери даної серії майже не відрізняються, основна відмінність лише в розмірі Flash-пам'яті та PSRAM (таблиця 2.4).

Таблиця 2.4 – Порівняння характеристик пам'яті версій ESP32-S3 [8]

Версія ESP32-S3	Flash	PSRAM
ESP32-S3-WROOM-1-N8	8 МБ	немає
ESP32-S3-WROOM-1-N16	16 МБ	немає

Продовження таблиці 2.4

Версія ESP32-S3	Flash	PSRAM
ESP32-S3-WROOM-1-N8R8	8 МБ	8 МБ
ESP32-S3-WROOM-1-N16R8	16 МБ	8 МБ
ESP32-S3-WROOM-2-N32R16V	32 МБ	16 МБ
ESP32-S3-MINI-1-N8	8 МБ	немає
ESP32-S3-MINI-1-N4R2	4 МБ	2 МБ

Flash є постійною пам'яттю в якій зберігається програма, конфігурації, моделі wake word та інші компоненти, які необхідні для роботи системи. PSRAM – це оперативна пам'ять, яка використовується лише під час роботи пристрою, після вимкнення живлення очищається. В проєкті важлива наявність обох видів пам'яті оскільки нам потрібно створити тимчасовий буфер для зберігання запису з мікрофону перед надсиланням на сервер.

В результаті вибрано ESP32-S3-WROOM-1-N16R8 як тимчасовий варіант для реалізації оскільки в залежності від складності розробленої конфігурації необхідний об'єм Flash-пам'яті може змінюватись.

2.3 Аналіз компонентів аудіозахоплення та відтворення

Реалізація повноцінного периферійного вузла на базі ESP32 включає правильний підбір компонентів, які були б сумісні з мікроконтролером. В голосовому асистенті ключову роль відіграють мікрофон та динаміки, проте для відтворення звуку необхідний підсилювач на вихід звуку.

Для захоплення голосових команд у системі використовується цифровий MEMS-мікрофон INMP441. Його перевагою є наявність цифрового інтерфейсу I2S, що дозволяє передавати аудіосигнал безпосередньо на мікроконтролер без використання окремого аналого-цифрового перетворювача.

Він складається з датчика MEMS, обробки сигналу, аналого-цифрового перетворювача, фільтра згладжування, керування живленням та 24-бітного

інтерфейсу I2S промислового стандарту (таблиця 2.5). I2S дозволяє підключатися безпосередньо до цифрових процесорів, таких як DSP та мікроконтролери, без необхідності використання аудіокодеків у системі. Завдяки високому відношенню сигнал/шум, він є відмінним вибором для додатків ближнього поля [9].

Таблиця 2.5 – Основні характеристики INMP441

Характеристика	Значення
Тип	MEMS-мікрофон
Інтерфейс	I2S
Розрядність	24 біти
SNR	61 dBA
Чутливість	-26 dBFS
Частотний діапазон	60 Гц – 15 кГц
Споживання	близько 1.4 мА

Використання INMP441 є обґрунтованим, оскільки він добре сумісний з ESP32-S3, має цифровий вихід і забезпечує достатню якість аудіосигналу для подальшого розпізнавання мовлення.

Для відтворення голосової відповіді системи доцільно використати підсилювач MAX98357A. Це цифровий PCM/I2S Class D підсилювач, який приймає аудіодані через I2S і може безпосередньо працювати з невеликим динаміком.

MAX98357A не потребує окремого I2C-програмування і не вимагає зовнішнього MCLK-сигналу, що спрощує його підключення до ESP32-S3. MAX98357A/MAX98357B автоматично розпізнає до 35 PCM і TDM clocking schemes, а також спрощує роботу за рахунок відсутності потреби в MCLK [10].

У системі голосового асистента цей модуль разом із динаміком 3 Вт 4 Ом використовується для відтворення аудіофайлів, згенерованих модулем синтезу мовлення.

Для коректної роботи всіх модулів необхідно визначити єдиний формат всіх записів. WakeNet для ESP32-S3 працює з аудіо 16 кГц, mono, signed 16-bit також мікрофон INMP441 записує в даному форматі, що робить його логічним базовим вибором.

Найкращий тип збереження такого формату даних є WAV-файл оскільки він містить службовий заголовок з параметрами аудіо та сирі дані, що спрощує наступну обробку, збереження та відтворення, до того ж підтримується як на ESP32 так і на Raspberry Pi без попередніх обробок [11].

2.4 Вибір програмних засобів для активації, розпізнавання та обробки голосових команд

Серверну частину розроблюваної системи краще реалізувати мовою Python з використанням мікрофреймворку Flask. Він є легким і гнучким, ідеальним для побудови REST API на Raspberry Pi, не накладає обмежень на структуру проєкту, споживає мінімум ресурсів (таблиця 2.6).

Таблиця 2.6 – Переваги Flask [12]

Перевага	Пояснення
Простота	легко створити сервер для приймання аудіофайлів від ESP32
Легкість	не перевантажений зайвими модулями, добре підходить для Raspberry Pi
Python	зручно інтегрується з Vosk, Piper, Groq API, MQTT
HTTP API	легко зробити маршрути типу /upload, /status, /devices
Робота з файлами	зручно приймати, зберігати й обробляти WAV-записи
Гнучкість	можна додавати нові функції без повної перебудови проєкту
Веб-панель	можна зробити просту сторінку для керування IoT-пристроями

Продовження таблиці 2.6

Перевага	Пояснення
Локальна робота	сервер може працювати в локальній мережі без хмари

У межах цього проєкту Flask виконуватиме такі функції:

- приймання аудіофайлів;
- збереження записів;
- запуск модуля розпізнавання мовлення;
- передавання тексту в NLU;
- формування відповіді;
- повернення результату користувачу;
- реалізація вебпанелі адміністрування.

Цей мікрофреймворк найкраще підходить для голосового асистента з функціями розумного дому оскільки його можливості покривають всі необхідні функції сервера.

2.4.1 Whisper vs Vosk

Розглянуто два варіанти розпізнавання мовлення у системі: Whisper і Vosk. Обидва рішення можуть використовуватися для перетворення мовлення в текст, проте мають різні особливості.

Vosk – легка система на базі Kaldi. Vosk є чітким вибором для мобільних, вбудованих або високонавантажених застосунків, де ефективність важливіша за ідеальну точність. Моделі мають розмір лише ~50 МБ, підтримують потокове аудіо та є напрочуд якісними навіть для загальних задач диктування – за умови відносно чистого аудіо та нескладного словникового запасу [13].

Whisper так само є системою автоматичного розпізнавання мовлення, проте на відміну від Vosk навчена на більшій кількості годин, що забезпечує стійкість до шуму, акцентів та спеціалізованої лексики. Краща якість перетворення потребує більше часу та ресурсів, до того ж система побудована на обміні інформацією з хмарою.

Реалізація проєкту не вимагає надзвичайно точного розпізнання мовлення саме тому для дотримання локальності, простоти запуску та помірного використання ресурсів доцільніше використати Vosk-модель.

2.4.2 Виявлення wake word

Ключове слово можна реалізувати двома методами WakeNet та OpenWakeWord. Обидва засоби призначені для розпізнання короткої фрази активації, однак орієнтовані на різні платформи. WakeNet є частиною ESP-SR і спеціально призначений для роботи на мікроконтролерах Espressif. Він підходить для реалізації постійного очікування ключового слова без передавання аудіопотоку на Raspberry Pi [4].

OpenWakeWord є open-source бібліотекою для створення voice-enabled applications і має готові моделі для поширених фраз, однак більше орієнтований на запуск у Python/Linux-середовищі. Зважаючи на метод реалізації у першому обробка буде виконуватись на ESP32, у другому випадку нам доведеться транслювати аудіо на Raspberry Pi, що створюватиме додаткове навантаження на мережу та на сам центральний вузол.

Найкращим рішенням буде використовувати WakeNet технологію, оскільки це допоможе зекономити ресурси, до того ж ESP-SR надає велику кількість готових моделей wake word на кшталт «Hi Esp», «Alexa» та інші.

2.4.3 Класифікація намірів

Після розпізнавання мовлення система отримує текстову команду. Наступним кроком є визначення наміру користувача, тобто визначити яку дію потрібно виконати.

Прості команди подібні до «вимкни лампу», «яка температура» не потребують складних архітектурних рішень. У такій системі команда розбивається на намір і параметри. Наприклад, у фразі «увімкни світло в кімнаті» наміром є дія «увімкнути», а параметром – пристрій «світло» або зона «кімната». Перевага такого способу у тому що він швидкий, зрозумілий, легко розширюється та не потребує підключення до інтернету.

Складні ж команди або фрази в яких складно визначити намір звичайним методом до прикладу «в кімнаті надто темно», потребують сильного ускладнення NLU (Natural Language Understanding) або необхідно використовувати спеціалізовані інструменти. Одним із таких рішень є LLM (Large Language Model) зокрема модель llama-3.1-8b-instant яка розроблена спеціально для визначення наміру швидкість обрахунку досягає 1000+ токенів на секунду.

Для реалізації проекту найкраще підходить гібридний формат: прості команди через локальну NLU, складні через логічну модель.

2.4.4 Відповіді на запитання від LLM через Groq API

Голосовий асистент передбачає ще обробку команд, які несуть інформаційний характер і не мають відношення до системи розумного дому на кшталт «Яка погода?» або «Що таке Едельвейс?». Створюється необхідність використання великих мовних моделей через API.

Серед різних інтерфейсів прикладного програмування вирішено вибрати Groq API оскільки він швидкий та зручний в інтеграції з Flask сервером (таблиця 2.7).

Таблиця 2.7 – Порівняння API [14]

API	Переваги	Недоліки
Groq API	висока швидкість, низька вартість, OpenAI-сумісність	залежність від Інтернету, не локальна робота
OpenAI API	сильні моделі, стабільна екосистема, багато інструментів	вища вартість для потужних моделей
Anthropic Claude API	якісні відповіді, сильне reasoning/coding	вища ціна, окрема інтеграція
Google Gemini API	мультимодальність, інтеграція з Google, free tier	залежність від Google-екосистеми
Mistral AI API	open-weight моделі, європейська платформа	менш поширена екосистема

Оптимальна модель, на яку дається багато токенів та має низьку вартість це llama-3.3-70b-versatile на відміну від instant версії вона надає повну розгорнуту відповідь на запитання.

2.4.5 Синтез мовлення

Синтез мовлення необхідний для перетворення відповідей з LLM та результатів виконання команд на аудіо яке буде виводитись на ESP32. Серед безкоштовних варіантів можна виділити декілька значущих, їх порівняння наведено у таблиці 2.8.

Таблиця 2.8 – Порівняння засобів синтезу мовлення

Засіб	Тип роботи	Якість голосу	Українська мова	Вимоги до ресурсів	Переваги
RHVoice	локально	середня	є	низькі	швидкий, простий, офлайн
Voicebox / Qwen3-TTS	локально	висока	залежить від моделі	високі	клонування голосу, природність
Piper TTS	локально	добра	є моделі	низькі / середні	швидкий, легкий, офлайн
Microsoft SpeechT5	локально / API	висока	обмежено	середні / високі	нейронна модель, якість

Аспекти на які при виборі слід звернути увагу в генерації аудіо є швидкість, якість вимови, наявність мови синтезу в моделі та вимоги до ресурсів. Зважаючи на ці критерії найкращим рішенням буде обрати Piper TTS оскільки він задовольняє всі необхідні вимоги, зокрема містить і українську мову.

