НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра системного програмування і спеціалізованих

комп'ютерних систем

До захисту допущено:

Завідувач кафедри

_____ Віталій РОМАНКЕВИЧ

«___»_____2025 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою

«Системне програмування та спеціалізовані комп'ютерні системи»

спеціальності 123 «Комп'ютерна інженерія»

на тему: «Комп'ютерна система голосового управління ліфтом на основі мікроконтролера ESP32»

Виконав (-ла):
студент (-ка) IV курсу, групи КВ-12
Гусельніков Антон Олексійович _____

Керівник:
Доцент кафедри СПІСКС, к.т.н., доцент,
Петрашенко Андрій Васильович _____

Консультант з нормоконтролю:
Доцент кафедри СПІСКС, к.т.н., доцент,
Клятченко Ярослав Михайлович _____

Рецензент:
Доцент кафедри ПЗКС, к.т.н., доцент,
Юрчишин Василь Якович _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2025 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет прикладної математики
Кафедра системного програмування і
спеціалізованих комп'ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 123 «Комп'ютерна інженерія»

Освітньо-професійна програма «Системне програмування та спеціалізовані комп'ютерні системи»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Віталій РОМАНКЕВИЧ
«___»_____2025 р.

ЗАВДАННЯ

на дипломний проєкт студенту

Гусельніков Антон Олексійович

1. Тема проєкту «Комп'ютерна система голосового управління ліфтом на основі мікроконтролера ESP32», керівник проєкту Петрашенко Андрій Васильович, к.т.н., доцент, затверджені наказом по університету від «29» травня 2025 р. №1808-С
2. Термін подання студентом проєкту _____
3. Вихідні дані до проєкту: див. Технічне завдання
4. Зміст пояснювальної записки
   - АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ОБҐРУНТУВАННЯ ТЕМИ ДИПЛОМНОГО ПРОЄКТУ
   - АПАРАТНО-ПРОГРАМНІ КОМПОНЕНТИ СИСТЕМИ: ОГЛЯД ТЕХНІЧНИХ МОЖЛИВОСТЕЙ
   - АЛГОРИТМИ ФУНКЦІОНУВАННЯ СИСТЕМИ
   - ТЕСТУВАННЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ СИСТЕМИ
5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо)
   - Комп'ютерна система голосового управління ліфтом на основі мікроконтролера ESP32. Схема електрична принципова
   - Алгоритм взаємодії користувача з системою. Блок-схема
   - Алгоритм роботи апаратної частини системи. Блок-схема
   - Алгоритм роботи серверної частини системи. Блок-схема

6. Консультанти розділів проекту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Клятченко Я.М., к.т.н., доцент каф. СП і СКС		

7. Дата видачі завдання

Календарний план

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів	Примітка
1.	Вивчення літератури за тематикою проекту	15.12.2024	
2.	Розроблення та узгодження технічного завдання	30.01.2025	
3.	Аналіз існуючих рішень	01.04.2025	
4.	Підготовка матеріалів першого розділу дипломного проекту	08.04.2025	
5.	Підготовка матеріалів другого розділу дипломного проекту	22.04.2025	
6.	Підготовка матеріалів третього розділу дипломного проекту	06.05.2025	
7.	Підготовка матеріалів четвертого розділу дипломного проекту	18.05.2025	
8.	Підготовка графічної частини дипломного проекту	20.05.2025	
9.	Оформлення документації дипломного проекту	25.05.2025	
10.	Попередній огляд матеріалів диплому на кафедрі	30.05.2025	

Студент _____ Антон ГУСЕЛЬНІКОВ
(підпис)

Керівник проекту _____ Андрій ПЕТРАШЕНКО
(підпис)

АНОТАЦІЯ

Кваліфікаційна робота включає пояснювальну записку (63 с., 29 рис. 1 табл., 5 додатків).

Об’єкт розробки – створення комп'ютерної системи голосового управління ліфтом на основі мікроконтролера ESP32.

Комп’ютерна система дозволяє: здійснювати запис аудіо; аналізувати аудіо та співставлення з наявними командами; проводити налаштування системи за рахунок WIFI з’єднання; зберігати проведені налаштування; показувати конфігурацію через веб-сайт; передавати данні через мережу Internet; комунікувати з користувачем; передбачені механізми захисту від помилок; забезпечувати цілісність даних, що передаються.

У процесі розробки були використані мови програмування Python та C++. Застосовувалися як базові бібліотеки, так і бібліотеки Arduino. Як середовища розробки було обрано PyCharm та Arduino IDE. Із хмарних сервісів використано AWS EC2. Також було інтегровано OpenAI API.

В ході розробки:

- розроблено апаратну частину;

- розроблено програмне забезпечення;

- проведено аналіз існуючих рішень;

- досліджено засоби реалізації;

Ключові слова: Голосове управління ліфтом, налаштування системи, мікроконтролер ESP32, AWS EC2, C++, Python, Arduino, OpenAI API.

ABSTRACT

The qualification work includes an explanatory note (63 pages, 29 figures, 1 table, 5 appendices).

The object of development is the creation of a computer-based voice control system for an elevator based on the ESP32 microcontroller. The computer system provides the following capabilities: audio recording; audio analysis and matching with available commands; system configuration via Wi-Fi connection; saving the applied settings; displaying the configuration through a web interface; data transmission over the Internet; user interaction; built-in error protection mechanisms; and ensuring the integrity of transmitted data.

During development, the Python and C++ programming languages were used. Both standard and Arduino libraries were applied. PyCharm and the Arduino IDE were chosen as development environments. AWS EC2 cloud services were utilized. Additionally, the OpenAI API was integrated.

During the project, the following tasks were completed:

- development of the hardware part;
- development of the software;
- analysis of existing solutions;
- research on implementation tools.

Keywords: Voice elevator control, system configuration, ESP32 microcontroller, AWS EC2, C++, Python, Arduino, OpenAI API.

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ.045200.002 ТЗ	Комп’ютерна	6		
			система голосового			
			управління ліфтом			
			на основі			
			мікроконтролера			
			ESP32.			
			Технічне завдання			
	А4	ІАЛЦ.045200.003 ТП	Комп’ютерна	3		
			система голосового			
			управління ліфтом			
			на основі			
			мікроконтролера			
			ESP32.			
			Відомість технічного			
			проекту			

					ІАЛЦ.045200.001 ОА		
Змін.	Арк.	№ докум.	Підпис	Дата			
Розробив		Гусельніков А. О.			Комп'ютерна система голосового управління ліфтом на основі мікроконтролера ESP32 **Опис альбому**	Літ.	Аркуш 1 / Аркушів 4
Перевірив		Петрашенко А.В.					
Консульт.						КПІ ім. Ігоря Сікорського, ФПМ, КВ-12	
Н. контроль		Клятченко Я.М.					
Зав.каф.		Романкевич В.О.					

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ.045200.004 ПЗ	Комп’ютерна	63		
			система голосового			
			управління ліфтом			
			на основі			
			мікроконтролера			
			ESP32.			
			Пояснювальна			
			записка			
	А4	ІАЛЦ.045200.005 Е3	Комп’ютерна	1		
			система голосового			
			управління ліфтом			
			на основі			
			мікроконтролера			
			ESP32.			
			Схема електрична			
			принципова			

					ІАЛЦ.045200.001 ОА	Арк.
						2
Змін.	Арк.	№ докум.	Підпис	Дата		

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ.045200.006 ПЕ	Схема електрична	1		
			принципова			
			Перелік елементів			
	А4	ІАЛЦ.045200.007 Д1	Алгоритм взаємодії	1		
			користувача з			
			системою.			
			Блок-схема			
	А4	ІАЛЦ.045200.008 Д2	Алгоритм роботи	1		
			апаратної частини			
			Системи.			
			Блок-схема			
	А4	ІАЛЦ.045200.009 Д3	Алгоритм роботи	1		
			серверної частини			
			Системи.			
			Блок-схема			

Змін.	Арк.	№ докум.	Підпис	Дата	ІАЛЦ.045200.001 ОА	Арк. 3

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ.045200.010 Д4	Комп’ютерна	1		
			система голосового			
			управління ліфтом			
			на основі			
			мікроконтролера			
			ESP32.			
			Текст програмного коду			
		Диск DVD-ROM	Текст пояснювальної			
			Записки.			
			Графічний матеріали			

						Арк.
					ІАЛЦ.045200.001 ОА	4
Змін.	Арк.	№ докум.	Підпис	Дата		

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ РОЗРОБКИ

Назва розробки: «Комп'ютерна система голосового управління ліфтом на основі мікроконтролера ESP32».

Галузь застосування: житлова та комерційна нерухомість, соціальна інфраструктура, розумні будинки та офіси.

2. ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на дипломне проектування на здобуття першого (бакалаврського) рівня вищої освіти, затверджене кафедрою системного програмування і спеціалізованих комп’ютерних систем Національного технічного університету України «Київський Політехнічний Інститут імені Ігоря Сікорського».

3. МЕТА І ПРИЗНАЧЕННЯ РОБОТИ

Метою даного проєкту є створення комп’ютерної системи голосового управління ліфтом на основі мікроконтролера ESP32, яка забезпечує взаємодію з користувачем за допомогою голосових команд, підтримує аудіозворотний зв’язок, а також дозволяє здійснювати конфігурацію параметрів системи через вбудований вебінтерфейс.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелом інформації є технічна та науково-технічна література, технічна документація, публікації у періодичних виданнях та електронні статті у мережі Інтернет.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до програмного продукту, що розробляється

- запис аудіо фрагментів;
- передача аудіо фрагментів до серверу у реальному часі;
- можливість налаштування апаратної частини;
- легке підключення системи поверх існуючих базових ліфтових систем;
- можливість зручного розширення функціоналу:
    - додати функціонал для повідомлення ліфтера;
    - додати базу даних;
    - додати веб-інтерфейс для налашутвання серверу;
    - створення та інтеграція мобільного застосунку.
- мінімальна вартість;
- взаємодія з користувачем;
- максимальна оптимізація;
- наявність базового захисту апаратної частини;
- можливість підключення до Internet.

5.2. Вимоги до апаратного забезпечення

- Підтримка інтерфейсів UART, I2S, GPIO;
- Підтримка Wi-Fi;
- Живлення: 5V з можливістю забезпечити щонайменше 1 А струму;
- Оперативна пам’ять: не менше 520 КБ SRAM;
- Вбудована флеш-пам’ять: не менше 4 МБ;

- Підтримка microSD карт з обсягом не менше 1 GB для зберігання даних;
- Підтримка відтворення MP3-файлів з microSD;
- Мікрофонний модуль з I2S-інтерфейсом
- Динамік до 3 Вт, 8 Ом;
- Наявність маршрутизатора з доступом до мережі Internet;
- Хост-сервер для обробки аудіо:
    - Процесор: не нижче Intel® Xeon® або AMD EPYC;
    - Оперативна пам’ять: від 1 ГБ (також 1 ГБ файлу підкачки);
    - Сховище: не менше 5 ГБ для тимчасових файлів аудіо;
    - Операційна система: Linux;
    - Підключення до Інтернету зі швидкістю не нижче 5 Мбіт/с.
- Процесор: Intel® Pentium®.
- Оперативна пам’ять: 4 Гб.
- Простір на диску: 4 Гб.
- Тип системи: 64-розрядна операційна система.