2.5. Вибір засобів інтеграції з пристроями розумного дому

Інтеграція голосового асистента з пристроями розумного дому є одним із важливих етапів розробки. Після розпізнавання голосової команди та визначення наміру користувача система повинна передати відповідну команду виконавчому пристрою: реле, лампі, розетці, датчику або іншому IoT-модулю.

2.5.1 Протоколи взаємодії

MQTT (Message Queuing Telemetry Transport) – це легкий протокол обміну командами, який широко використовується в системах Інтернету речей. Він працює за системою publish/subscribe, тобто пристрої не обмінюються між собою напряму, а лише отримують вказівки від MQTT-брокера (рисунок 2.1) [15].

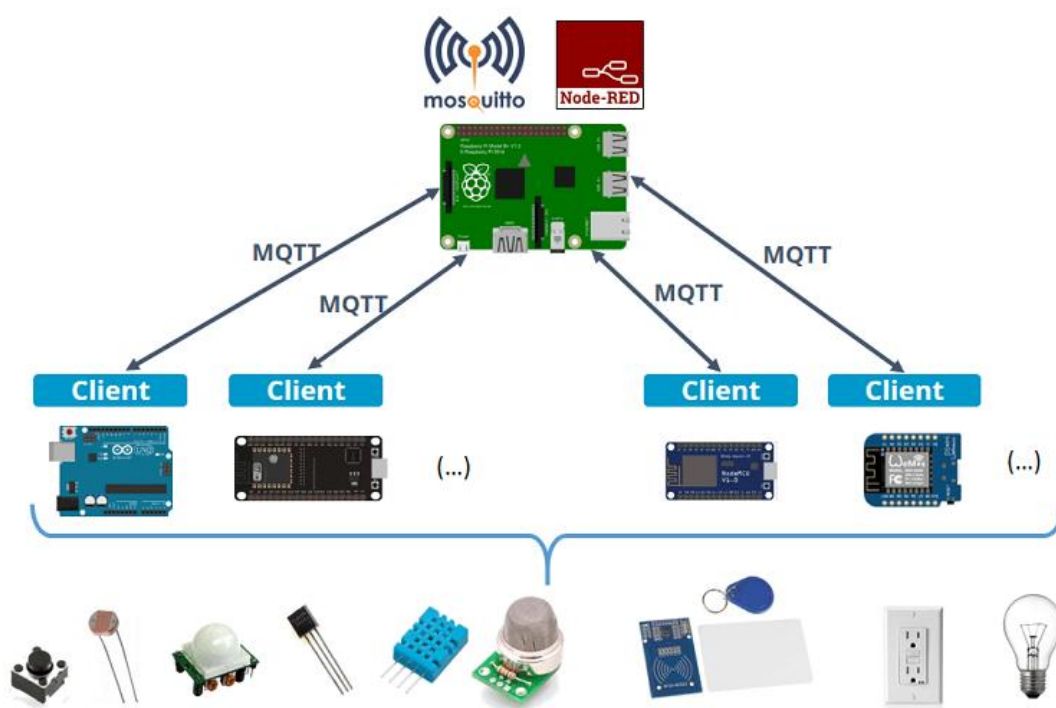


Рисунок 2.1 – Схема роботи MQTT [16]

У системі голосового асистента роль центрального вузла так само може виконувати Raspberry Pi, який після обробки голосової команди буде надсилати повідомлення у відповідну MQTT-тему. Наприклад, після вказівки користувача

«увімкни світло» сервер через брокера буде вносити команду в тему `home/light/set`, а всі компоненти, які підписані на неї виконуватимуть відповідну дію.

Основною перевагою саме цього протоколу для системи є невелике навантаження на мережу. До того ж завдяки свої простоті реалізації додавання нових пристроїв не викликатиме труднощів. Окремо варто відмітити асинхронну обробку повідомлень, що значно прискорює роботи системи, та зручність роботи з мікроконтролерами.

У системі також доцільно використовувати HTTP, який на відміну від MQTT, працює за моделлю `request/response`, тобто клієнт надсилає запит, а сервер повертає відповідь. Такий підхід зручний для передавання файлів, роботи веб-інтерфейсу та реалізації API.

В даній системі роль центрального вузла виконуватиме мікрокомп'ютер з Flask сервером, на який по POST запиту прийматиме WAV-файл з ESP32 та у відповідь надсилати відповідь на питання або результат виконання команди. За допомогою цього ж протоколу можна створити веб-панель для ручного керування пристроями IoT у будинку та простого логування/тестування дієздатності системи [17].

2.5.2 Веб-панель адміністрування IoT

Для зручного керування та тестування системи доцільно передбачити веб-панель адміністрування. Вона може бути реалізована на базі Flask і відкриватися у браузері в межах локальної мережі.

Панель не є обов'язковою для основної роботи голосового асистента, однак вона значно спрощує процес налагодження, перевірки пристроїв і ручного керування системою. Через неї користувач або розробник може перевірити стан підключених модулів, переглянути журнал команд, вручну увімкнути чи вимкнути пристрій, а також перевірити роботу STT і TTS.

Основні можливості:

- перегляд списку підключених пристроїв;
- ручне керування IoT-модулями;

- перегляд журналу голосових команд;
- перевірка стану ESP32-S3;
- тестування розпізнавання мовлення;
- тестування синтезу голосової відповіді;
- налаштування MQTT-тем.

У межах проєкту вона може використовуватися як додатковий інтерфейс керування розумним домом разом із голосовим керуванням.

Узагальнюючи вибрані апаратні та програмні засоби доцільно подати зведену таблицю, яка розміщена в додатку А. Вона відображає основні компоненти системи, їх призначення та роль у загальній архітектурі голосового асистента.

Обраний набір компонентів дозволяє реалізувати модульну систему, у якій кожна частина виконує окрему функцію. ESP32-S3 відповідає за отримання голосу, Raspberry Pi – за обробку, MQTT – за взаємодію з пристроями, а HTTP і веб-панель – за сервісне керування та адміністрування.

РОЗДІЛ 3

ПРОЄКТУВАННЯ, РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ

Система голосового асистента із функціями розумного дому побудована за розподіленою архітектурою. Такий підхід передбачає поділ функцій між окремими апаратними та програмними модулями. Основна ідея полягає в тому, що периферійний пристрій на базі ESP32-S3 відповідає за роботу з аудіосигналом, а центральний вузол на базі Raspberry Pi виконує обробку голосових команд, формування відповіді та керування пристроями розумного дому.

3.1 Розробка структурної архітектури системи

Використання розподіленої архітектури дозволяє ефективно використати можливості кожного компонента. Мікроконтролер є зручним для роботи з мікрофоном, активації за ключовим словом і передавання аудіоданих через Wi-Fi. мікрокомп'ютер, у свою чергу, має достатню продуктивність для запуску серверної частини, модуля розпізнавання мовлення, логіки обробки команд, синтезу мовлення та взаємодії з IoT-пристроями.

Загальна структура системи передбачає послідовну обробку голосової команди: користувач вимовляє ключове слово, після чого ESP32-S3 записує аудіофрагмент і передає його на Raspberry Pi. Серверна частина приймає аудіофайл, виконує розпізнавання мовлення, визначає намір користувача та приймає рішення щодо подальшої дії.

Загальна схема роботи голосового асистента відображає послідовність взаємодії користувача з системою. Основним джерелом вхідних даних є голосова команда, яка надходить через мікрофон INMP441. Він надсилає цифровий аудіосигнал на ESP32-S3 через інтерфейс I2S. Мікроконтролер аналізує аудіопотік, очікує ключове слово та після його виявлення записує короткий аудіофрагмент команди.

Після запису аудіофайл у форматі WAV передається на Raspberry Pi через HTTP-запит. На стороні мікрокомп'ютера працює Flask-сервер, який приймає файл, зберігає його в окрему директорію та передає на модуль розпізнавання мовлення. Для перетворення голосу в текст використовується Vosk. Отриманий результат надходить до модуля обробки команд, де визначається намір користувача (рисунк 3.1).

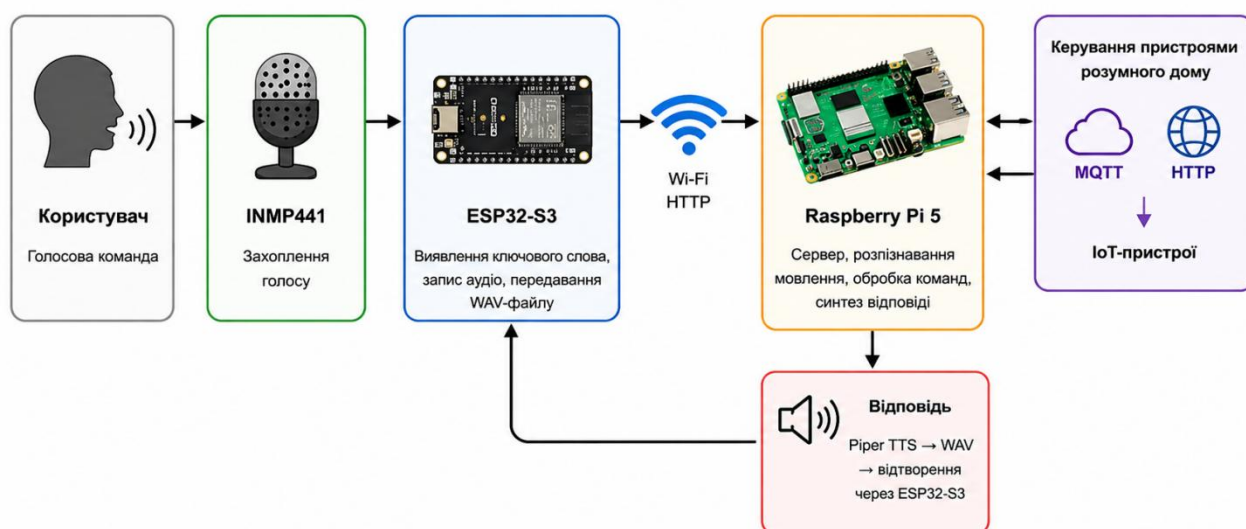


Рисунок 3.1 – Загальна схема роботи голосового асистента

Одним із ключових рішень під час проєктування системи є розподіл функцій між периферією та серверним вузлом. Такий поділ дозволяє уникнути надмірного навантаження і перенести складні обчислення на більш продуктивний пристрій.

ESP32-S3 виконує функції, пов'язані з безпосередньою роботою з аудіосигналом. До них належать зчитування даних із мікрофона, попередня обробка аудіо, виявлення ключового слова, запис голосового фрагмента та передавання WAV-файлу на сервер. Також ESP32-S3 може приймати відповіді від Raspberry Pi та відтворювати його через I2S-підсилювач MAX98357A (рисунк 3.2).