5.3. Вимоги до програмного та апаратного забезпечення користувача

- Смартфон, планшет або ноутбук з можливістю підключення до Wi-Fi для доступу до вебінтерфейсу конфігурації системи;
- Операційна система:
    - Для смартфонів: Android 8.0+ або iOS 13+;
    - Для ноутбуків/ПК: Windows 10+, macOS 10.15+, Linux (будь-який сучасний дистрибутив).
- Браузер з підтримкою HTML5, JavaScript:

- Google Chrome (рекомендовано);
- Mozilla Firefox;
- Safari;
- Microsoft Edge.

6. ЕТАПИ РОЗРОБКИ

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів
1.	Вивчення літератури за тематикою проекту	15.12.2024
2.	Розроблення та узгодження технічного завдання	30.01.2025
3.	Аналіз існуючих рішень	01.04.2025
4.	Підготовка матеріалів першого розділу дипломного проекту	08.04.2025
5.	Підготовка матеріалів другого розділу дипломного проекту	22.04.2025
6.	Підготовка матеріалів третього розділу дипломного проекту	06.05.2025
7.	Підготовка матеріалів четвертого розділу дипломного проекту	18.05.2025
8.	Підготовка графічної частини дипломного проекту	20.05.2025
9.	Оформлення документації дипломного проекту	25.05.2025
10.	Попередній огляд матеріалів диплому на кафедрі	30.05.2025

ВІДОМІСТЬ ТЕХНІЧНОГО ПРОЄКТУ

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ.045200.004 ПЗ	Комп’ютерна система голосового управління ліфтом на основі мікроконтролера ESP32. Пояснювальна записка	63		
	А4	ІАЛЦ.045200.005 Е3	Комп’ютерна система голосового управління ліфтом на основі мікроконтролера ESP32. Схема електрична принципова	1		

					ІАЛЦ.045200.003 ТП			
Змін.	Арк.	№ докум.	Підпис	Дата				
Розробив		Гусельніков А. О.			Комп'ютерна система голосового управління ліфтом на основі мікроконтролера ESP32 **Відомість технічного проєкту**	Літ.	Аркуш	Аркушів
Перевірив		Петрашенко А.В.					1	3
Консульт.						КПІ ім. Ігоря Сікорського, ФПМ, КВ-12		
Н. контроль		Клятченко Я.М.						
Зав.каф.		Романкевич В.О.						

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість документів	№ прим.	Примітки
	А4	ІАЛЦ.045200.006 ПЕ	Схема електрична	1		
			принципова			
			Перелік елементів			
	А4	ІАЛЦ.045200.007 Д1	Алгоритм взаємодії	1		
			користувача з			
			системою.			
			Блок-схема			
	А4	ІАЛЦ.045200.008 Д2	Алгоритм роботи	1		
			апаратної частини			
			Системи.			
			Блок-схема			
	А4	ІАЛЦ.045200.009 Д3	Алгоритм роботи	1		
			серверної частини			
			Системи.			
			Блок-схема			

					ІАЛЦ.045200.003 ТП	Арк.
						2
Змін.	Арк.	№ докум.	Підпис	Дата		

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ.045200.010 Д4	Комп’ютерна	1		
			система голосового			
			управління ліфтом			
			на основі			
			мікроконтролера			
			ESP32.			
			Текст програмного коду			
		Диск DVD-ROM	Текст пояснювальної			
			Записки.			
			Графічний матеріали			

					ІАЛЦ.045200.003 ТП	Арк.
						3
Змін.	Арк.	№ докум.	Підпис	Дата		

Пояснювальна записка

до дипломного проєкту

на тему: Комп'ютерна система голосового управління ліфтом на основі мікроконтролера ESP32

Київ-2025

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

IoT (Internet of Things) — «Інтернет речей» — концепція мережі пристроїв, які мають сенсори, виконавчі механізми та можливість підключення до інтернету для збору, обміну й обробки даних. У цьому проєкті IoT-пристрій забезпечує голосове керування ліфтом через локальну мережу з підключенням до сервера [1].

I2S (Inter-IC Sound) — цифровий послідовний інтерфейс, призначений для передавання аудіоданих між мікросхемами, зокрема між мікрофоном і мікроконтролером. У даному проєкті використовується для зчитування аудіосигналу з цифрового мікрофона [2].

WebSocket — протокол мережевого зв’язку, що забезпечує постійне двостороннє з’єднання між клієнтом і сервером у режимі реального часу. Широко використовується для онлайн-спілкування, передачі даних без затримок тощо [3].

UART (Universal Asynchronous Receiver-Transmitter) — універсальний асинхронний приймач-передавач, що використовується для послідовної передачі даних між пристроями [4].

RX (Receive) — лінія прийому даних в інтерфейсі UART. Вона отримує дані, що надходять від передавача іншого пристрою.

TX (Transmit) — лінія передачі даних в інтерфейсі UART. Через неї мікроконтролер або інший пристрій надсилає дані до зовнішнього обладнання.

I2C (Inter-Integrated Circuit) — послідовний інтерфейс для обміну даними між мікроконтролером і периферійними пристроями з використанням лише двох дротів (SDA та SCL) [5].

SPI (Serial Peripheral Interface) — швидкий синхронний послідовний інтерфейс, що використовується для підключення мікроконтролера до периферії, наприклад дисплеїв, пам’яті тощо [6].

API (Application Programming Interface) — інтерфейс прикладного програмування; набір правил і методів, які дозволяють одній програмі взаємодіяти з іншою [7 - 8].

ESP32 — потужний мікроконтролер із підтримкою Wi-Fi та Bluetooth, розроблений компанією Espressif. Підходить для створення IoT-рішень і вбудованих систем [9].

с (секунди) — одиниця часу в міжнародній системі одиниць (SI), що дорівнює 1 секунді.

мс (мілісекунди) — одна тисячна частина секунди (1 мс = 0.001 с).

GND (Ground) — загальний (нульовий) провід, спільна точка відліку напруги в електричному колі.

HTTP (Hypertext Transfer Protocol) — протокол передачі гіпертексту, який використовується для зв’язку між веб-браузером і сервером [10].

MicroSD — компактна карта пам’яті, яка використовується для зберігання даних у портативних пристроях [11].

DFPlayer Mini — компактний MP3-модуль для відтворення аудіофайлів з microSD-карти, який може керуватись мікроконтролером [12].

INMP441 — цифровий мікрофон з інтерфейсом I²S, який використовується для зчитування аудіосигналів [13].

GPIO (General-Purpose Input/Output) — універсальні порти введення/виведення на мікроконтролерах, які можуть програмно налаштовуватись для різних цілей.

CLK (Clock) — сигнал синхронізації, який визначає ритм обміну даними в цифрових пристроях.

WS (Word Select / Left-Right Clock) — сигнал у I²S-інтерфейсі, що вказує, до якого аудіоканалу належить поточний біт (лівий або правий).

SD (Serial Data / Data In) — лінія послідовної передачі даних, наприклад, у SPI або I²S.

VCC / VDD — позначення живлення:

VCC (Voltage at the Collector) — джерело живлення для цифрових схем (частіше — логіка CMOS/TTL).

VDD (Voltage at the Drain) — напруга живлення в логіці MOSFET.

AWS (Amazon Web Services) — хмарна платформа від Amazon, яка надає обчислювальні ресурси, сховище, бази даних, аналітику тощо [14].

EC2 (Elastic Compute Cloud) — сервіс у межах AWS, що дозволяє орендувати віртуальні сервери для розміщення додатків і обробки даних.

LINUX — вільна й відкрита операційна система, що часто використовується в серверних, вбудованих та IoT-системах.

JSON (JavaScript Object Notation) — легкий формат обміну даними, що використовується для серіалізації структурованої інформації у вигляді тексту. Застосовується в API, конфігураціях тощо [15].

ВСТУП

Сучасна інженерія дедалі частіше спрямована на розробку рішень, що полегшують повсякденне життя людини, роблять технології більш доступними, зручними та інклюзивними. Одним із таких напрямів є голосове керування пристроями, що дозволяє взаємодіяти з технікою без фізичних дій — лише за допомогою мови. Такий підхід особливо актуальний в умовах зростання потреб у безконтактних рішеннях, а також у контексті розширення можливостей людей з обмеженою рухливістю.

Повномасштабна війна в Україні стала не лише трагедією, а й важким викликом для суспільства. Серед її наслідків — значне збільшення кількості людей з інвалідністю, зокрема воїнів, які повертаються з фронту з втраченими кінцівками або серйозними пораненнями. У таких умовах критично важливо розробляти технології, що забезпечують цим людям гідний рівень життя, незалежність і безбар’єрний доступ до інфраструктури.

Одним із елементів такої інфраструктури є ліфт. Традиційні системи управління ним передбачають використання фізичних кнопок, що створює додаткові труднощі для людей із вадами опорно-рухового апарату. Рішенням може стати система голосового управління, яка дозволить взаємодіяти з ліфтом безконтактно — лише за допомогою голосової команди.

У цій дипломній роботі представлено прототип комп'ютерної системи голосового управління ліфтом на основі мікроконтролера ESP32. Це лише перша, маленька, але дійсно важлива сходинка на шляху до створення інклюзивного технологічного середовища. Вона демонструє, що навіть з обмеженим бюджетом, використовуючи доступні мікроконтролери та програмне забезпечення, можна реалізувати рішення, які здатні змінити життя конкретної людини на краще.

Прототип розроблено з урахуванням можливості подальшого масштабування, модернізації та адаптації до реальних умов експлуатації. Його головна мета — продемонструвати принципову можливість реалізації

голосового інтерфейсу для системи керування ліфтом, що базується на сучасних технологіях — мікроелектроніці, обробці звуку, інтернеті речей (IoT) та хмарних обчисленнях.

Цей проєкт не лише технічна розробка — це вклад у побудову доступного, гідного і людяного середовища, в якому кожен матиме змогу жити без бар’єрів.

1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ОБҐРУНТУВАННЯ ТЕМИ ДИПЛОМНОГО ПРОЄКТУ

Перший розділ цієї роботи присвячено огляду існуючих рішень у сфері автоматизації та модернізації ліфтових систем. Особливу увагу приділено огляду традиційних і сучасних систем керування ліфтами, аналізу рішень провідних виробників, а також обґрунтуванню доцільності розробки власного IoT-пристрою для голосового керування ліфтом. Огляд мікроконтролерних платформ і IoT-архітектури дозволив сформувати технічну основу для розробки ефективного, доступного та адаптивного рішення, орієнтованого на українські реалії.

1.1. Стандартні системи керування ліфтами

Більшість існуючих ліфтів обладнані традиційними кнопковими панелями керування — як всередині кабіни, так і на поверхах. Такі системи залишаються зручними та звичними для більшості користувачів, але мають обмеження щодо інклюзивності. Для людей з інвалідністю, літніх людей або в ситуаціях, коли руки зайняті, кнопкове керування стає недоступним або незручним.

У новіших моделях ліфтів почали впроваджувати додаткові способи керування — безконтактні кнопки, системи розпізнавання карт доступу, мобільні застосунки тощо. Однак подібні системи залишаються малопоширеними в Україні та практично не інтегруються в уже наявні ліфти.