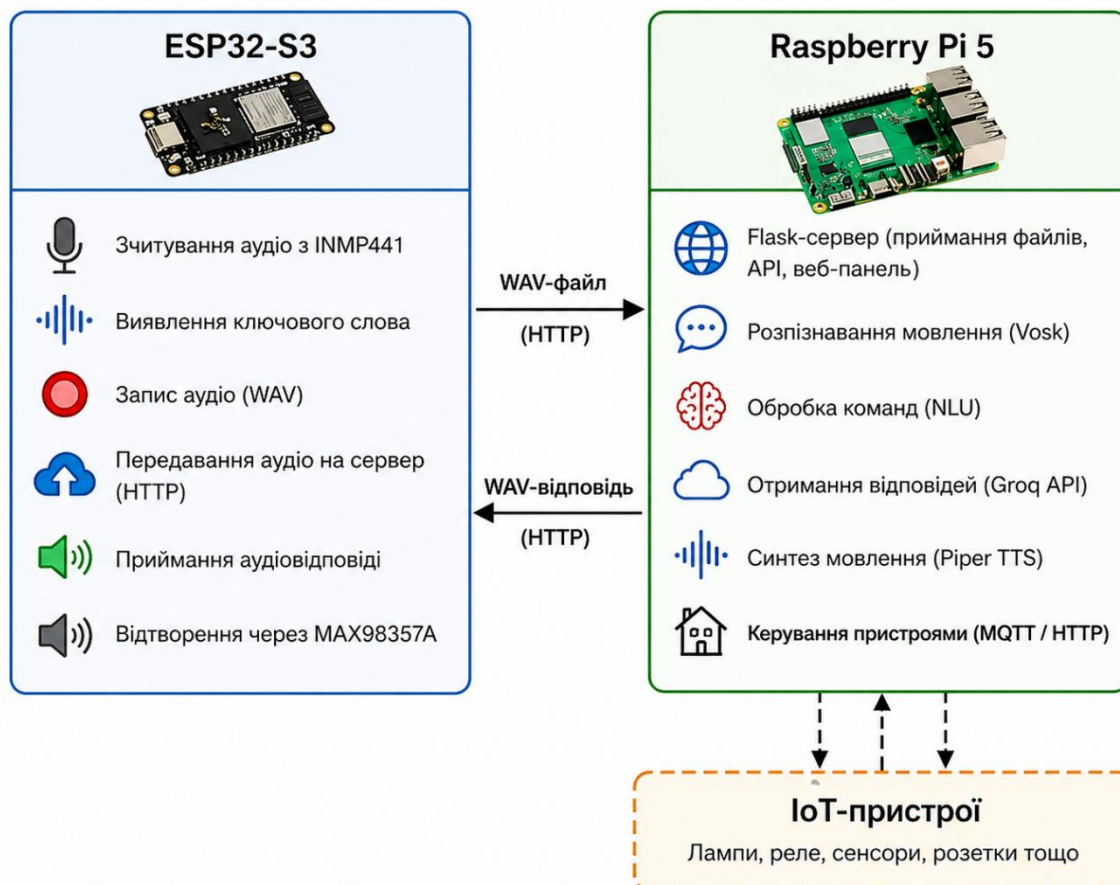


Рисунок 3.2 – Поділ функцій між ESP32-S3 та Raspberry Pi 5

Raspberry Pi 5 виконує роль центрального обчислювального вузла. На ньому розміщується Flask-сервер, який приймає аудіодані від ESP32-S3. Крім цього, Raspberry Pi виконує розпізнавання мовлення, обробку текстової команди, визначення наміру користувача, керування пристроями розумного дому, формування відповіді та синтез мовлення.

Такий розподіл є доцільним, оскільки це підвищує стабільність системи та спрощує її подальше розширення.

Апаратна частина системи складається з кількох основних компонентів: мікроконтролера ESP32-S3-WROOM-1-N16R8, одноплатного комп'ютера Raspberry Pi 5, цифрового мікрофона INMP441, підсилювача MAX98357A, динаміка та IoT-пристроїв. Кожен із цих компонентів виконує окрему функцію в загальній структурі голосового асистента (таблиця 3.1).

Таблиця 3.1 – Основні функції апаратної частини системи

Компонент	Призначення
ESP32-S3-WROOM-1-N16R8	аудіовузол, запис команди, передавання даних
Raspberry Pi 5	серверна обробка, STT, NLU, TTS, IoT-керування
INMP441	цифровий I2S-мікрофон для захоплення голосу
MAX98357A	I2S-підсилювач для відтворення відповіді
Динамік	виведення голосової відповіді
ESP32-C3 / реле / лампа	прикладі IoT-пристроїв для керування
Wi-Fi мережа	передавання аудіо та команд між модулями

Програмна частина системи складається з кількох взаємопов'язаних модулів. Частина програмного забезпечення працює на ESP32-S3, а основна серверна логіка розміщується на Raspberry Pi.

На мікроконтролері реалізовано програмний модуль, який відповідає за підключення до Wi-Fi, роботу з мікрофоном через I2S, виявлення ключового слова, запис аудіофайлу та передавання його на сервер. Також периферійний вузол приймає відповідь від центрального вузла і відтворює її через підсилювач. Таким чином, програмна частина ESP32-S3 забезпечує вхід та вихід аудіо з мережевий обміном.

Серверна частина, реалізована на Flask. Вона містить набір маршрутів для приймання аудіофайлів, перевірки стану системи, перегляду журналу команд, роботи з веб-панеллю та керування пристроями. Після отримання аудіофайлу сервер передає його до модуля розпізнавання мовлення Vosk, який перетворює аудіо в текст.

Результат обробки надходить до NLU-модуля. Його завданням є визначення наміру користувача. Для простих команд розумного дому використовуються локальні правила та ключові слова. Наприклад, фрази на

зразок «увімкни світло» або «вимкни реле» перетворюються на відповідні команди керування. Якщо запит не є командою розумного дому, він може бути переданий до Groq API для формування інформаційної відповіді.

Після виконання команди або отримання результату система повертає відомості для користувача. Цей текст передається до Piper TTS, який генерує WAV-файл із голосовою відповіддю. Далі файл повертається на ESP32-S3 і відтворюється через динамік.

Модульна організація дозволяє змінювати або розширювати окремі компоненти без повної перебудови всієї системи (таблиця 3.2).

Таблиця 3.2 – Програмні модулі системи

Модуль	Місце виконання	Призначення
Wi-Fi-модуль	ESP32-S3	підключення до локальної мережі
I2S audio-модуль	ESP32-S3	робота з INMP441 та MAX98357A
Wake word-модуль	ESP32-S3	активація за ключовим словом
HTTP client	ESP32-S3	передавання WAV-файлу на сервер
Flask-сервер	Raspberry Pi	приймання аудіо, API, веб-панель
Vosk STT	Raspberry Pi	розпізнавання мовлення
NLU-модуль	Raspberry Pi	визначення наміру користувача
Groq API-модуль	Raspberry Pi / хмара	відповіді на складні запитання
Piper TTS	Raspberry Pi	синтез голосової відповіді
MQTT / HTTP IoT-модуль	Raspberry Pi	керування пристроями розумного дому
Журнал команд	Raspberry Pi	збереження історії запитів і результатів

3.2 Проєктування загального алгоритму роботи системи

Після визначення структурної архітектури системи необхідно описати загальний порядок її роботи. Основна логіка голосового асистента полягає в

послідовному проходженні команди від користувача до відповідного результату: виконання дії в системі розумного дому або формування інформаційної відповіді. Такий підхід дозволяє розділити роботу системи на окремі етапи, кожен із яких виконує конкретне завдання.

Взаємодія користувача з голосовим асистентом побудована так, щоб не вимагати додаткових дій, окрім голосового звернення. Користувач вимовляє ключове слово, після чого система переходить із режиму очікування до запису команди. Це дає змогу уникнути постійного передавання аудіопотоку на сервер і зменшити навантаження на мережу та центральний вузол.

На першому етапі ESP32-S3 постійно аналізує короткі аудіофрагменти, отримані з мікрофона INMP441. Якщо ключове слово не виявлено, система залишається в режимі очікування. Після спрацювання активації мікроконтролер починає запис голосової команди у форматі WAV. Тривалість запису може бути фіксованою, наприклад кілька секунд, або визначатися на основі наявності мовлення.

Після завершення запису аудіо надсилається на Raspberry Pi. Сервер приймає файл, виконує розпізнавання мовлення та отримує текстову команду. Далі система визначає, чи потрібно керувати пристроєм розумного дому, чи сформувати звичайну відповідь на запитання. Після виконання відповідної дії користувач отримує голосове підтвердження (рисунк 3.1).

Обробка голосової команди є основним етапом роботи системи. Саме на ньому аудіофайл перетворюється на текст і після на конкретну дію. Для цього використовується кілька послідовних програмних модулів: розпізнавання мовлення, аналіз тексту, визначення наміру та вибір сценарію виконання.

Після отримання WAV-файлу сервер передає його до модуля розпізнавання мовлення. У розроблюваній системі для цього використовується Vosk, який працює локально на Raspberry Pi. Результатом цього етапу є текстовий рядок, наприклад: «увімкни світло» або «яка температура».

Далі текст надходить до NLU-модуля. Його завдання полягає в тому, щоб визначити намір користувача. Наприклад, команда «увімкни світло»

відповідає наміру `light_on`, а команда «вимкни реле» – наміру `relay_off`. Якщо команда належить до заздалегідь визначених дій розумного дому, вона обробляється локально. Це дозволяє виконувати типові вказівки швидко та без звернення до зовнішніх сервісів.

Якщо ж текст не відповідає жодному визначеному правилу, система може передати його до Groq API. Такий механізм використовується для запитань загального характеру, де потрібна відповідь у природній мовній формі. У результаті система підтримує два режими роботи: локальне виконання побутових команд і зовнішню генерацію відповідей для складніших запитів (рисунок 3.3).

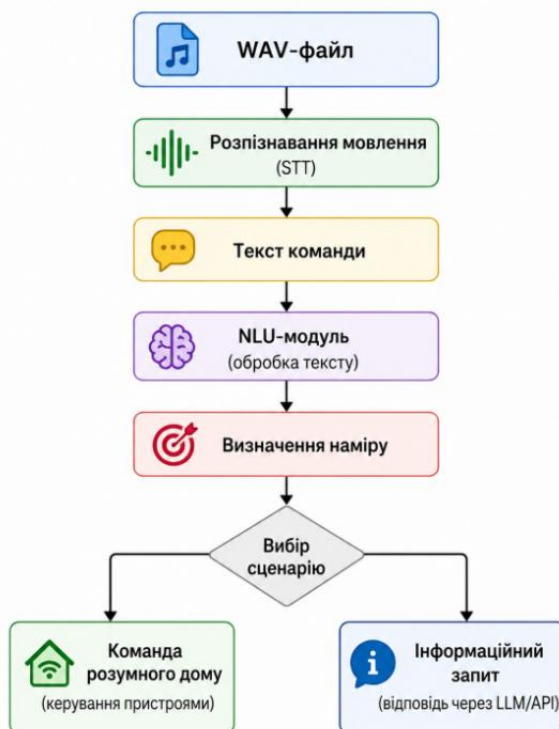


Рисунок 3.3 – Алгоритм обробки голосової команди

Обмін даними між основними компонентами організовано через локальну Wi-Fi мережу. Для передавання аудіофайлу використовується запит, оскільки такий спосіб є простим у реалізації, добре підтримується як на ESP32, так і на Flask-сервері, а також зручний для тестування під час розробки.

Після виявлення ключового слова і запису голосової команди мікроконтролер формує POST-запит і надсилає файл на серверний маршрут Raspberry Pi. У запиті передається тіло файлу, а також службові заголовки, наприклад тип вмісту audio/wav.