1.2. Комерційні рішення: аналіз функціональних можливостей та обмежень

На світовому ринку присутні кілька великих гравців, що пропонують розширені рішення для автоматизації ліфтів:

- Otis (США) — деякі моделі Otis Gen3 та Gen360 підтримують цифрову платформу Otis ONE, яка дає змогу здійснювати моніторинг та виклик ліфта за допомогою мобільного додатку eCall. Хоча безпосередньо голосове керування не вбудоване, додаток підтримує взаємодію через голосові помічники на смартфоні. Це рішення орієнтоване на нові ліфти та є частиною закритої екосистеми [16].
- KONE (Фінляндія) — має рішення KONE Elevator Call, що дозволяє викликати ліфт через смартфон. Також представлена система KONE KDS Home V, яка підтримує голосовий виклик ліфта у житлових будинках. Платформа KONE 24/7 Connected Services реалізує IoT-моніторинг. Проте підтримки української мови немає, а обладнання є дорогим [17].
- Schindler (Швейцарія) — пропонує рішення myPORT і CleanCall HoloVoice. CleanCall HoloVoice дозволяє безконтактно взаємодіяти з ліфтом за допомогою голографічних кнопок і голосового управління. Це рішення орієнтоване на висококласні будівлі й вимагає спеціального дорогого обладнання. Українська мова не підтримується [18].
- Thyssenkrupp (Німеччина) — реалізує систему MAX, яка аналізує стан ліфтів у хмарі для запобігання поломкам. Це рішення складне для інтеграції в старі ліфти й не призначене для локалізації [19].

Незважаючи на інноваційність, усі згадані системи мають спільні недоліки:

- Висока вартість впровадження.
- Залежність від виробника та неможливість інтеграції в стороннє обладнання.
- Відсутність підтримки української мови.

- Неможливість або складність встановлення поверх старих ліфтів у житловому фонді.

1.3. Аналіз мікроконтролерних платформ для реалізації

Для створення недорогого, але функціонального пристрою, було проаналізовано кілька мікроконтролерних платформ:

- Arduino UNO / Nano — прості у використанні, але мають обмежену пам’ять і потужність.
- Raspberry Pi — потужний варіант, проте не завжди зручний для вбудованих систем через великі розміри та споживання енергії.
- ESP8266 — компактний, але менш функціональний за ESP32.
- ESP32 — двоядерний мікроконтролер з підтримкою Wi-Fi, Bluetooth, I2S, великою кількістю GPIO, що ідеально підходить для збору аудіо та передачі його на сервер.

Вибір зупинився на ESP32, як на найкращому рішенні для реалізації голосового інтерфейсу та комунікації з сервером.

1.4. IoT-рішення та розвиток

Моя ідея полягає у створенні компактного IoT-пристрою, який об'єднує ESP32, мікрофон I2S, модуль DFPlayer mini для відтворення голосових відповідей, Wi-Fi модуль для підключення до мережі, вебінтерфейс для налаштування та WebSocket для передавання аудіо на сервер. На сервері відбувається розпізнавання голосу за допомогою VOSK.

Модель VOSK — це офлайн-система розпізнавання, яка є безкоштовною, підтримує українську мову та не вимагає значних обчислювальних ресурсів [20]. У цьому проєкті VOSK працює на сервері, тому інтернет-з’єднання все ще необхідне, проте вся обробка даних залишається під контролем користувача, без передачі інформації до сторонніх сервісів.

Прототип розроблено таким чином, щоб його можна було легко встановити на вже існуючі ліфти, не змінюючи їх конструкцію. Це робить систему доступною для широкого використання.

Переваги запропонованого рішення:

- Орієнтація на українського користувача (підтримка мови, локальні реалії).
- Модульність — легко адаптується під різні конфігурації ліфтів.
- Низька вартість у порівнянні з комерційними альтернативами.
- Відкритість — програмна частина не прив’язана до одного постачальника.

Подальші напрями розвитку:

- Підключення бази даних для зберігання історії команд, інцидентів, активностей та налаштувань.
- Механізм сповіщення ліфтерів через сервер у разі виявлення помилок або збоїв.
- Розробка мобільного застосунку для обслуговування та діагностики.
- Розширення словника команд і багатомовна підтримка.

Висновок

Отже, було проведено ґрунтовний аналіз сучасних та традиційних рішень у сфері ліфтових систем. Встановлено, що переважна більшість ліфтів, особливо в Україні, все ще використовує кнопкові панелі керування, які є незручними або недоступними для людей з інвалідністю, літніх осіб та у випадках, коли руки зайняті. Сучасні комерційні рішення, що включають мобільні додатки, голосових асистентів та хмарні сервіси, реалізовані лише в окремих новітніх моделях ліфтів і не адаптовані до умов українського житлового фонду. Крім того, вони потребують значних фінансових витрат,

мають обмежену підтримку мов (відсутність української), часто закриту архітектуру та складність інтеграції у вже встановлене обладнання.

Аналіз мікроконтролерних платформ показав, що оптимальним рішенням для побудови бюджетного, але функціонального пристрою голосового керування є мікроконтролер ESP32, який забезпечує необхідну обчислювальну потужність, підтримку периферії та мережеву взаємодію. У поєднанні з відкритим програмним забезпеченням для розпізнавання мови (VOSK) та модульним IoT-підходом було розроблено концепцію пристрою, здатного ефективно працювати на вже існуючих ліфтах без суттєвого втручання в їх конструкцію.

Таким чином, доцільність розробки власного IoT-рішення обґрунтована:

- технічною доступністю реалізації;
- економічною вигідністю порівняно з промисловими аналогами;
- адаптацією під локальні потреби та українську мову;
- гнучкістю інтеграції в наявну інфраструктуру.

Запропонований підхід відкриває нові можливості для модернізації ліфтів з урахуванням вимог інклюзивності, доступності та автономності.

2. АПАРАТНО-ПРОГРАМНІ КОМПОНЕНТИ СИСТЕМИ: ОГЛЯД ТЕХНІЧНИХ МОЖЛИВОСТЕЙ

На схемі (рис. 2.1) зображено взаємодію між апаратними й програмними модулями системи голосового керування ліфтом на базі ESP32. Ось короткий опис кожного модуля та його призначення:

ESP32:

- Головний модуль – відповідає за загальну координацію роботи системи, обробку вхідних даних та взаємодію з іншими компонентами.
- Модуль команд ліфту – надсилає команди до системи ліфту.
- Модуль керування DFPlayerMini – контролює аудіопрогравач DFPlayerMini для відтворення аудіо-відповідей.

Інші компоненти:

- Мікрофон INMP441 – цифровий мікрофон, що передає аудіо на ESP32 для надсилання на сервер.
- DFPlayerMini – мініатюрний MP3-програвач для озвучення відповідей.
- Динамік – виводить звук із DFPlayerMini.
- Вебсервер – призначений для налаштування системи.

AWS EC2:

- Головний модуль – обробляє запити з ESP32, проводить розпізнавання мовлення.
- Модуль для роботи з OpenAI API – відповідає за приведення команд до наявного пулу.

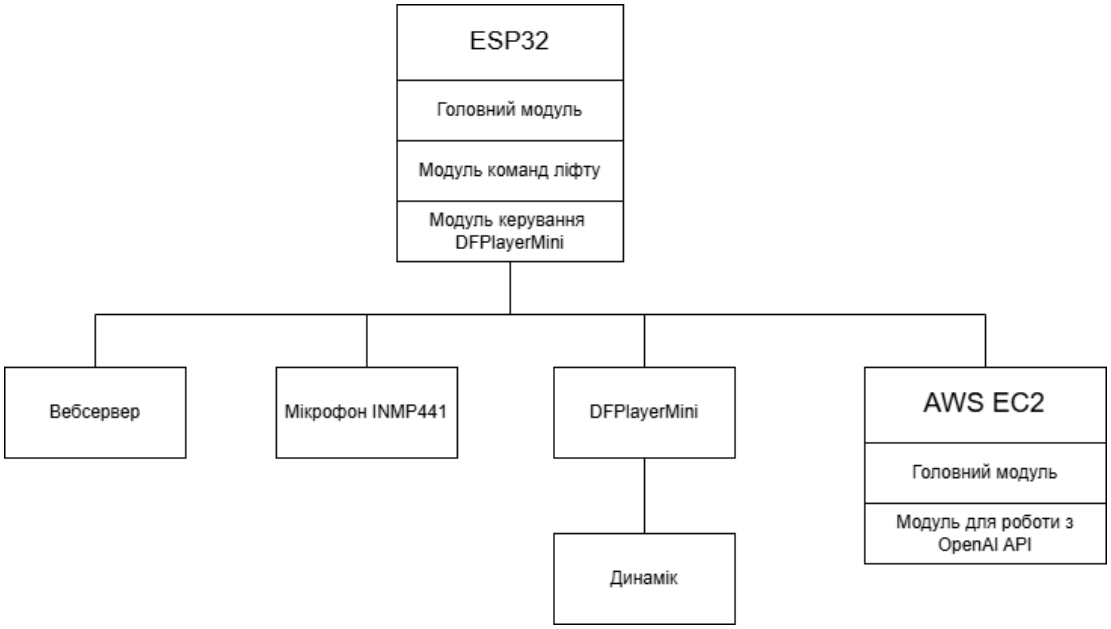


Рис. 2.1 – Схема взаємодії модулів

2.1. Апаратні компоненти

У цьому підрозділі представлено огляд основних апаратних компонентів, що формують прототип комп’ютерної системи голосового керування ліфтом. Компоненти обрано з урахуванням критерію сумісності, низького енергоспоживання, доступності на українському ринку та можливості подальшого масштабування. Принципову електричну схему з'єднань наведено на рис. 2.2.

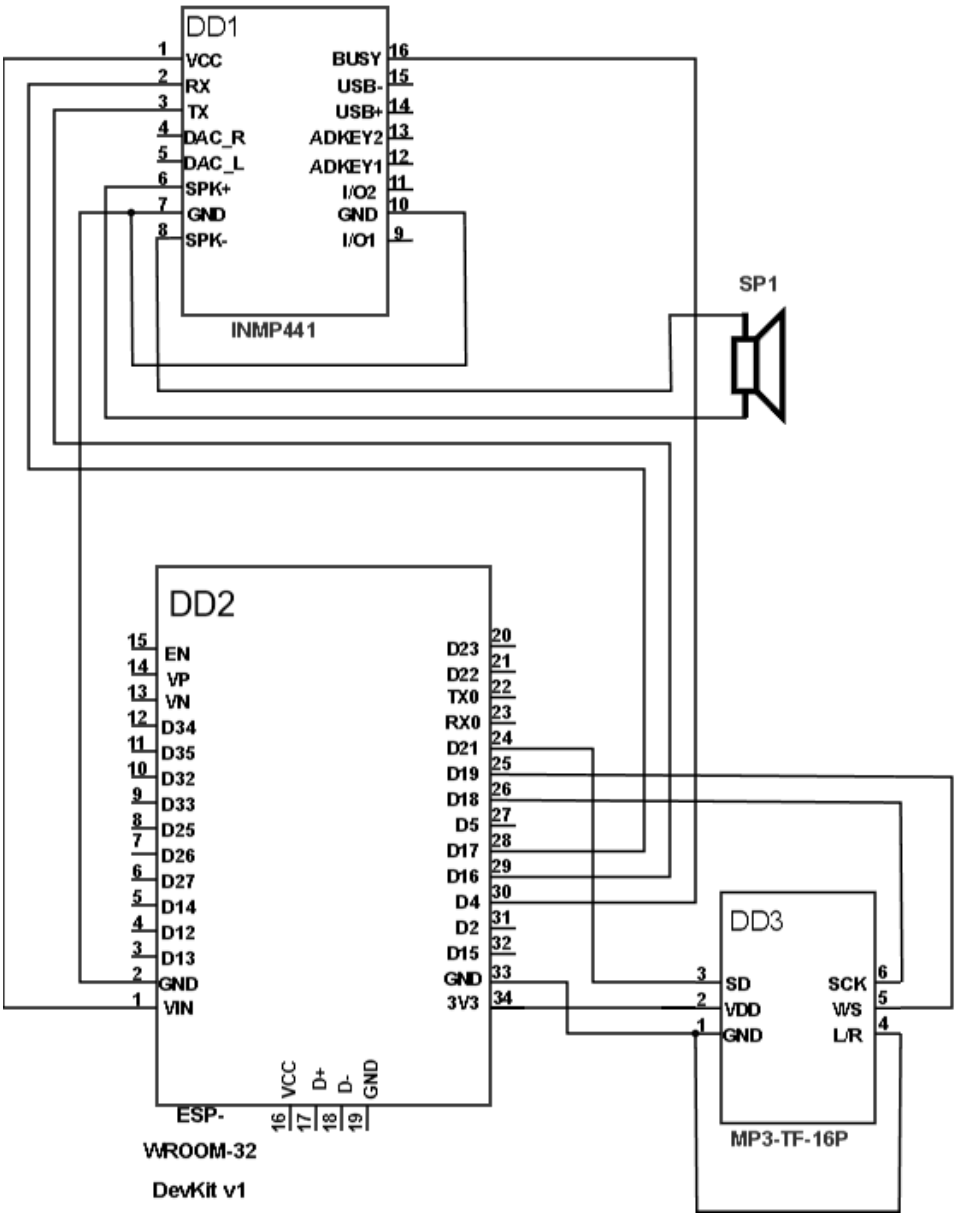


Рис. 2.2 – Принципова схема підключення компонентів системи

Нижче подано опис усіх елементів зі схеми, включаючи їх загальне призначення, виробника (де це актуально), технічні характеристики та роль у системі:

- DD1. Мікрофонний модуль INMP441 (I2S-мікрофон). Цифровий всеспрямований мікрофон із підтримкою інтерфейсу I2S, що дозволяє безпосередньо зчитувати аудіо у цифровому вигляді без потреби в АЦП. Призначення: зчитування голосових команд користувача. Виробник: InvenSense або OEM-виробники.

- DD2. ESP32 DEVKIT V1 (Espressif Systems). Це основний мікроконтролер системи, на якому реалізовано логіку взаємодії з усіма іншими компонентами. Модуль має двоядерний 32-бітний процесор, вбудований Wi-Fi та Bluetooth, а також апаратну підтримку інтерфейсів UART, I2C, SPI та I2S. Призначення: обробка голосових даних, керування периферією, комунікація із сервером. Виробник: Espressif Systems (Китай).
- DD3. DFPlayer Mini (MP3-TF-16P). Компактний аудіопрогравач, що здатен відтворювати MP3-файли з microSD карти. Підтримує керування через UART або кнопки. Працює автономно, потребує лише живлення та команди з ESP32. Призначення: відтворення попередньо записаних голосових повідомлень. Виробник: YX Electronics (Китай).
- SP1. Динамік 3 Вт, 8 Ом. Пасивний акустичний елемент, підключений до DFPlayer Mini. Призначення: відтворення аудіоповідомлень для користувача. Виробник: залежить від постачальника, використовується недорогий стандартний динамік.

Усі обрані компоненти є масово доступними, мають відкриту документацію та підтримуються спільнотою розробників. Це дозволяє не лише зменшити вартість системи, а й забезпечити її ремонтопридатність, повторюваність та адаптованість до подальших змін.

2.2. Програмна реалізація системи

2.2.1. Огляд використаних бібліотек

Для реалізації функціональності системи було використано набір як вбудованих, так і сторонніх бібліотек, які забезпечують роботу з апаратними модулями, мережею, передачею даних та обробкою аудіо. Підрозділ поділений

на бібліотеки, що використовуються на стороні ESP32 та серверної частини на Python.

2.2.1.1. Arduino та базові бібліотеки C++

У цій частині розглядаються основні бібліотеки Arduino та C++, які використовуються для управління апаратними засобами ESP32.

- WiFi.h: Ця бібліотека використовується для підключення ESP32 до мережі Wi-Fi як у режимі станції (STA), так і в режимі точки доступу (AP). Забезпечує функції керування з'єднанням, отримання IP-адреси, перевірки статусу підключення тощо [21].
- WebServer.h: Дає змогу створити простий синхронний HTTP-сервер на ESP32. Використовується для обробки HTTP-запитів, що надходять до пристрою, наприклад, із веб-інтерфейсу конфігурації [22].
- Preferences.h: Дозволяє зберігати і зчитувати налаштування в енергонезалежну пам'ять (NVS – Non-Volatile Storage) ESP32. Застосовується для збереження даних, які мають залишатися після перезавантаження, наприклад, конфігурацій Wi-Fi [23].
- driver/i2s.h: Це низькорівнева бібліотека ESP-IDF, що забезпечує доступ до інтерфейсу I2S (Inter-IC Sound), який використовується для передачі цифрового аудіо. Застосовується для зчитування даних з мікрофона INMP441 у реальному часі [24].
- WebSocketsClient.h: Бібліотека для створення WebSocket-клієнта. Забезпечує двосторонній зв'язок у режимі реального часу між ESP32 та сервером, що особливо важливо для передачі аудіо та отримання команд з хмари [25].
- ArduinoJson.h: Ця бібліотека використовується для серіалізації та десеріалізації JSON-даних. Дозволяє зручно створювати, зчитувати

та обробляти JSON-повідомлення, що передаються через WebSocket або HTTP [26].

- queue: Стандартна бібліотека C++ для роботи з чергами. У вашій програмі може використовуватись для збереження аудіофрагментів перед їх передачею на сервер [27].
- vector: Ще одна стандартна бібліотека C++ STL, яка реалізує динамічні масиви. Дозволяє зручно зберігати та керувати змінними наборами даних, наприклад, фрагментами аудіо або параметрами налаштувань [28].
- HardwareSerial.h: Ця бібліотека забезпечує доступ до апаратного послідовного інтерфейсу UART ESP32. Використовується, зокрема, для комунікації з DFPlayer Mini або для виведення налагоджувальної інформації через порт Serial [29].
- DFRobotDFPlayerMini.h: Ця бібліотека використовується для керування аудіомодулем DFPlayer Mini, який дозволяє відтворювати звукові файли (у форматах MP3/WAV) безпосередньо з карти microSD. Вона надає зручний інтерфейс для:
    - відтворення файлів за номером або з папки;
    - паузи, зупинки, перемикання треків;
    - керування гучністю;
    - обробки подій (наприклад, кінець треку).

  Бібліотека спілкується з модулем через UART (послідовний порт) і дозволяє створювати звуковий інтерфейс, наприклад, для озвучення повідомлень користувачеві у голосовій системі [30].

2.2.1.2. Python бібліотеки

Цей підпункт присвячено бібліотекам, які використовуються на сервері для прийому аудіо, обробки мовлення та управління з'єднаннями з клієнтами.

- fastapi: Потужний сучасний фреймворк для створення веб-додатків і API. У даній системі використовується для створення WebSocket-сервера, який приймає аудіодані з ESP32 та повертає відповіді на основі розпізнавання мови [31].
- vosk: Бібліотека для офлайн-розпізнавання мови. Model завантажує мовну модель, а KaldiRecognizer виконує розпізнавання аудіо у режимі реального часу. Використовується для перетворення голосових команд у текст [32].
- json: Стандартна бібліотека Python для роботи з JSON-даними. Застосовується для серіалізації результатів розпізнавання, обміну даними з клієнтом через WebSocket [33].
- wave: Модуль для зчитування та збереження аудіофайлів у форматі WAV. Використовується для зберігання отриманих з ESP32 аудіофрагментів перед подальшою обробкою або відладкою [34].
- os: Стандартна бібліотека для роботи з операційною системою, зокрема з файловою системою. Дозволяє створювати, перейменовувати або видаляти тимчасові файли [35].
- tempfile: Забезпечує створення тимчасових файлів і каталогів, що використовуються для зберігання аудіо на короткий час перед обробкою або надсиланням у зовнішню систему [36].
- openai: Клієнтська бібліотека для доступу до API OpenAI. Використовується, якщо в системі передбачено обробку команд за допомогою мовних моделей OpenAI, наприклад, для приведення фрази користувача до наявного пулу команд [37].
- dotenv (load_dotenv): Бібліотека для зчитування конфігураційних змінних з .env файлу. Дозволяє безпечно зберігати ключі API, налаштування шляху до моделей, конфігурацію порту тощо [38].

- uvicorn: ASGI-сервер, який запускає FastAPI-додаток. Забезпечує обробку WebSocket-з'єднань і HTTP-запитів. Легкий та оптимізований для високої продуктивності у реальному часі [39].
- time: стандартна бібліотека Python для роботи з часом. Дозволяє виконувати затримки (sleep), отримувати поточний час (time), вимірювати інтервали та працювати з часовими мітками у секундах. Широко використовується для керування таймерами, відлагодження та часової синхронізації [40].

2.2.2. Призначення та функціональні можливості модулів

Підрозділ присвячений розділу програми на певні модулі, для зручної модифікації.

Програма апаратної частини складається з п’яти файлів: двох для модуля PlayerController, двох для модуля ElevatorCommands та одного — для основного скетчу.

Модуль PlayerController містить в собі бібліотеку DFRobotDFPlayerMini та реалізує керування аудіоплеєром на базі зовнішнього серійного модуля через апаратний серійний інтерфейс. Він інкапсулює функціональність ініціалізації, запуску відтворення треків, встановлення гучності та контролю стану відтворення.