Сервер приймає запит, зберігає отриманий файл, виконує його обробку та формує відповідь. Особливістю реалізації є те, що ESP32 після надсилання очікує результат у межах того самого HTTP-з'єднання. Після розпізнавання команди, виконання дії та синтезу мовлення сервер повертає файл із голосовою відповіддю без необхідності окремого запиту на завантаження.

Такий підхід спрощує логіку роботи периферійного вузла. мікроконтролер виконує одну послідовність: надсилає аудіо, очікує відповідь, отримує аудіо і відтворює його через MAX98357A.

Після розпізнавання мовлення система отримує текстовий запит користувача. Оскільки вони можуть мати різне призначення, необхідно розділити їх на дві основні групи: команди керування пристроями розумного дому та інформаційні запитання.

До першої групи належать команди, що мають чітку дію і можуть бути виконані локально. Наприклад: «увімкни світло», «вимкни реле», «яка температура», «увімкни лампу». Такі запити не потребують звернення до великої мовної моделі, оскільки їх можна обробити за допомогою правил, ключових слів і таблиці відповідності команд (таблиця 3.3). Це забезпечує швидку реакцію системи та зберігає працездатність основних функцій навіть без доступу до Інтернету.

Таблиця 3.3 – Типи запитів користувача та спосіб їх обробки

Тип запиту	Приклад	Спосіб обробки	Результат
Команда керування	Увімкни світло	Локальний NLU	MQTT / HTTP-команда
Запит стану	Яка температура	Локальний NLU + датчик	Повернення показника

Продовження таблиці 3.3

Тип запиту	Приклад	Спосіб обробки	Результат
Інформаційне питання	Що таке розумний дім	Groq API	Текстова відповідь
Нерозпізнаний запит	Некоректна або шумна фраза	Повідомлення про помилку	Прохання повторити команду

До другої групи належать запити, для яких немає наперед визначеної локальної дії. Наприклад, користувач може попросити пояснити певне поняття, дати коротку довідку або відповісти на загальне питання. У такому випадку система передає текст запиту до Groq API. Після отримання відповіді текст надходить до модуля синтезу мовлення.

Такий підхід є доцільним для голосового асистента, оскільки критичні функції розумного дому залишаються локальними, а зовнішній LLM використовується тільки як додатковий модуль.

Після виконання команди або отримання інформаційної відповіді система повинна повідомити користувача про результат. Для цього формується короткий текст, який потім перетворюється на аудіофайл за допомогою Piper TTS.

Якщо була виконана команда розумного дому, відповідь має бути лаконічною. Наприклад, після команди «увімкни світло» система може сформулювати фразу «Світло увімкнено». Якщо команда не виконалася, користувач отримує повідомлення про помилку або неможливість виконання дії. Для інформаційних запитів відповідь формується на основі результату, отриманого від Groq API.

Після створення тексту сервер передає його до Piper TTS. Модуль синтезу мовлення генерує файл, який повертається на ESP32-S3 у тілі відповіді. Мікроконтролер приймає аудіодані, перевіряє WAV-заголовок, пропускає службову частину файлу та відтворює PCM-дані через I2S-підсилювач

MAX98357A. У результаті користувач чує голосове підтвердження або відповідь на своє запитання (рисунок 3.4).



Рисунок 3.4 – Алгоритм формування та відтворення голосової відповіді

3.3 Реалізація периферійного модуля на ESP32-S3

Периферійний модуль системи реалізовано на базі мікроконтролера ESP32-S3-WROOM-1-N16R8. У розробленій системі він не виконує повного розпізнавання мовлення, що дозволяє зменшити навантаження на мікроконтролер, але водночас відповідає за задачі реального часу: постійне прослуховування аудіопотоку, роботу з I2S-пристроями та мережевий обмін із сервером (рисунок 3.5).

Загальний цикл його роботи має такий вигляд: після запуску пристрій ініціалізує пам'ять, аудіоінтерфейс і Wi-Fi-з'єднання, після чого переходить у режим очікування ключового слова. Після спрацювання WakeNet мікроконтролер записує п'ятисекундний WAV-файл, надсилає його на сервер через HTTP POST, отримує у файл-відповідь із синтезованим мовленням і відтворює його через I2S-підсилювач MAX98357A.

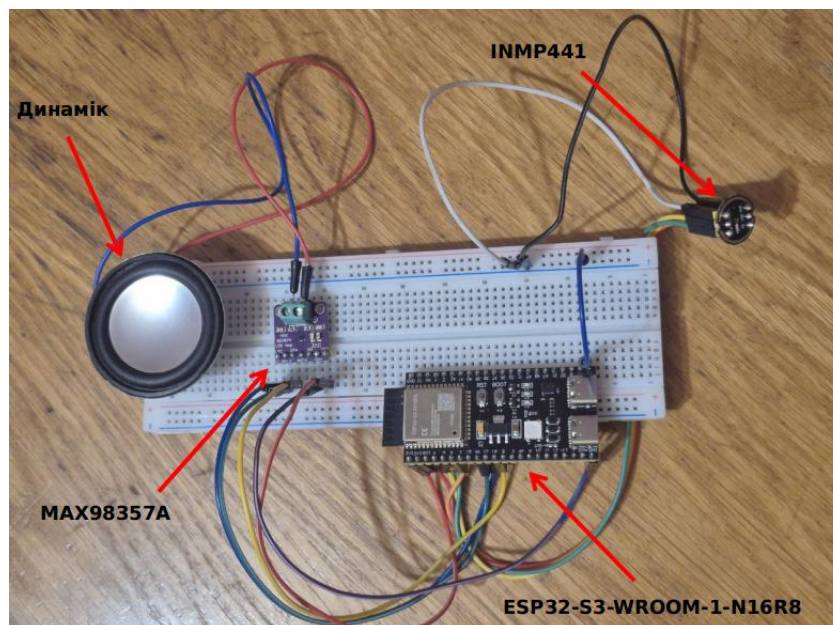


Рисунок 3.5 – Загальна схема периферійного вузла

Для розробки програмного забезпечення периферійного модуля використано середовище PlatformIO з фреймворком ESP-IDF. Такий підхід дає змогу працювати з низькорівневими можливостями ESP32-S3, зокрема з Wi-Fi, HTTP-клієнтом, I2S, FreeRTOS-задачами, NVS-пам'яттю та компонентами голосового розпізнання ESP-SR.

У файлі конфігурації `platformio.ini` задано середовище `esp32-s3-n16r8`, плату `esp32-s3-devkitc-1`, фреймворк `espidf`, швидкість монітора 115200 бод і параметри флеш-пам'яті. Також вказано використання власної таблиці розділів `partitions.csv`, розмір Flash 16 МБ, режим `qio` та тип PSRAM `opi` [18].

Окрему роль відіграють параметри `build_flags`, у яких визначаються дані Wi-Fi-мережі, адреса серверного маршруту Raspberry Pi, а також контакти для мікрофона INMP441 і підсилювача MAX98357A. Це дозволяє змінювати основні параметри підключення без редагування логіки програми. Лістинг файлу конфігурацій наведено у додатку Б.

У файлі `CMakeLists.txt` зареєстровано основний файл `main.cpp` і вказано необхідні компоненти ESP-IDF: `driver`, `esp_http_client`, `esp_wifi`, `esp_event`,

esp_netif та nvs_flash. Це забезпечує доступ до периферії, мережевих функцій і системної пам'яті (лістинг 3.1).

Лістинг 3.1 – Компоненти зареєстровані в CMakeLists.txt

```
idf_component_register(
    SRCS "main.c"
    INCLUDE_DIRS "."
    REQUIRES
    driver
    esp_http_client
    esp_wifi
    esp_event
    esp_netif
    nvs_flash
)
```

кінець лістингу 3.1

Для підтримки ESP-SR і WakeNet у таблиці розділів створено окремий розділ model, який починається з адреси 0x410000 і має розмір 0x600000. Саме в цей розділ записуються моделі розпізнавання ключового слова. Додатково передбачено розділ spiffs, який може використовуватися для зберігання допоміжних файлів (лістинг 3.2).

Лістинг 3.2 – Схема розмітки пам'яті

#	Name,	Type,	SubType,	Offset,	Size
nvs,		data,	nvs,	0x9000,	0x5000
otadata,		data,	ota,	0xe000,	0x2000
app0,		app,	ota_0,	0x10000,	0x200000
app1,		app,	ota_1,	0x210000,	0x200000
model,		data,	0x40,	0x410000,	0x600000
spiffs,		data,	spiffs,	0xA10000,	0x5F0000

кінець лістингу 3.2

Підключення ESP32-S3 до локальної мережі реалізовано в режимі station, тобто мікроконтролер підключається до вже наявної точки доступу Wi-Fi. Це необхідно для передавання аудіофайлу на сервер, який працює в тій самій локальній мережі.

Лістинг файлу `main.cpp` у додатку Б, містить метод ініціалізації та налаштування мережевого підключення реалізований функціями `wifi_event_handler` та `wifi_init_sta`. У програмі для роботи з Wi-Fi використано механізм подій ESP-IDF. Створюється група подій FreeRTOS, у якій фіксується успішне підключення або помилка після перевищення кількості спроб. У разі розриву з'єднання пристрій намагається повторно підключитися до мережі. Максимальна кількість повторних спроб становить 10.

Після налаштування Wi-Fi встановлюється режим `WIFI_MODE_STA`, передаються SSID і пароль мережі, після чого запускається підключення. Для стабільнішого мережевого обміну енергозбереження Wi-Fi вимикається через `esp_wifi_set_ps(WIFI_PS_NONE)`. Це важливо, оскільки передавання файлів потребує стабільного з'єднання без небажаних затримок.

Для роботи з аудіосигналом використовується інтерфейс I2S. Він застосовується у двох режимах: приймання аудіо з мікрофона INMP441 та відтворення звуку через підсилювач MAX98357A. У реалізації використовується один I2S-порт, який залежно від етапу роботи перемикається між режимом приймання та режимом передавання.

Мікрофон INMP441 I2S налаштовано в режимі `I2S_MODE_RX`, тобто отримання звуку. Частота дискретизації становить 16 кГц, мікрофон передає 32-бітні семпли, які перетворюються у 16-бітний формат (лістинг 3.3). Для цього після читання з I2S виконується зсув значення, що дозволяє отримати придатний для подальшої обробки PCM-сигнал.

Лістинг 3.3 – Конфігурація мікрофона INMP441

```
i2s_config_t cfg = {
    .mode = (i2s_mode_t)(I2S_MODE_MASTER | I2S_MODE_RX),
    .sample_rate = SAMPLE_RATE,
    .bits_per_sample = I2S_BITS_PER_SAMPLE_32BIT,
    .channel_format = I2S_CHANNEL_FMT_ONLY_LEFT,
    .communication_format = I2S_COMM_FORMAT_STAND_I2S,
    .dma_buf_count = 8,
    .dma_buf_len = 256,
};
```

кінець лістингу 3.3

Для відтворення голосової відповіді I2S перемикається в режим I2S_MODE_TX. У цьому режимі ESP32-S3 передає 16-бітні аудіодані на MAX98357A. Підсилювач приймає цифровий сигнал і відтворює його через динамік.

У таблицях 3.4 та 3.5 наведено особливості підключення для мікрофона INMP441 та підсилювача MAX98357A відповідно. Порти GPIO можуть відрізнятися проте важливо змінити їх у програмі.

Таблиця 3.4 – Підключення INMP441 до ESP32

INMP441	GPIO ESP32-S3	Призначення
WS	GPIO 4	вибір каналу / синхронізація
SCK	GPIO 5	тактовий сигнал
SD	GPIO 6	цифрові аудіодані
VCC	3.3 V	живлення
GND	GND	спільна земля
L/R	GND	стерео підтримка

Таблиця 3.5 – Підключення MAX98357A до ESP32

MAX98357A	GPIO ESP32-S3	Призначення
BCLK	GPIO 16	тактовий сигнал
LRC / WS	GPIO 17	синхронізація каналу
DIN	GPIO 18	аудіодані
VIN	5 V	живлення підсилювача
GND	GND	спільна земля

Особливістю реалізації є перемикання одного I2S-порту між мікрофоном і динаміком. Перед відтворенням відповіді драйвер I2S зупиняється та деінсталюється, після чого повторно ініціалізується вже в режимі передавання. Після завершення відтворення система повертає I2S у режим приймання з

мікрофона. Це перемикання реалізовано двома функціями `audio_switch_to_speaker` та `audio_switch_to_mic` вміст яких наведено у додатку Б.

Активація голосового асистента реалізована за допомогою стеку ESP-SR і WakeNet. У проєкті використовується модель `wn9_alexa`, яка завантажується з окремого розділу пам'яті `model`. Після ініціалізації у функції `sr_init` створюється AFE-контекст, що відповідає за приймання аудіофреймів, попередню обробку та перевірку на наявність ключового слова. Додатково в ній же перевіряється наявність моделі у пам'яті.

Під час роботи основної задачі ESP32-S3 постійно читає аудіофрагменти з мікрофона через цикл `while`. Отримані семпли передаються в AFE через функцію `feed`. Після цього викликається `fetch`, яка повертає результат обробки. Якщо стан `wakeup_state` дорівнює `WAKENET_DETECTED`, система вважає, що ключове слово було вимовлено.

Для роботи AFE створюються буфери `mono` і `feed`. Перший містить одноканальний аудіосигнал з мікрофона, другий формується з урахуванням кількості каналів, потрібної для AFE. Буфери виділяються в PSRAM, що дозволяє зменшити навантаження на внутрішню SRAM мікроконтролера.

Після виявлення ключового слова ESP32-S3 записує голосову команду у форматі WAV. У поточній реалізації тривалість запису становить 5 секунд, а частота дискретизації – 16 кГц. Аудіо записується у форматі `mono`, 16-bit PCM.

Спочатку в PSRAM виділяється буфер для PCM-даних, обсяг яких розраховується добутком швидкості запису, часу та розміром одиниці пам'яті. У нашому випадку 16000семплів/с, які видає мікрофон, множимо на 5 секунд та 2 байта, в результаті маємо 160000 байт для запису команди (лістинг 3.4).

Лістинг 3.4 – Розрахунок пам'яті для запису

```
...
const uint32_t total_samples = SAMPLE_RATE * RECORD_SECONDS;
const uint32_t pcm_size = total_samples * sizeof(int16_t);
const uint32_t wav_size = sizeof(wav_header_t) + pcm_size;
...
```

кінець лістингу 3.4

Після цього з мікрофона послідовно зчитуються блоки по 512 семплів. Кожен блок додається до загального буфера. Коли запис завершено, створюється другий буфер для файлу. На початок цього буфера записується WAV-заголовок, після якого копіюються PCM-дані. Тобто спочатку виділяється пам'ять і дані пишуться до тих пір поки вона не переповниться, отже якщо ми хочемо змінити тривалість запису треба буде перерозрахувати обсяг даних для того часу.

У програмі окремо реалізовано структуру заголовка, яка містить службові поля RIFF, WAVE, fmt, data, частоту дискретизації, кількість каналів, розрядність і розмір аудіоданих. Завдяки цьому сервер отримує не просто сирий PCM-потік, а повноцінний WAV-файл, який можна зберігати, перевіряти та передавати в модуль розпізнавання мовлення. Функція яка відповідає за запис в програмі – `record_wav_to_memory` (додаток Б).

Після формування файлу ESP32-S3 надсилає його на Raspberry Pi за допомогою HTTP POST-запиту. Адреса сервера задається в конфігурації проєкту як `UPLOAD_URL`.

Передавання реалізовано через компонент `esp_http_client`. У заголовках запиту вказується тип даних `audio/wav`, а також Асепт: `audio/wav`, що означає очікування аудіофайлу у відповіді. Додатково передається заголовок, за яким сервер може ідентифікувати периферійний вузол.

WAV-файл передається частинами по 2048 байт, це дозволяє не створювати додаткових проміжних буферів і стабільніше працювати з обмеженою пам'яттю мікроконтролера. Після завершення передавання ESP32 очікує відповідь від сервера.

Особливістю реалізації є те, що сервер повертає не JSON із посиланням на аудіо, а безпосередньо файл, тому мікроконтролер може одразу зберегти отримані байти в пам'ять і передати їх у модуль відтворення.

Надсилання та отримання аудіофайлу реалізовано функцією `http_upload_wav_and_get_reply` в ній відкривається з'єднання та спочатку через

один цикл зчитує і передає файл і команду, після надання сервером відповіді перевіряє її та через цикл записує в пам'ять мікроконтролера (додаток Б).

Після отримання WAV-відповіді від сервера ESP32-S3 переходить до етапу відтворення. Спочатку програма переглядає наявність службових міток RIFF, WAVE, fmt і data, а також перевіряє формат аудіо, кількість каналів і розрядність. Це потрібно для того, щоб не передати в I2S випадкові або некоректні дані.

Після перевірки заголовка визначається початок PCM-даних. Оскільки службовий WAV-заголовок займає 44 байти, відтворення починається після нього. Перед програванням система перемикає I2S-порт у режим передавання і налаштовує його на частоту, вказану у аудіо.

Далі дані передаються на MAX98357A блоками по 2048 байт за допомогою `i2s_write`. Підсилювач приймає цифровий аудіопотік, перетворює його та відтворює через підключений динамік. Після завершення відтворення буфер I2S очищується, а аудіоінтерфейс знову перемикається в режим роботи з мікрофоном та очікування ключового слова. Всі завдання від перевірки файлу до програвання реалізовано в функції `play_wav_from_memory` (додаток Б).

3.4 Реалізація серверної частини на Raspberry Pi

Серверна частина системи реалізована на базі одноплатного комп'ютера Raspberry Pi 5. Саме цей вузол виконує основні обчислювальні операції, які є занадто ресурсомісткими для мікроконтролера.

У ролі серверного фреймворка використано Flask, оскільки він дає змогу швидко створити локальний HTTP API для обміну даними між ESP32-S3, панеллю та модулями керування IoT-пристроями. Сервер працює в локальній мережі та приймає аудіозаписи за маршрутом `/audio`. Після обробки запиту Raspberry Pi повертає на мікроконтролер готовий WAV-файл із голосовою відповіддю.

Серверну частину організовано у вигляді модульного Flask-застосунку (рисунок 3.6). Такий підхід дає змогу розділити різні функціональні частини системи на окремі файли: маршрути API, конфігурацію, обробку аудіо, модуль визначення наміру, TTS, IoT-керування та логування.

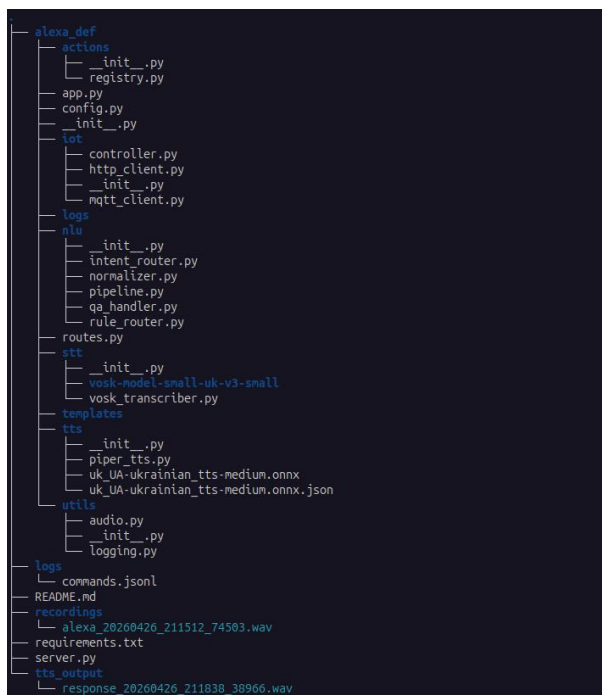


Рисунок 3.6 – Структура Flask-сервера голосового асистента

Основним файлом запуску є `server.py`. У ньому створюється застосунок, виводиться інформація про доступність основних модулів і запускається сервер. Під час старту відображається стан Vosk STT, Piper TTS, директорія для збереження записів і адреса веб-панелі адміністрування (рисунок 3.7).

```
(.venv) rapi@raspberrypi:~/Desktop/pi_server_v1 $ python3 server.py
=====
Atlas Voice Server (ESP32-S3 → Raspberry Pi)
Vosk STT : YES (./alexa_def/stt/vosk-model-small-uk-v3-small)
Piper TTS: YES
Recordings: /home/rapi/Desktop/pi_server_v1/recordings/
Web panel : http://0.0.0.0:5000/admin
=====
* Serving Flask app 'alexa_def.app'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://10.198.114.131:5000
Press CTRL+C to quit
```

Рисунок 3.7 – Вивід інформації при запуску сервера

Створення Flask-застосунку винесено у файл `app.py`. У ньому викликається функція `create_app()`, створюються необхідні директорії для записів, TTS-файлів і логів, а також підключається основний Blueprint із маршрутами.

Конфігурація зберігається у файлі `config.py`. У ньому визначаються шляхи до директорій, параметри аудіо, шлях до Vosk-моделі, налаштування Groq API, Piper TTS, MQTT, HTTP-керування та список IoT-пристроїв. Така структура дозволяє змінювати параметри системи через змінні середовища без редагування основної логіки програми.

Основним маршрутом взаємодії Raspberry Pi та периферійних модулів є `/audio`, через нього відбувається обмін файлами. До допоміжних належать всі інші маршрути, які є частиною веб-панелі наприклад: `/admin`, `/nodes`, `/health`, `/status`, `/devices`, `/log`. Для повноцінної роботи системи достатньо лише основного маршруту, всі інші для аналізу та тестування.

Після надходження аудіофайлу на сервер виконується перевірка його коректності. Насамперед контролюється розмір файлу, наявність заголовка, формат аудіо, кількість каналів, частота дискретизації та розрядність. Також обчислюється рівень сигналу, щоб визначити, чи запис не є надто тихим або порожнім. Якщо файл не відповідає вимогам, сервер припиняє подальшу обробку та повертає повідомлення про помилку.

Якщо аудіофайл успішно проходить перевірку, він зберігається на Raspberry Pi у спеціальній директорії для записів. Для кожного файлу формується унікальна назва з датою та часом отримання, що дозволяє зручно вести історію голосових команд. Збереження записів також спрощує налагодження системи, оскільки дає змогу повторно прослухати аудіо та перевірити якість розпізнавання.