PlayerController.cpp:

```
#include "PlayerController.h"

PlayerController::PlayerController(HardwareSerial &serial, uint8_t
rxPin, uint8_t txPin)
: serialPort(serial), rx(rxPin), tx(txPin) {}

bool PlayerController::begin() {
 serialPort.begin(9600, SERIAL_8N1, rx, tx);
 delay(500);
 if (!player.begin(serialPort, true, false)) {
   return false;
 }
 player.volume(20);
 if (busyPin != 255) {
   pinMode(busyPin, INPUT);
 }
 return true;
}
```

```
void PlayerController::playTrack(uint16_t trackNumber) {
 while (isPlaying()) {
   delay(100);
 }
 player.playMp3Folder(trackNumber);
 delay(1000);
}

void PlayerController::setVolume(uint8_t volume) {
 if (volume >= 0 && volume <= 30)
 {
   player.volume(volume);
 }
 else {
   player.volume(20);
 }

}
void PlayerController::setBusyPin(uint8_t pin) {
 busyPin = pin;
 pinMode(busyPin, INPUT);
}

bool PlayerController::isPlaying() {
 if (busyPin == 255) return false;
 return digitalRead(busyPin) == LOW;
}
```

1. Конструктор
   - Ініціалізує об'єкт, зберігаючи посилання на серійний порт та номери пінів RX і TX.
   - Використовується для задання апаратного порту, через який буде здійснюватися зв'язок із плеєром.
2. Метод begin()
   - Запускає серійне з'єднання на швидкості 9600 бод.
   - Ініціалізує підключення до модуля плеєра.
   - Встановлює початкову гучність (20).
   - Якщо вказано пін busyPin, ініціалізує його як вхід.
   - Повертає true, якщо ініціалізація успішна, інакше — false.
3. Метод playTrack(uint16_t trackNumber)
   - Очікує завершення поточного відтворення (якщо воно триває).
   - Відтворює трек із папки MP3 за номером trackNumber.
   - Затримка 1000 мс після запуску відтворення — для уникнення накладання команд.

4. Метод setVolume(uint8_t volume)
   - Встановлює гучність відтворення (в діапазоні від 0 до 30).
5. Метод setBusyPin(uint8_t pin)
   - Зберігає номер піна, який використовується для перевірки стану відтворення (через сигнал BUSY).
   - Налаштовує пін як вхід.
6. Метод isPlaying()
   - Перевіряє стан плеєра через пін BUSY.
   - Якщо busyPin не заданий (значення 255), повертає false.
   - Якщо пін активний, повертає true, коли плеєр відтворює трек (рівень LOW на піні BUSY).

Модуль ElevatorCommands реалізує шаблон Команда (Command Pattern) для керування основними діями ліфта. Він дозволяє інкапсулювати окремі команди (відкриття/закриття дверей, перехід на поверх) у вигляді об’єктів, які можна зберігати, передавати та виконувати незалежно від логіки виклику. Таким чином, ці класи можна легко адаптувати під конкретну систему керування ліфтом.

ElevatorCommands.h:

```
#ifndef ELEVATOR_COMMANDS_H
#define ELEVATOR_COMMANDS_H

#include <Arduino.h>

class ElevatorCommand {
public:
 virtual void execute() = 0;
 virtual ~ElevatorCommand() {}
};

class OpenDoorCommand : public ElevatorCommand {
public:
 void execute() override;
};

class CloseDoorCommand : public ElevatorCommand {
```

```
public:
 void execute() override;
};

class GoToFloorCommand : public ElevatorCommand {
private:
 int targetFloor;
public:
 GoToFloorCommand(int floor);
 void execute() override;
};

#endif
```

Базовий клас ElevatorCommand

- Абстрактний клас, який визначає інтерфейс для всіх команд.
- Має чисто віртуальний метод execute(), що реалізується в похідних класах.
- Деструктор віртуальний, що забезпечує коректне знищення об’єктів-нащадків.

Класи OpenDoorCommand, CloseDoorCommand та GoToFloorCommand є нащадками від базового класу ElevatorCommand, і вони реалізують його чисто віртуальну функцію execute().

2.2.3. Підключення та ініціалізація апаратних компонентів

У цьому підрозділі описується початкова конфігурація ключових апаратних модулів системи. Це важливий етап, що забезпечує правильну взаємодію між мікроконтролером ESP32 та підключеними пристроями.

2.2.3.1. Мікрофонний модуль INMP441

Розглядається процес налаштування цифрового мікрофону INMP441, який використовується для захоплення аудіосигналів. Детально описано інтерфейс I2S, параметри вибірки та особливості ініціалізації.

Ініціалізація:

```
// Мікрофон
#define I2S_SCK 18  //зелений
#define I2S_WS 19   //помаранчевий
#define I2S_SD 21   //синій
#define I2S_PORT I2S_NUM_0
```

```
#define bufferLen 512
#define NOISE_THRESHOLD_FACTOR 1.3
#define CONTINUE_SENDING_TIME 1500

int16_t sBuffer[bufferLen];
int16_t sBuffer2[bufferLen];

float baselineNoise = 100;
bool isSpeaking = false;
bool sendEND = false;
unsigned long lastSpeechTime = 0;
// Мікрофон
```

У цьому фрагменті задаються конкретні GPIO, до яких підключено сигнали SCK (Serial Clock), WS (Word Select / LRCLK) та SD (Serial Data). Також визначається, що використовується I2S-порт 0, один із двох доступних в ESP32. GND та L/R піни мікрофону під'єднані до GND піну ESP32. VDD пін – до 3V3 ESP32. Також ініціалізуються константи, змінні, флаги та буфери, які використовуються для запису звуку.

Розпінування модуля INMP441 наведено на рис. 2.3.

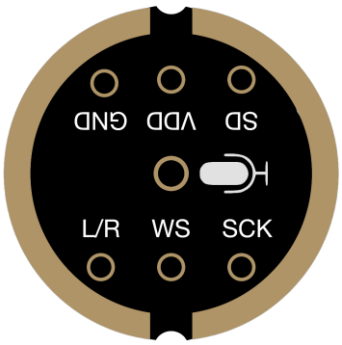


Рис. 2.3 – Розпінування модуля INMP441

```
void i2s_install() {
// Set up I2S Processor configuration
 const i2s_config_t i2s_config = {
      .mode = i2s_mode_t(I2S_MODE_MASTER | I2S_MODE_RX),
      .sample_rate = 16000,
      .bits_per_sample = i2s_bits_per_sample_t(16),
      .channel_format = I2S_CHANNEL_FMT_ONLY_LEFT,
      .communication_format =
      i2s_comm_format_t(I2S_COMM_FORMAT_STAND_I2S),
      .intr_alloc_flags = 0,
      .dma_buf_count = 8,
      .dma_buf_len = bufferLen,
      .use_apll = false
      };

i2s_driver_install(I2S_PORT, &i2s_config, 0, NULL);
}
```

Ця функція ініціалізує драйвер I2S, встановлюючи основні параметри:

- I2S_MODE_MASTER | I2S_MODE_RX: ESP32 працює як майстер і тільки зчитує (RX).
- sample_rate = 16000: частота дискретизації — 16 кГц, достатньо для розпізнавання мови.
- bits_per_sample = 16: 16-бітний аудіопотік — типове представлення для PCM.
- I2S_CHANNEL_FMT_ONLY_LEFT: використовується лише лівий канал (моно).
- I2S_COMM_FORMAT_STAND_I2S: стандартний I2S протокол.
- dma_buf_count = 8 та dma_buf_len = bufferLen: налаштування буфера DMA — визначає обсяг тимчасової пам'яті для обробки даних без перевантаження CPU.
- use_apll = false: не використовує аудіо PLL (фазову автопідстройку частоти).

Після цього викликається i2s_driver_install() — це реєструє та активує драйвер I2S з заданими параметрами.

```
void i2s_setpin() {
// Set I2S pin configuration
const i2s_pin_config_t pin_config = {
      .bck_io_num = I2S_SCK,
      .ws_io_num = I2S_WS,
      .data_out_num = -1,
      .data_in_num = I2S_SD
      };

i2s_set_pin(I2S_PORT, &pin_config);
}
```

Ця функція встановлює пін-контроль для I2S:

- bck_io_num = I2S_SCK: битовий тактовий сигнал (BCLK).
- ws_io_num = I2S_WS: сигнал вибору слова (LRCLK).
- data_out_num = -1: передача не використовується (тільки RX).

- data_in_num = I2S_SD: вхідні аудіо-дані (із мікрофона INMP441).

Потім i2s_set_pin() зв’язує ці піни з відповідним портом I2S.

2.2.3.2. DFPlayer Mini

У цьому пункті описано підключення та ініціалізацію аудіомодуля DFPlayer Mini, що забезпечує відтворення голосових фраз, збережених на microSD-карті. Модуль використовується для голосового зворотного зв’язку з користувачем.

Для керування аудіомодулем, як було зазначено раніше в підрозділі «3.2 Призначення та функціонал модулів», використовується модуль PlayerController, який складається з двох файлів: PlayerController.cpp та PlayerController.h.

В основному скетчі відбувається ініціалізація:

```
#define PIN_MP3_RX 16  // RX ESP32 ← TX DFPlayer
#define PIN_MP3_TX 17  // TX ESP32 → RX DFPlayer
HardwareSerial dfSerial(1);
PlayerController myPlayer(dfSerial, PIN_MP3_RX, PIN_MP3_TX);
```

В функції setup() відбувається запуск:

```
myPlayer.setBusyPin(4);
myPlayer.begin();
```

Отже, тут визначаються піни для з’єднання з DFPlayer Mini:

- PIN_MP3_RX = 16 — пін прийому даних (RX) на ESP32, до якого підключено TX вихід DFPlayer.
- PIN_MP3_TX = 17 — пін передачі даних (TX) на ESP32, що йде на RX DFPlayer.

Створюється апаратний серійний порт UART1. Ініціалізує об'єкт керування DFPlayer Mini через зручний інтерфейс класу PlayerController. Потім в setup() обирається пін, який буде приєднано до BUSY DFPlayer Mini.

Розпінування модуля DFPlayer Mini наведено на рис. 2.4. Залишається підключити лише чотири контакти: VCC до VIN на ESP32, GND до GND ESP32, а також SPK+ і SPK– до динаміка.

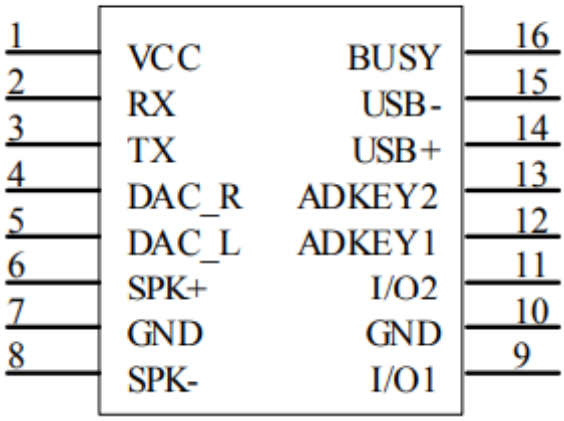


Рис. 2.4 – Розпінування модуля DFPlayer Mini

2.2.4. Ініціалізація та запуск серверу

Цей розділ описує процес запуску серверної частини системи, яка розгорнута в хмарному середовищі AWS EC2. Сервер відповідає за прийом аудіоданих по WebSocket, їх обробку та передачу результатів назад до ESP32.

AWS EC2 (Amazon Web Services Elastic Compute Cloud) — це сервіс хмарних віртуальних серверів, який дозволяє запускати обчислювальні ресурси в інфраструктурі Amazon. Простіше кажучи, EC2 — це "віртуальний комп’ютер у хмарі", який можна використовувати для запуску сайтів, серверів, обробки даних, машинного навчання тощо.

Для мінімального запуску серверу необхідні 1ГБ оперативної пам’яті та 1ГБ файл підкачки (swap). Такий обсяг оперативної пам’яті надається у пакеті AWS EC2 t2.micro. Цей пакет є безкоштовним, що робить його придатним для створення прототипу, тому ми оберемо саме його. Також була обрана операційна система LINUX.