Після збереження WAV-файл передається до модуля розпізнавання мовлення Vosk. Він працює локально на Raspberry Pi та перетворює голосову команду користувача в текст після чого використовується для визначення наміру користувача.

Розпізнавши аудіо сервер отримує текст команди. Далі він передається в NLU-модуль, який визначає, що саме хотів зробити користувач. Наприклад, фраза «увімкни світло» визначається як команда вмикання освітлення, а «вимкни реле» – як команда вимикання реле.

Якщо команда стосується розумного дому, сервер одразу передає її в модуль керування пристроями. Якщо ж це звичайне запитання, яке не пов'язане з керуванням, запит може бути переданий до Groq API для формування відповіді (рисунок 3.8).

```
[RECV] 160,044 B | 16000 Hz | 1ch | 80,000 samples | 5.00s | RMS=3807.0
[SAVE] /home/rapi/Desktop/pi_server_v1/recordings/alexa_20260502_222400_42599.wav
[STT] 'хто такий комп'ютерний інженер'
[NLU] source=groq normalized='хто такий комп'ютерний інженер' intent=qa_general conf=1.00 slots={'room': None, 'device': None, 'value': None}
[QA] Asking: 'хто такий комп'ютерний інженер'
[QA] Answer: 'Комп'ютерний інженер - це фахівець, який займається розробкою, проектуванням та підтримкою комп'ютерної техніки, програмного забезпечення та систем. Він поєднує знання з електроніки, програмування та інформатики для створення інноваційних рішень у сфері комп'ютерних технологій. Вони можуть спеціалізуватися на певних галузях, таких як розробка апаратного забезпечення, програмного забезпечення чи мережевих систем.'
[TTS] text='Комп'ютерний інженер - це фахівець, який займається розробкою, проектуванням та підтримкою комп'ютерної техніки, програмного забезпечення та систем. Він поєднує знання з електроніки, програмування та інформатики для створення інноваційних рішень у сфері комп'ютерних технологій. Вони можуть спеціалізуватися на певних галузях, таких як розробка апаратного забезпечення, програмного забезпечення чи мережевих систем.'
[TTS] generated: /home/rapi/Desktop/pi_server_v1/tts_output/response_20260502_222402_34536.wav
[RESP] returning WAV directly: /home/rapi/Desktop/pi_server_v1/tts_output/response_20260502_222402_34536.wav
[RESP] sending WAV bytes: /home/rapi/Desktop/pi_server_v1/tts_output/response_20260502_222402_34536.wav | 1080876 bytes
192.168.50.96 - - [02/May/2026 22:24:07] "POST /audio HTTP/1.1" 200 -
```

Рисунок 3.8 – Логування інформаційної відповіді сервера

Сервер створює коротку текстову відповідь для користувача, наприклад: «Світло увімкнено» (рисунок 3.9) або «Команду не розпізнано» (рисунок 3.10). Цей текст передається в Piper TTS, який генерує WAV-файл. Готовий аудіофайл повертається на ESP32-S3 і відтворюється через динамік.

```
[RECV] 160,044 B | 16000 Hz | 1ch | 80,000 samples | 5.00s | RMS=1040.2
[SAVE] /home/rapi/Desktop/pi_server_v1/recordings/alexa_20260502_221503_81169.wav
[STT] 'світло'
[NLU] source=groq normalized='світло' intent=light_on conf=1.00 slots={'room': None, 'device': None, 'value': None}
[TTS] text='Світло увімкнено.'
[TTS] generated: /home/rapi/Desktop/pi_server_v1/tts_output/response_20260502_221505_43589.wav
[RESP] returning WAV directly: /home/rapi/Desktop/pi_server_v1/tts_output/response_20260502_221505_43589.wav
[RESP] sending WAV bytes: /home/rapi/Desktop/pi_server_v1/tts_output/response_20260502_221505_43589.wav | 63532 bytes
192.168.50.96 - - [02/May/2026 22:15:07] "POST /audio HTTP/1.1" 200 -
```

Рисунок 3.9 – Логування вдалої обробки команди розумного дому

```
[RECV] 160,044 B | 16000 Hz | 1ch | 80,000 samples | 5.00s | RMS=1308.8
[SAVE] /home/rapi/Desktop/pi_server_v1/recordings/alexa_20260502_221159_50497.w
av
[TTS] text='Не вдалося розпізнати команду.'
[TTS] generated: /home/rapi/Desktop/pi_server_v1/tts_output/response_20260502_2
21201_16360.wav
[RESP] returning WAV directly: /home/rapi/Desktop/pi_server_v1/tts_output/respo
nse_20260502_221201_16360.wav
[RESP] sending WAV bytes: /home/rapi/Desktop/pi_server_v1/tts_output/response_2
0260502_221201_16360.wav | 87084 bytes
192.168.50.96 - - [02/May/2026 22:12:03] "POST /audio HTTP/1.1" 200 -
```

Рисунок 3.10 – Логування сервера при невдалому розпізнанні команди

Весь цикл обробки аудіо та формування відповіді реалізований функцією `_handle_audio`, лістинг якої наведено у додатку В.

3.5 Реалізація керування пристроями розумного дому

Після розпізнавання голосової команди система повинна не лише сформувати відповідь, а й виконати дію над пристроєм розумного дому. Для цього в серверній частині передбачено окремий модуль керування пристроями. Він приймає визначений намір користувача і перетворює його на конкретну команду для пристрою.

У проєкті розглянуто два способи керування: через HTTP та через MQTT. Перший використовується для прямого звернення до пристрою або тестового API, а другий – для гнучкішої взаємодії через брокер повідомлень. Це дозволяє показати два варіанти інтеграції голосового асистента з пристроями.

Модуль керування пристроями виконує роль проміжного рівня між NLU-модулем і фізичними пристроями. Після того як система визначила намір користувача, наприклад `light_on` або `get_temp`, IoT-модуль вибирає відповідний пристрій і спосіб передавання команди.

У структурі модуля можна виділити кілька основних частин: список доступних пристроїв, обробник команд, HTTP-клієнт, MQTT-клієнт і журнал виконаних дій. Завдяки цьому можна додавати нові пристрої без зміни основної логіки голосового асистента (рисунок 3.11).

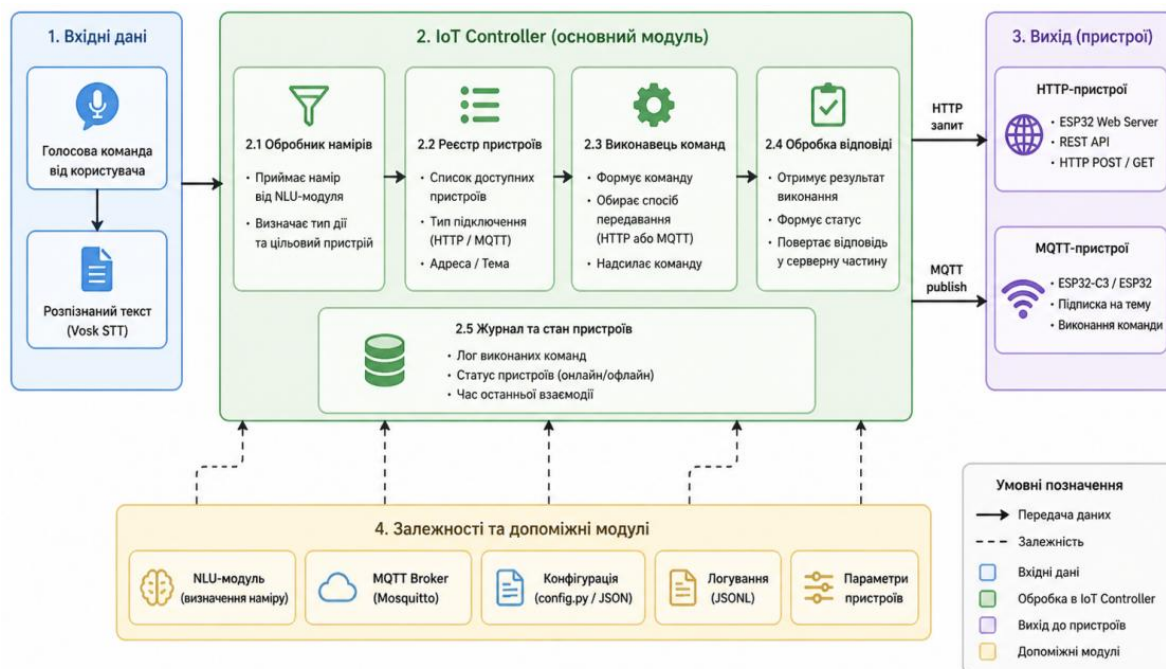


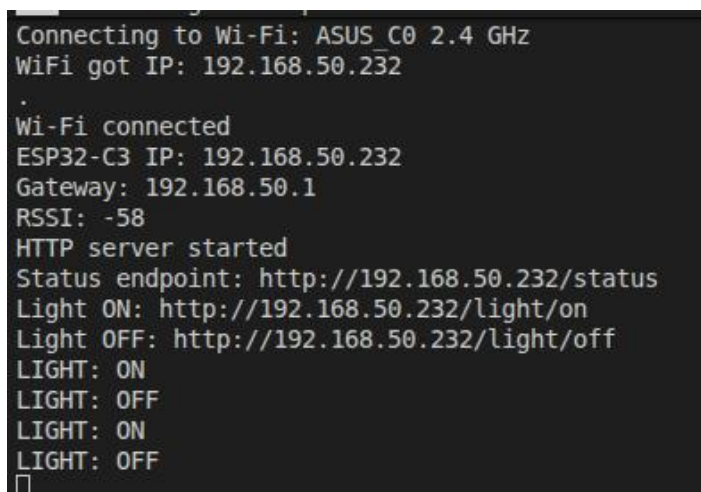
Рисунок 3.11 – Структура модуля керування

Керування через HTTP реалізовано за принципом прямого запиту від Raspberry Pi до IoT-пристрою. Після розпізнавання команди, наприклад «увімкни світло», сервер визначає намір `light_on` і надсилає HTTP-запит на відповідний маршрут пристрою `http://192.168.50.232/light/on` (рисунок 3.12).

```
LOG (VoskAPI:ReadDataFiles():model.cc:282) Loading HCL and G from ./alexa_def/stt/vosk-model-small-uk-v3-small/graph/HCLr.fst ./alexa_def/stt/vosk-model-small-uk-v3-small/graph/Gr.fst
LOG (VoskAPI:ReadDataFiles():model.cc:297) Loading words from ./alexa_def/stt/vosk-model-small-uk-v3-small/graph/words.txt
LOG (VoskAPI:ReadDataFiles():model.cc:308) Loading winfo ./alexa_def/stt/vosk-model-small-uk-v3-small/graph/phones/word_boundary.int
[STT] Ready.
[STT] 'увімкнув світло'
[IOT] device=light state=on mode=http
[HTTP-IOT] http://192.168.50.232/light/on -> 200
[NLU] source=groq normalized='увімкнув світло' intent=light_on conf=1.00 slots={'room': None, 'device': None, 'value': None}
[TTS] text='Світло увімкнено.'
[TTS] generated: /home/rapi/Desktop/pi_server_v1/tts_output/response_20260503_002837_87290.wav
[RESP] returning WAV directly: /home/rapi/Desktop/pi_server_v1/tts_output/response_20260503_002837_87290.wav
[RESP] sending WAV bytes: /home/rapi/Desktop/pi_server_v1/tts_output/response_20260503_002837_87290.wav | 71724 bytes
192.168.50.96 - - [03/May/2026 00:28:40] "POST /audio HTTP/1.1" 200 -
```

Рисунок 3.12 – Логи сервера при команді увімкнення світла через HTTP

На стороні ESP32-C3 працює невеликий HTTP-сервер, який приймає цей запит і змінює стан GPIO, до якого підключено світлодіод. Такий спосіб є простим у реалізації та зручним для тестування, оскільки команду можна перевірити не лише через голосового асистента, а й вручну за допомогою браузера або команди curl (рисунок 3.13).



```
Connecting to Wi-Fi: ASUS C0 2.4 GHz
WiFi got IP: 192.168.50.232
.
Wi-Fi connected
ESP32-C3 IP: 192.168.50.232
Gateway: 192.168.50.1
RSSI: -58
HTTP server started
Status endpoint: http://192.168.50.232/status
Light ON: http://192.168.50.232/light/on
Light OFF: http://192.168.50.232/light/off
LIGHT: ON
LIGHT: OFF
LIGHT: ON
LIGHT: OFF
```

Рисунок 3.13 – Логи IoT при команді перемикання світла через HTTP

Роботу HTTP-керування підтверджують логи серверної частини, наведені на рисунку 3.12. У них видно, що після розпізнавання голосової команди сервер визначає намір увімкнення світла, виконує звернення до пристрою та формує голосову відповідь для користувача. На рисунку 3.13 показано логування самого пристрою, де зафіксовано отримання HTTP-запиту та зміну стану світла.

Керування через MQTT реалізовано за моделлю publish/subscribe. У цьому випадку Raspberry Pi не звертається до пристрою напряму, а публікує повідомлення в MQTT-тему. Наприклад, для вмикання світла використовується тема home/light/set, а як повідомлення передається значення ON (рисунок 3.14) [19].

```

LOG (VoskAPI:ReadDataFiles():model.cc:297) Loading words from ./alexa_def/stt/vosk-model-small-uk-v3-small/graph/words.txt
LOG (VoskAPI:ReadDataFiles():model.cc:308) Loading winfo ./alexa_def/stt/vosk-model-small-uk-v3-small/graph/phones/word_boundary.int
[STT] Ready.
[STT] 'увімкнув світло'
[IOT] device=light state=on mode=mqtt
[MQTT] Connected to localhost:1883
[MQTT-IOT] topic=home/light/set payload=ON
[NLU] source=groq normalized='увімкнув світло' intent=light_on conf=1.00 slots={'room': None, 'device': None, 'value': None}
[TTS] text='Світло увімкнено.'
[TTS] generated: /home/rapi/Desktop/pi_server_v1/tts_output/response_20260503_003215_26539.wav
[RESP] returning WAV directly: /home/rapi/Desktop/pi_server_v1/tts_output/response_20260503_003215_26539.wav
[RESP] sending WAV bytes: /home/rapi/Desktop/pi_server_v1/tts_output/response_20260503_003215_26539.wav | 65068 bytes
192.168.50.96 - - [03/May/2026 00:32:17] "POST /audio HTTP/1.1" 200 -

```

Рисунок 3.14 – Логи сервера при команді увімкнення світла через MQTT

IoT-пристрій на базі ESP32-C3 підписується на цю тему та після отримання повідомлення виконує відповідну дію. Такий підхід є зручнішим для масштабування, оскільки до однієї системи можна додавати кілька пристроїв, які реагуватимуть на власні теми (рисунок 3.15).

Роботу MQTT-керування показано на рисунках 3.14 та 3.15. На першому наведено логи сервера, де видно публікацію команди на вмикання світла. На другому подано логи IoT-пристрою, який отримує повідомлення з брокера та змінює стан виходу [20].

Таким чином, у системі підтримуються два варіанти керування пристроями. HTTP доцільно використовувати для простих пристроїв і прямого керування, а MQTT – для більш гнучкої взаємодії між кількома IoT-вузлами. У межах проекту обидва способи були перевірені на прикладі команди перемикавання світла.

```

WiFi got IP: 192.168.50.232
.
Wi-Fi connected
ESP32-C3 IP: 192.168.50.232
Gateway: 192.168.50.1
RSSI: -54
MQTT configured
Connecting to MQTT broker: 192.168.50.9:1883
MQTT connected
Subscribed to: home/light/set
MQTT topic: home/light/set
MQTT message: ON
LIGHT: ON

```

Рисунок 3.15 – Логи IoT при команді перемикавання світла через MQTT

Для зручності перевірки роботи системи реалізовано веб-панель адміністрування. Вона відкривається у браузері в межах локальної мережі та дозволяє контролювати основні елементи голосового асистента без необхідності щоразу використовувати голосові команди.

У веб-панелі можна переглядати список налаштованих IoT-пристроїв, їхні назви та ідентифікатори. Для виконавчих пристроїв передбачені кнопки ручного керування, наприклад ON та OFF, за допомогою яких можна увімкнути або вимкнути світло чи реле. Це дає змогу окремо перевірити роботу HTTP або MQTT-керування.

Також у панелі відображається журнал останніх команд. У ньому можна побачити розпізнаний текст, визначений намір, відповідь системи та результат виконання дії. Це спрощує налагодження, оскільки дозволяє швидко зрозуміти, на якому етапі виникла проблема (рисуюнок 3.16).

Atlas Voice Server

Панель адміністрування для тестування IoT-команд, журналу та стану сервера.

Пристрої

Світло light

ON OFF

Реле relay

ON OFF

Останні команди

Час	Текст	Намір	Відповідь
2026-05-03T00:33:29	вимкнути світло	light_off	Світло вимкнено.
2026-05-03T00:32:17	увімкнути світло	light_on	Світло увімкнено.
2026-05-03T00:29:47	вимкнути світло	light_off	Світло вимкнено.
2026-05-03T00:28:40	увімкнути світло	light_on	Світло увімкнено.
2026-05-03T00:19:37	увімкнути світло	light_on	Світло увімкнено.
2026-05-03T00:18:31	увімкнути світло	light_on	Світло увімкнено.
2026-05-03T00:05:40	вимкнути світло	light_on	Світло увімкнено.
2026-05-03T00:04:54	увімкнути світло	light_on	Світло увімкнено.
2026-05-03T00:03:22	увімкнути світло	light_on	Світло увімкнено.
2026-05-03T00:02:03	увімкнути світло	light_on	Світло увімкнено.
2026-05-02T23:57:41	увімкнути світло	light_on	Світло увімкнено.
2026-05-02T23:55:28	увімкнені світло	light_on	Світло увімкнено.
2026-05-02T23:55:16	вимкнути світло	light_off	Світло вимкнено.
2026-05-02T23:46:05	увімкнути світло	light_on	Світло увімкнено.
2026-05-02T23:33:43	увімкнути світло	light_on	Світло увімкнено.
2026-05-02T23:08:29			
2026-05-02T23:08:28			
2026-05-02T23:08:25			
2026-05-02T23:08:24			
2026-05-02T22:24:07	хто такий комп'ютерний інженер	qa_general	Комп'ютерний інженер - це фахівець, який займається розробкою, проєктуванням та підтримкою комп'ютерної техніки, програмного забезпечення та систем. Він поєднує знання з електроніки, програмування та інформатики для створення інноваційних рішень у сфері комп'ютерних технологій. Вони можуть спеціалізуватися на певних галузях, таких як розробка апаратного забезпечення, програмного забезпечення чи мережевих систем.

[/health](#) · [/logs](#) · [/devices](#) · [/nodes](#)

Рисуюнок 3.16 – Загальний вигляд адмін-панелі

Окремо у веб-панелі розміщено посилання на службові маршрути. Маршрут `/health` використовується для перевірки загального стану сервера та доступності основних модулів. Шлях `/logs` дає змогу переглянути останні оброблені команди, їхній текст, визначений намір і відповідь системи. Через `/devices` можна отримати список налаштованих IoT-пристроїв, доступних для керування. Маршрут `/nodes` показує підключені вузли системи, наприклад ESP32-S3, який надсилає аудіофайли на сервер. Це дозволяє швидко перевіряти стан системи та спрощує процес налагодження [21].

Отже, веб-панель використовується як допоміжний інструмент для тестування, ручного керування та моніторингу роботи системи. Вона не є обов'язковою для основного сценарію голосового керування, але значно спрощує перевірку працездатності окремих модулів.

3.6 Тестування системи та оцінка результатів

Після реалізації основних модулів було проведено тестування системи голосового асистента. Перевірка виконувалася поетапно під час розробки та налагодження окремих частин: периферійного модуля ESP32-S3, серверної частини на Raspberry Pi, розпізнавання мовлення, синтезу відповіді та керування IoT-пристроями.

Основним способом перевірки були логи, які виводилися в термінал ESP32, Flask-сервера та IoT-пристрою. Завдяки цьому можна було простежити проходження команди через усі етапи: запис аудіо, передавання WAV-файлу, розпізнавання мовлення, визначення наміру, виконання дії та формування голосової відповіді.

Тестування основних функцій системи. Перевірка роботи апаратної частини починалася з тестування підключення ESP32-S3 до Wi-Fi, роботи мікрофона INMP441 та відтворення звуку через MAX98357A. Коректність запису аудіо перевірялася по логуваннях сервера, у яких відображалися параметри отриманого WAV-файлу: розмір, частота дискретизації, кількість

каналів, тривалість запису та рівень RMS. Якщо сервер отримував файл розміром близько 160 КБ, з частотою 16 кГц, одним каналом і тривалістю близько 5 секунд, запис вважався коректним.

Серверна частина перевірялася через службові Flask-маршрути `/health`, `/logs`, `/devices` та `/nodes`.

Розпізнавання мовлення спостерігалось за виводом Vosk у логах сервера. Після отримання аудіофайлу сервер відображав розпізнаний текст, наприклад «увімкни світло» або «вимкни світло». Далі перевірялася робота NLU-модуля: система мала визначити відповідний намір, наприклад `light_on` або `light_off`. Якщо розпізнаний текст містив незначні помилки, але намір визначався правильно, команда вважалася успішно обробленою.

Синтез мовлення перевірявся через Piper TTS. Після формування текстової відповіді сервер створював WAV-файл і повертав його на ESP32-S3. У логах це підтверджувалося повідомленнями про створення TTS-файлу та надсилання WAV-байтів у HTTP-відповіді. На стороні ESP32-S3 перевірялося, чи отриманий файл має коректний заголовок і чи відтворюється через динамік.

Керування пристроями розумного дому тестувалося окремо для HTTP та MQTT. Для HTTP-керування спочатку вручну виконувалися функціональні запити до ESP32-C3, наприклад `curl http://192.168.50.25/light/on`.

Після підтвердження роботи HTTP-пристрою перевірялося виконання тих самих дій через голосову команду. У логах сервера мало з'являтися повідомлення про HTTP-запит до IoT-пристрою (рисунок 3.12), а на стороні ESP32-C3 – повідомлення про зміну стану світла (рисунок 3.13).