Після встановлення Python, необхідних бібліотек та створення віртуального середовища для сервера, його можна запустити за допомогою команди:

uvicorn server.main:app --host 0.0.0.0 --port 8000 --ws websockets --ws-ping-interval 30 --ws-ping-timeout 10

Ця команда запускає FastAPI-сервер, що знаходиться у файлі main.py в модулі server. Сервер працює на всіх мережевих інтерфейсах пристрою (IP 0.0.0.0) на порту 8000. Для підтримки WebSocket-з'єднань використовується бібліотека websockets. Сервер кожні 30 секунд надсилає ping повідомлення для перевірки активності з’єднання, і, якщо протягом 10 секунд не отримує відповіді, вважає з'єднання втраченим. Такий сервер використовується для обробки аудіоданих, що надходять від ESP32 через WebSocket, у системі голосового управління ліфтом.

2.2.5. Налаштування системи

Тут описано механізм локального налаштування системи користувачем через точку доступу ESP32.

Перед підключенням до Інтернету ESP32 створює точку доступу з веб-інтерфейсом. Точка доступу має базовий логін «ESP32_AP» та пароль «12345688». Ця точка доступу підтримується ввімкненою ще 5 хвилин від того моменту, як останній пристрій відключився від неї. Підключившись до цієї точки доступу можна проводити налаштування, ввівши в браузері адресу «192.168.4.1». Ці налаштування має проводити ліфтер. Усі налаштування зберігаються в енергонезалежній пам’яті ESP32 та завантажуються під час запуску ESP32.

На рисунках 2.5, 2.6, 2.7 зазначені форми, які мають бути заповнені при першому запуску системи.

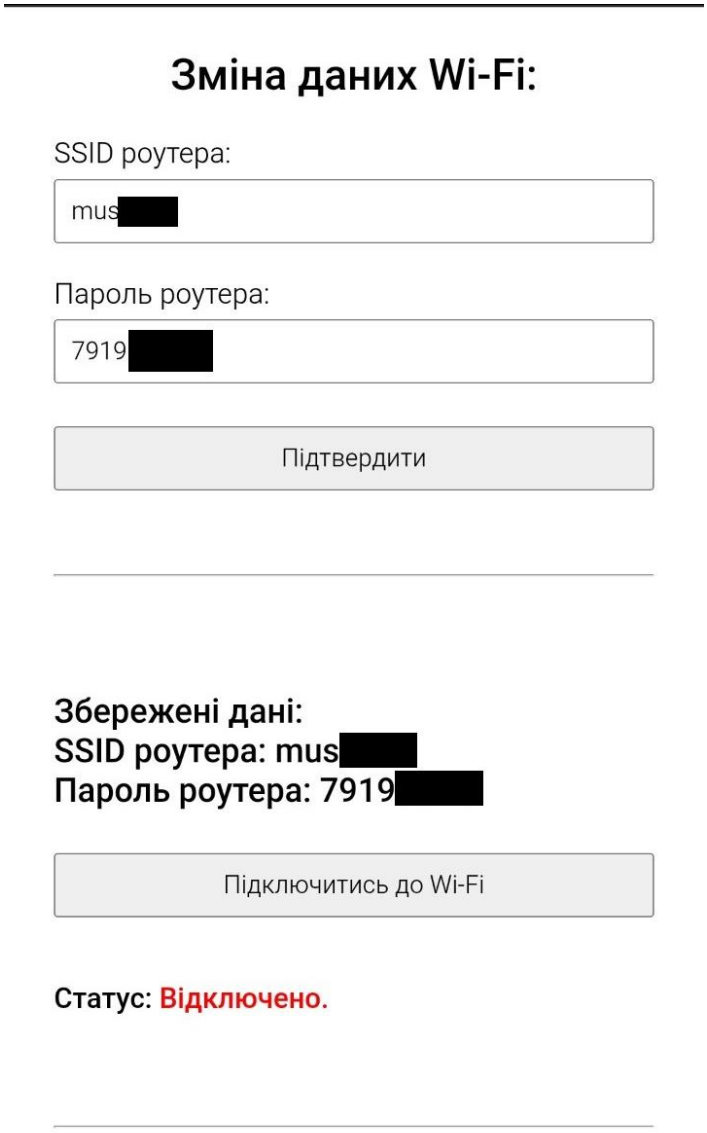


Рис. 2.5 – Перша форма та тестова спроба підключитись до WI-FI

Перша форма призначена для введення SSID та пароля маршрутизатора, який має доступ до мережі Internet. Під цією формою розташована кнопка «Підключитись до Wi-Fi» та рядок зі статусом підключення. Після введення даних до форми можна спробувати підключитися до зазначеного маршрутизатора. У рядку статусу відображається текстова та кольорова індикація поточної стадії підключення. Спроби підключення виконуватимуться кожну хвилину — доти, доки система буде активною.

Рис. 2.6 – Друга форма

Друга форма відповідає за зміну параметрів точки доступу, яка створюється на ESP32. Зміни набувають чинності під час наступного запуску ESP32. Оскільки ця точка доступу автоматично закривається через 5 хвилин після виходу або за натисканням спеціальної кнопки (про неї буде сказано далі), ніхто, крім ліфтера, не зможе змінити налаштування у його відсутності.

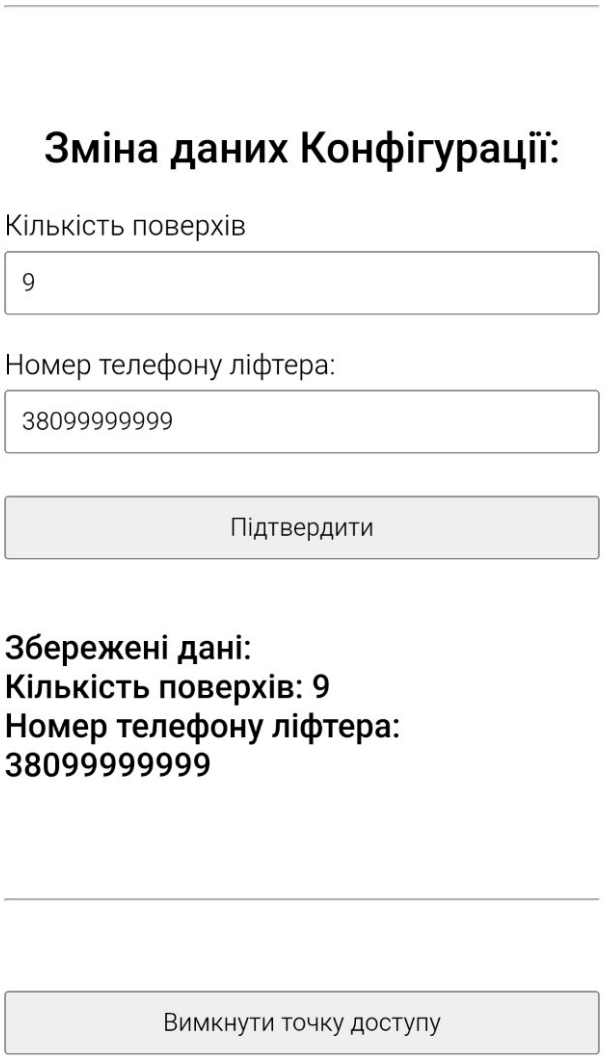


Рис. 2.7 – Третя форма та кнопка «Вимкнути точку доступу»

Третя форма відповідає за авторизацію на сервері. У ній задається конфігурація будинку (кількість поверхів) та номер телефону ліфтера, відповідального за цю систему. Без заповнення цих полів з’єднання з сервером є неможливим.

Кнопка «Вимкнути точку доступу» призначена для запуску системи, коли проведені усі налаштування. Також вимикається точка доступу до наступного перезапуску ESP32.

2.2.6. Ініціалізація точки доступу на ESP32

У підрозділі детально розглянемо, як ініціалізується точка доступу з WebServer та як обробляються POST та GET запити.

```
WebServer server(80);
```

Цей рядок створює екземпляр вбудованого HTTP-сервера, який буде працювати на ESP32 і слухати порт 80 — стандартний порт для веб-запитів (HTTP).

Після цього в setup() створюються endpoints. Це дозволяє організувати просту взаємодію користувача з ESP через веб-інтерфейс.

1. server.on("/", handleRoot);
   - Обробка головної сторінки (/).
   - Викликає функцію handleRoot, яка відправляє HTML-сторінку налаштувань.
2. server.on("/submit", HTTP_POST, handleSubmit);
   - Обробка форми, яка надсилається методом POST на адресу /submit.
   - Викликає функцію handleSubmit, що зчитує надіслані дані (SSID/пароль Wi-Fi маршрутизатора).
3. server.on("/connect", HTTP_GET, []() {...});
   - При GET-запиті на /connect викликає:
   - connectToWiFi(); — спроба підключитись до Wi-Fi.
   - checkWiFiStatus(); — перевірка результату.
   - Відправляє статус з’єднання у відповідь (connectionStatus).
4. server.on("/status", HTTP_GET, []() {...});
   - При запиті на /status віддає поточний статус підключення (connectionStatus).
5. server.on("/submitAP", HTTP_POST, []() {...});
   - Обробка форми налаштування точки доступу (AP).

- Викликає handleAPSubmit() для зміни та збереження налаштувань AP.

6. server.on("/submitCONFIG", HTTP_POST, []() {...});
   - Обробка форми конфігурації.
   - Викликає handleCONFIGSubmit() для зміни та збереження налаштувань.
7. server.on("/END", HTTP_POST, []() {...});
   - Сигнал на завершення роботи точки доступу.
   - Викликає stopAP() — вимикає AP та веб-сервер.

2.2.7. Авторизація ESP32 на сервері

У підрозділі розглянемо яким чином відбувається авторизація мікроконтролера ESP32 на сервері AWS EC2.

```
void webSocketEvent(WStype_t type, uint8_t* payload, size_t length) {
  switch (type) {
    case WStype_CONNECTED:
      {
        msgToCom("WebSocket: Підключено.");
        isWebSocketConnected = true;
        StaticJsonDocument<200> jsonDoc;
        jsonDoc["floors"] = floors;
        jsonDoc["number"] = number;

        String jsonString;
        serializeJson(jsonDoc, jsonString);
        client.sendTXT(jsonString);
        break;
      }
    ...
  }
}
```

У відповідь на підключення до WebSocket-сервера:

1. Встановлюється флаг isWebSocketConnected = true, що сигналізує про успішне з'єднання.
2. Формується JSON-документ jsonDoc, який містить два ключові поля:
   - "floors" — кількість поверхів, які підтримує даний ліфт.

- "number" — номер телефону ліфтера.

3. Після серіалізації JSON-документу у форматі рядка (jsonString), він надсилається серверу за допомогою client.sendTXT(jsonString).

2.2.8. Збір та передача даних

У цьому підрозділі описано, як мікроконтролер ESP32 здійснює збір аудіосигналу, його попередню обробку та передачу даних на сервер через WebSocket-з'єднання.

Мікрофон, підключений до ESP32 через інтерфейс I2S, забезпечує зчитування аудіосигналу у вигляді дискретних значень (зразків). Основна обробка здійснюється у функції listening(), яка викликається періодично в основному циклі програми. Її логіка подана нижче:

```
void listening() {
  size_t bytes_read = 0;
  float sum = 0;

  memcpy(sBuffer, sBuffer2, bufferLen * sizeof(int16_t));

  if (i2s_read(I2S_PORT, sBuffer2, sizeof(sBuffer2), &bytes_read,
portMAX_DELAY) == ESP_OK) {
    size_t samples_read = bytes_read / sizeof(int16_t);

    for (size_t i = 0; i < samples_read; ++i) {
      sum += abs(sBuffer[i]);
    }

    float mean = sum / samples_read;
    float threshold = baselineNoise * NOISE_THRESHOLD_FACTOR;

    if (mean > threshold) {
      if (!isSpeaking) {
        msgToCom("Людина почала розмовляти!");
        isSpeaking = true;
      }
      lastSpeechTime = millis();
    } else if (isSpeaking && millis() - lastSpeechTime >
CONTINUE_SENDING_TIME) {
      msgToCom("Людина замовкла...");
      isSpeaking = false;
      sendEND = true;
    }

    // Якщо людина говорить або ще не завершився час очікування,
передаємо дані
    if (isSpeaking || millis() - lastSpeechTime <=
CONTINUE_SENDING_TIME) {
      sendAudioDataToWebSocket(sBuffer, bytes_read);
```

```
    }

    if (sendEND && millis() - lastSpeechTime > CONTINUE_SENDING_TIME) {
      sendEND = false;
      msgToCom("Надсилаю END");
      client.sendTXT("__END__");
    }
  }

}
```

Основні етапи обробки:

- Зчитування аудіо: За допомогою i2s_read() ESP32 отримує чергову порцію аудіоданих у буфер sBuffer2. Попередній буфер sBuffer копіюється з sBuffer2 для забезпечення безперервності потоку.
- Оцінка активності мови: Обчислюється середнє значення абсолютної амплітуди сигналу (mean). Якщо воно перевищує заздалегідь визначений поріг шуму (threshold), система розпізнає, що людина почала говорити. В іншому випадку, якщо мовлення припиняється на заданий інтервал часу (CONTINUE_SENDING_TIME), система визначає, що голосовий фрагмент завершено.
- Передача аудіоданих: У випадку активного мовлення або короткої паузи між словами аудіобуфер передається на сервер функцією sendAudioDataToWebSocket(...).
- Завершення передачі: Після визначення кінця мовлення надсилається службове повідомлення "__END__", яке сигналізує серверу про завершення потоку аудіо.

2.2.9. Список дозволених команд та збір їх до черги

У цьому підрозділі розглянемо механізм обробки розпізнаних голосових команд, перевірки їх на відповідність дозволеному списку та організації їх виконання через чергу.

Для забезпечення безпеки та стабільності роботи ліфтової системи, набір допустимих голосових команд обмежений попередньо визначеним переліком. До нього входять як загальні команди керування, так і команди виклику на конкретний поверх:

- «скинути команди» — очищає чергу всіх збережених команд;
- «список команд» — викликає відтворення списку дозволених команд;
- «поїхали» — ініціює виконання накопичених команд;
- «відчинити двері» — відкриває двері ліфта;
- «зачинити двері» — зачиняє двері ліфта;
- «викликати ліфтера» — надсилає відповідне повідомлення до сервісної служби;
- «[N]-й поверх», де N — число в діапазоні від 1 до максимальної кількості поверхів, яка задається параметром floors під час авторизації ESP32 на сервері.

До черги додаються виключно команди виду «[N]-й поверх». Це зроблено навмисно, щоб у ліфті кілька пасажирів могли одночасно назвати свої поверхи, не чекаючи завершення попередніх дій або прибуття ліфта. Такий підхід дозволяє мінімізувати затримки та зробити взаємодію більш природною, як при натисканні кнопок у звичайному ліфті. Ліміт черги становить 8 команд. Після чого вони не будуть додаватись.

Команди з черги не виконуються одразу — вони накопичуються до моменту отримання спеціальної команди «поїхали», яка запускає їх обробку. Після цього система виконує підйом/спуск на відповідні поверхи у заданому порядку.

Висновок

Отже, в даному розділі було розглянуто складові частини системи голосового керування ліфтом, а також їх взаємодію між собою.

Було описано ключові апаратні модулі, зокрема мікроконтролер ESP32, мікрофон INMP441 та аудіомодуль DFPlayer Mini, а також наведено схеми підключення та принципи їх ініціалізації. Розглянуто використані програмні бібліотеки на платформах Arduino та Python, що забезпечують збір, обробку та передавання аудіоданих.

Також було проаналізовано етапи запуску системи, починаючи з створення точки доступу на ESP32, перемикання в режим клієнта, авторизації на сервері через WebSocket та передачі аудіопотоку. Окрему увагу приділено логіці роботи черги команд, яка дозволяє накопичувати запити на певні поверхи, що підвищує ефективність використання ліфта.

Таким чином, цей розділ заклав технічну основу системи та продемонстрував її здатність забезпечувати повноцінну взаємодію між користувачем, апаратною частиною та серверною логікою, реалізуючи зручне голосове управління ліфтом.

3. АЛГОРИТМИ ФУНКЦІОНУВАННЯ СИСТЕМИ

3.1. Алгоритм взаємодії користувача з системою

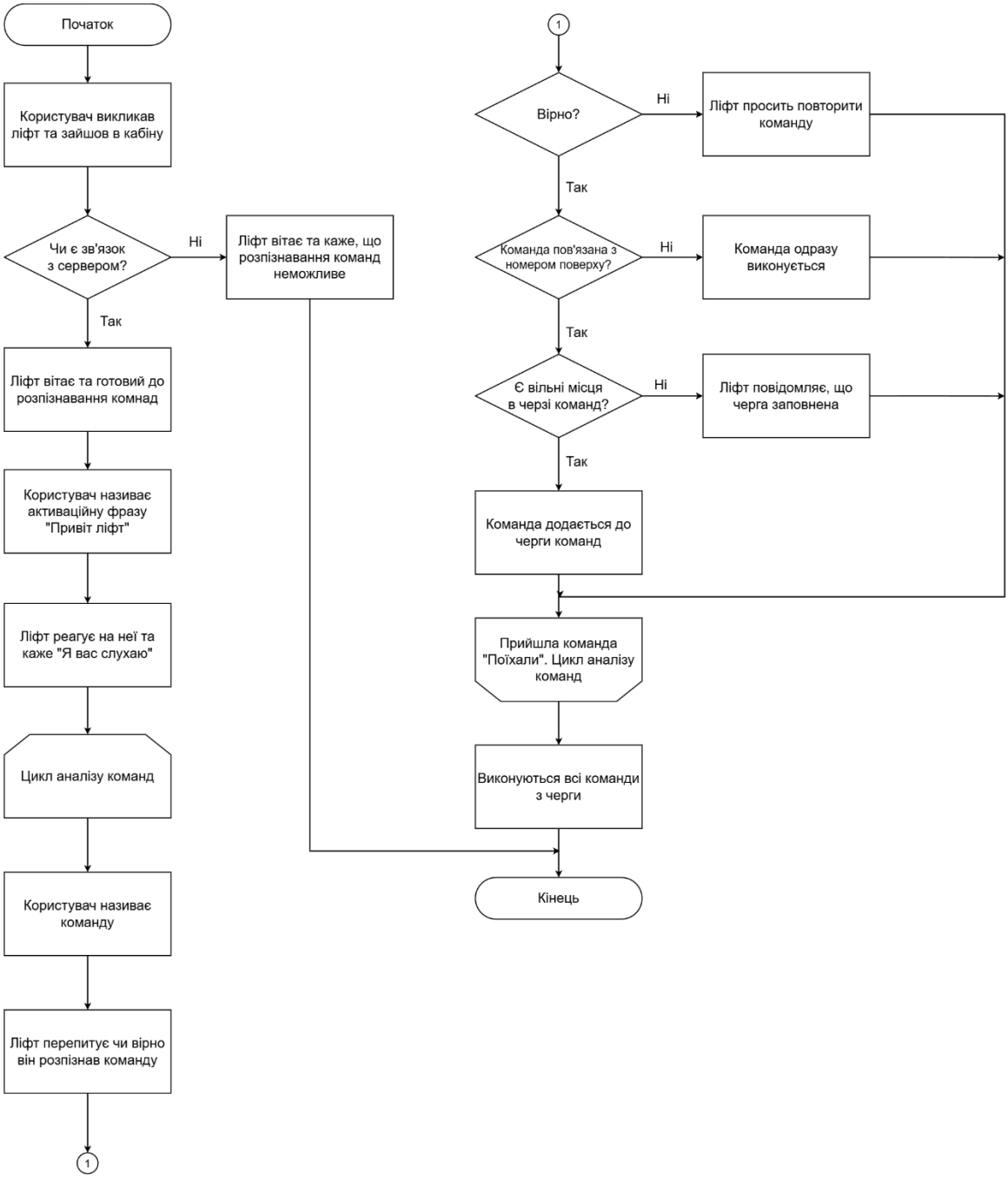


Рис. 3.1 – Алгоритм взаємодії користувача з системою

На рис. 3.1 зображено блок-схему послідовних дій користувача при голосовій взаємодії з системою управління ліфтом. Алгоритм починається з

моменту, коли користувач викликає ліфт та заходить у кабіну. Подальша логіка системи залежить від наявності зв’язку із сервером:

- Якщо зв’язок із сервером відсутній, система інформує користувача, що розпізнавання команд неможливе.
- Якщо зв’язок встановлено, ліфт переходить у режим очікування голосових команд.

Користувач промовляє активаційну фразу ("Привіт ліфт"), на яку система відповідає готовністю слухати — фразою "Я вас слухаю".

Після цього користувач диктує команду. Система розпізнає її та перепитує, чи правильно вона була розпізнана. Якщо команда неправильна — ліфт просить користувача повторити її.

У разі вірного розпізнавання система перевіряє тип команди:

- Якщо це команда, не пов’язана з поверхами (наприклад, "Поїхали") — вона виконується одразу.
- Якщо це команда з указанням номера поверху, система перевіряє, чи є в черзі місце для додавання команди:
  - Якщо є — команда додається до черги та інформує про це користувача.
  - Якщо черга переповнена — ліфт інформує про це користувача.

Таким чином, алгоритм забезпечує інтерактивний, адаптивний і користувач-орієнтований підхід до управління ліфтом за допомогою голосу, з урахуванням перевірки точності розпізнавання та обробки можливих помилок.