Для MQTT-керування тестувалося публікація повідомлень у відповідну тему, наприклад `home/light/set`, це можна побачити на рисунках 3.14 та 3.15. За допомогою `mosquitto_sub` можна було простежити, чи надсилає сервер повідомлення ON або OFF після голосової команди. На IoT-пристрої перевірялося отримання MQTT-повідомлення та зміна стану GPIO.

Таким чином, тестування підтвердило роботу основного циклу системи: ESP32-S3 активується за ключовим словом, записує команду, передає аудіо на

Raspberry Pi, сервер розпізнає мовлення, визначає намір, виконує дію або формує відповідь, після чого повертає голосовий звіт на мікроконтролер (рисунок 3.17).

```
[RECV] 160,044 B | 16000 Hz | 1ch | 80,000 samples | 5.00s | RMS=1595.4
[SAVE] /home/rapi/Desktop/pi_server_v1/recordings/alexa_20260503_011737_75663.wav
[STT] 'увімкнув світло'
[IOT] device=light state=on mode=mqtt
[MQTT-IOT] topic=home/light/set payload=ON
[NLU] source=groq normalized='увімкнув світло' intent=light_on conf=1.00 slots={'room': None, 'device': None, 'value': None}
[TTS] text='Світло увімкнено.'
[TTS] generated: /home/rapi/Desktop/pi_server_v1/tts_output/response_20260503_011739_09216.wav
[RESP] returning WAV directly: /home/rapi/Desktop/pi_server_v1/tts_output/response_20260503_011739_09216.wav
[RESP] sending WAV bytes: /home/rapi/Desktop/pi_server_v1/tts_output/response_20260503_011739_09216.wav | 69164 bytes
192.168.50.96 - - [03/May/2026 01:17:41] "POST /audio HTTP/1.1" 200 -
```

Рисунок 3.17 – Основний цикл роботи системи

Напрями подальшого вдосконалення. Попри працездатність реалізованого прототипу, систему можна вдосконалити в кількох напрямках. Насамперед доцільно покращити якість захоплення голосу. Для цього можна використати мікрофонний масив, додати шумозаглушення або оптимізувати розташування мікрофона в корпусі пристрою. Це дозволить підвищити якість запису та точність розпізнавання мовлення.

Ще одним напрямом удосконалення є покращення STT-модуля. У поточній реалізації використовується локальне розпізнавання через Vosk, яке забезпечує автономність, але не завжди ідеально розпізнає українські фрази. У майбутньому можна реалізувати комбінований режим: локальне STT для простих команд і точніший зовнішній сервіс для складніших запитів.

Також можна розширити NLU-модуль. Для цього варто додати більше варіантів формулювання команд, підтримку кімнат, сценаріїв автоматизації та параметрів пристроїв. Наприклад, система може обробляти команди типу «увімкни світло на кухні», «зменш яскравість» або «вимкни всі пристрої».

Окремим напрямом є розширення підтримки IoT-пристроїв. Крім HTTP та MQTT, можна додати інтеграцію з Home Assistant, Zigbee-шлюзами або RF433-модулями. Це дозволить використовувати голосового асистента з більшою кількістю побутових пристроїв.

Також доцільно вдосконалити веб-панель адміністрування. У майбутньому в ній можна реалізувати авторизацію, графік активності команд, перегляд стану пристроїв у реальному часі, налаштування MQTT-тем, IP-адрес вузлів і голосових сценаріїв без редагування конфігураційних файлів.

Крім того, можна покращити стабільність системи в довготривалій роботі. Для цього варто додати автоматичний запуск сервера через systemd, контроль помилок, повторні спроби підключення до IoT-пристроїв, очищення старих аудіофайлів і журналювання критичних подій.

У підсумку подальший розвиток системи може бути спрямований на підвищення точності розпізнавання, покращення якості голосової відповіді, розширення кількості підтримуваних пристроїв і створення більш зручного інтерфейсу для користувача.

ВИСНОВКИ

У кваліфікаційній роботі було розроблено голосового асистента із функціями розумного дому на базі мікроконтролера ESP32-S3 та одноплатного комп'ютера Raspberry Pi. Реалізована система забезпечує приймання голосових команд користувача, їх передавання на серверний вузол, розпізнавання мовлення, визначення наміру, формування відповіді та керування пристроями.

Реалізовано механізм приймання, передавання та обробки голосових команд користувача. Для цього на ESP32-S3 налаштовано роботу з цифровим мікрофоном INMP441 через I2S, запис аудіофрагмента у форматі WAV та передавання його на Raspberry Pi через HTTP. На серверній частині реалізовано приймання аудіофайлу, перевірку його параметрів, збереження запису та подальшу обробку за допомогою Vosk STT.

Розроблено архітектуру голосового асистента з розподілом функцій між ESP32-S3, Raspberry Pi та модулями керування пристроями. ESP32-S3 виконує функції периферійного аудіовузла, Raspberry Pi використовується як центральний сервер обробки, а IoT-модуль забезпечує взаємодію з пристроями розумного дому через HTTP та MQTT. Такий розподіл дозволив зменшити навантаження на мікроконтролер і винести ресурсоємні задачі на більш продуктивний пристрій.

Досліджено сучасні підходи до побудови голосових асистентів і систем розумного дому. Розглянуто особливості комерційних голосових платформ, принципи роботи систем розумного будинку, методи розпізнавання та синтезу мовлення, а також способи інтеграції IoT-пристроїв. На основі проведеного аналізу обґрунтовано доцільність використання локальної або гібридної архітектури для підвищення автономності системи.

Візуалізовано архітектуру системи, алгоритми її роботи та взаємодію між компонентами. У роботі подано структурні схеми голосового асистента, алгоритм проходження команди від користувача до відповіді, схеми обміну даними між ESP32-S3 та Raspberry Pi, а також варіанти керування пристроями

через HTTP і MQTT. Це дозволило наочно показати логіку роботи системи та зв'язки між її основними модулями.

Спроектовано та реалізовано модуль керування пристроями на основі голосових команд. Після розпізнавання тексту NLU-модуль визначає намір користувача після чого сервер передає команду до IoT-пристрою через HTTP або MQTT-повідомлення. Для перевірки роботи використано тестовий пристрій на базі ESP32-C3, який змінює стан виходу після отримання відповідної команди.

Запропоновано підхід до підвищення автономності, гнучкості та зручності використання системи в побутових умовах. Основні команди розумного дому обробляються локально, без обов'язкового звернення до хмарних сервісів. Для інформаційних запитів передбачено використання Groq API, а для голосової відповіді – локальний синтез мовлення через Piper TTS. Додатково реалізовано веб-панель адміністрування, яка дає змогу переглядати стан системи, журнали команд і вручну керувати пристроями.

У результаті тестування підтверджено працездатність основних функцій системи. Асистент здатний активуватися за ключовим словом, записувати голосову команду, передавати її на сервер, розпізнавати мовлення українською мовою, визначати намір користувача, виконувати команду керування та повертати голосову відповідь. Розроблену систему можна використовувати як прототип локального голосового асистента для побутового керування пристроями розумного дому.

Подальше вдосконалення системи може бути пов'язане з покращенням якості розпізнавання мовлення, додаванням шумозаглушення, розширенням списку голосових команд, підтримкою більшої кількості IoT-пристроїв та інтеграцією з платформами автоматизації, зокрема Home Assistant.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Що таке голосовий асистент – Терміни та визначення з кібербезпеки. Keepsolid VPN Unlimited. URL: <https://surl.lu/kqkkue> (дата звернення: 05.02.2026).
2. ASR (автоматичне розпізнавання мовлення) – визначення, випадки використання, приклад. Shaip. URL: <https://uk.shaip.com/blog/automatic-speech-recognition-a-complete-overview/> (дата звернення: 07.02.2026).
3. Розумний дім: розумне опалення вдома | Viessmann UA. Сучасні кліматичні рішення компанії Viessmann | Viessmann UA. URL: <https://www.viessmann.ua/uk/rishennya/smart-tekhnohohiyi/rozumnyy-dim.html> (дата звернення: 10.03.2026).
4. WakeNet Wake Word Model – ESP32 – ESP-SR latest documentation. Technical Documents | Espressif Systems. URL: https://docs.espressif.com/projects/esp-sr/en/latest/esp32/wake_word_engine/README.html (date of access: 12.03.2026).
5. Raspberry Pi 5. Raspberry Pi. URL: <https://www.raspberrypi.com/products/raspberry-pi-5/> (date of access: 08.04.2026).
6. Зустрічайте нове покоління "Малинки" – Raspberry PI 5. Інтернет магазин мікрокомп'ютерів та аксесуарів. URL: <https://evo.net.ua/zustrichaiemo-novui-raspberry-pi-5/> (дата звернення: 08.04.2026).
7. ESP Modules. Espressif Systems. URL: <https://surl.li/rzdegm> (date of access: 08.04.2026).
8. ESP32-S3 Series. Espressif Centralized Documentation Platform (CDP). URL: https://documentation.espressif.com/esp32-s3_datasheet_en.pdf (date of access: 09.04.2026).
9. Omnidirectional Microphone with Bottom Port and I2S Digital Output. 1wire.com.ua URL: <https://1wire.com.ua/download/inmp441.pdf> (date of access: 15.04.2026).

10. MAX98357A Datasheet and Product Info | Analog Devices. Mixed-signal and digital signal processing ICs | Analog Devices. URL: <https://www.analog.com/en/products/max98357a.html> (date of access: 15.04.2026).
11. Формат WAV – IT Master – електроніка та програмування. Головна – IT Master – електроніка та програмування. URL: <https://itmaster.biz.ua/directory/standarts/wav.html> (дата звернення: 17.04.2026).
12. Welcome to Flask – Flask Documentation (3.1.x). Welcome to Flask – Flask Documentation (3.1.x). URL: <https://flask.palletsprojects.com/en/stable/> (date of access: 20.04.2026).
13. Vosk Installation. VOSK Offline Speech Recognition API. URL: <https://alphacephei.com/vosk/install> (date of access: 20.04.2026).
14. OpenAI Compatibility – GroqDocs. GroqDocs. URL: <https://console.groq.com/docs/openai> (date of access: 21.04.2026).
15. MQTT Essentials | 2026 Guide | All The Core Concepts & Basics Explained. HiveMQ – Stream Data. Build Intelligence. Activate AI. URL: <https://www.hivemq.com/mqtt/> (date of access: 21.04.2026).
16. What is MQTT and How It Works | Random Nerd Tutorials. Random Nerd Tutorials. URL: <https://randomnerdtutorials.com/what-is-mqtt-and-how-it-works/> (date of access: 21.04.2026).
17. Overview of HTTP – HTTP | MDN. MDN Web Docs. URL: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/Overview> (date of access: 24.04.2026).
18. Espressif IoT Development Framework – PlatformIO latest documentation. Redirecting... URL: <https://docs.platformio.org/en/latest/frameworks/espidf.html> (date of access: 24.04.2026).
19. MQTT – The Standard for IoT Messaging. MQTT – The Standard for IoT Messaging. URL: <https://mqtt.org/> (date of access: 25.04.2026).
20. Eclipse Mosquitto. Eclipse Mosquitto. URL: <https://mosquitto.org/> (date of access: 25.04.2026).

21. Menezes N. Building a web application interface with Flask. Medium. URL: <https://medium.com/@menezesnatalia/building-a-web-application-interface-with-flask-794f09238946> (date of access: 25.04.2026).