3.2. Алгоритм роботи апаратної частини

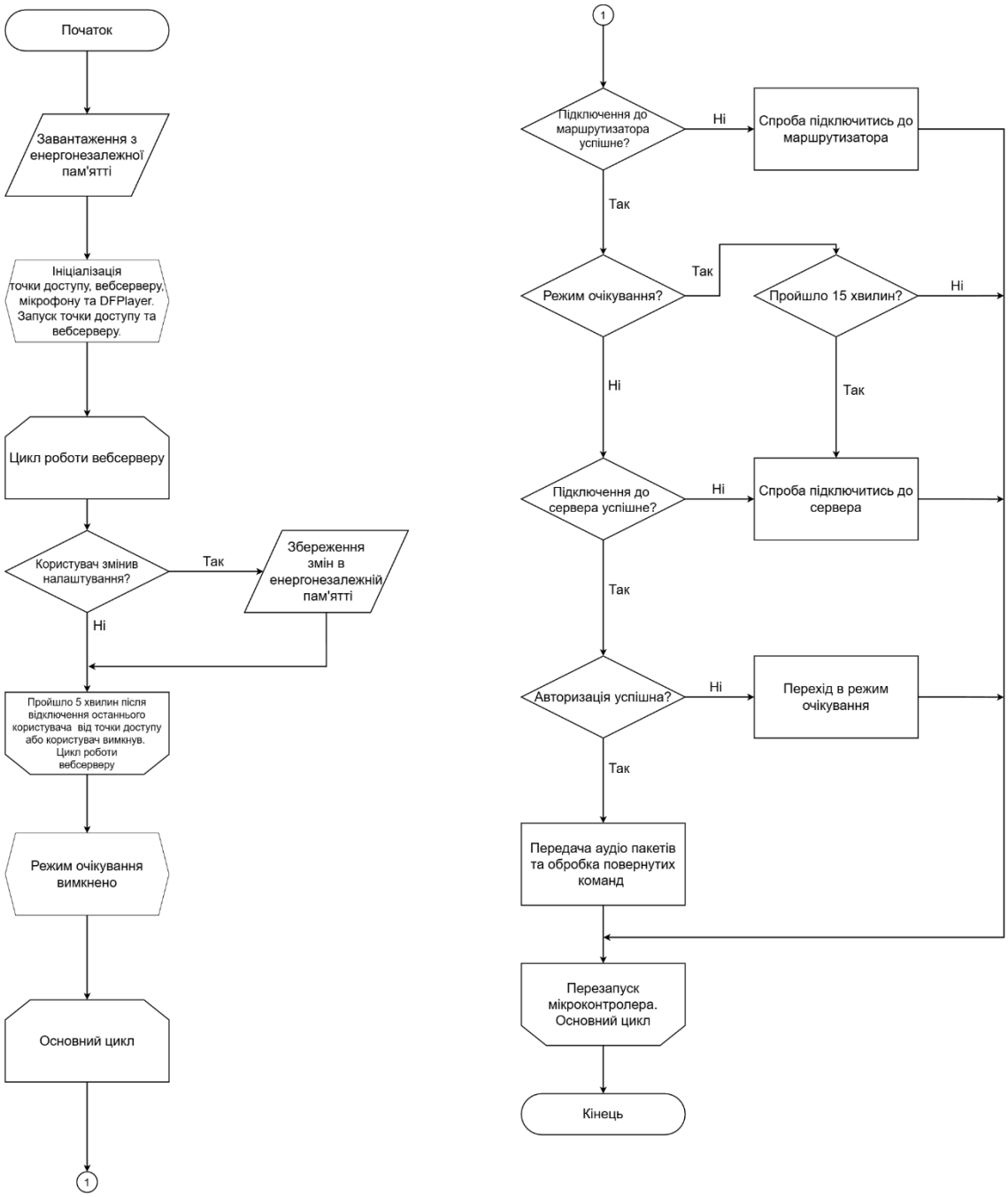


Рис. 3.2 – Алгоритм роботи апаратної частини

Після запуску мікроконтролера ініціалізується послідовний порт і відбувається затримка, щоб забезпечити стабільний старт. Далі з енергонезалежної пам’яті зчитуються збережені налаштування: ім’я та пароль точки доступу, цільова мережа Wi-Fi, кількість поверхів та номер телефону. Якщо SSID і пароль точки доступу не знайдені — встановлюються значення за замовчуванням.

Налаштовується вебсервер: обробники кореневої сторінки, прийому налаштувань, перевірки статусу з’єднання, а також спеціальних запитів, наприклад, зупинки точки доступу. Запускається точка доступу.

Далі ініціалізується мікрофон через I2S, DFPlayer, і перевіряється його готовність. Якщо плеєр не ініціалізувався — встановлюється код помилки.

Основна логіка відбувається у функції loop(). Якщо точка доступу активна — перевіряється кількість підключених клієнтів. Якщо кількість змінюється, надсилається повідомлення. Якщо протягом певного часу ніхто не підключався — точка доступу вимикається.

Якщо ж точка доступу неактивна і інтернет-з’єднання відсутнє, то після кількох невдалих спроб або при першій спробі підключення здійснюється нова спроба з’єднатися з роутером.

Якщо з’єднання з роутером є, але WebSocket-з’єднання ще не встановлено — періодично пробується перепідключення. Після шести невдалих спроб система перезавантажується.

Якщо WebSocket-з’єднання встановлено — перевіряється активність. У разі бездіяльності з боку сервера зв’язок скидається, встановлюється таймер для перепідключення і система повідомляє про втрату зв’язку. Якщо DFPlayer відтворює нічого, відбувається прослуховування мови.

Окремо періодично перевіряється статус підключення до маршрутизатора, якщо була активована відповідна перевірка.

Ця система організована як цикл перевірок і обробок, що працює постійно, реагуючи на зміни станів з’єднань, активність користувача і події, отримані із зовнішнього світу через WebSocket.

3.3. Алгоритм роботи серверної частини

Тут детально розглянемо роботу серверної частини проєкту.

Система працює в кілька етапів і управляється станами (режимами), як зображено на блок-схемі (рис. 3.3).

На початку ESP32 надсилає авторизаційні дані у форматі JSON. Якщо дані валідні (має бути номер телефону та список поверхів), сервер відповідає повідомленням про успішну авторизацію і повідомляє оператора ліфта. Якщо авторизація не проходить, з'єднання закривається.

Після авторизації сервер починає приймати аудіодані від пристрою. Весь звук записується у тимчасовий .wav файл. Як тільки ESP32 надсилає команду завершити передачу (__END__), сервер обробляє аудіо, розпізнає мову за допомогою Vosk і отримує текстову інтерпретацію сказаного.

Обробка команди залежить від поточного режиму. У режимі "activation" система чекає на активаційну фразу "привіт ліфт". Якщо така фраза виявлена, режим перемикається на "command", і сервер повідомляє про це клієнту.

У режимі "command" розпізнаний текст надсилається до LLM (мовної моделі), яка визначає зміст команди користувача — наприклад, до якого поверху потрібно їхати. Якщо команду не вдалося розпізнати, сервер надсилає повідомлення про помилку. У разі успішної інтерпретації сервер надсилає команду назад ESP32 і перемикається в режим "acknowledgement".

У режимі "acknowledgement" система очікує на відповідь користувача — підтвердження чи заперечення. Якщо користувач відповідає "так", надсилається відповідь __YES__ і система повертається до режиму команд. Якщо відповідь "ні", надсилається __NO__, і також повернення до попереднього етапу. Якщо ж відповідь не розпізнано, сервер надсилає повідомлення про помилку підтвердження.

У будь-який момент система також реагує на спеціальні текстові команди типу __START__ чи __RUN__, які скидають режим на початковий — "activation". Якщо виникає помилка DFPlayer або розривається з’єднання, сервер припиняє роботу з клієнтом і повідомляє оператора.

Усе аудіо обробляється поетапно, зберігається у тимчасових файлах і після завершення обробки очищується. Робота побудована на принципі станів, де кожен наступний крок залежить від результату попереднього, забезпечуючи плавну голосову взаємодію з системою.

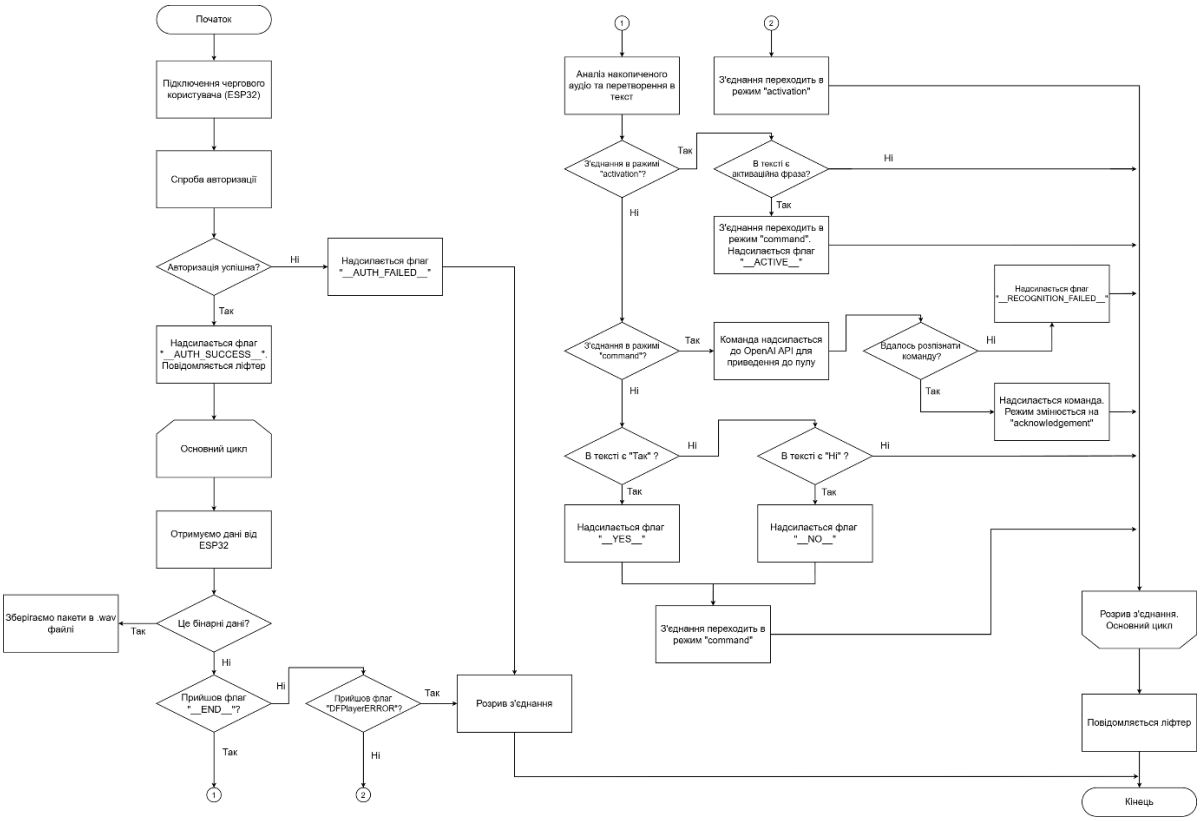


Рис. 3.3 – Алгоритм роботи серверної частини

Висновок

Отже, у цьому розділі було розглянуто ключові алгоритми, що забезпечують функціонування інтерактивної системи керування ліфтом на основі голосових команд. Детальний аналіз охопив три взаємопов’язані компоненти: алгоритм взаємодії користувача з системою, логіку роботи апаратної частини та алгоритм функціонування серверної частини.

Алгоритм взаємодії з користувачем демонструє, як голосове управління може бути реалізоване у вигляді послідовного, адаптивного процесу із врахуванням точності розпізнавання команд, наявності зв’язку з сервером, а

також обробки помилок і ситуацій перевантаження. Це забезпечує високий рівень зручності та інформованості для користувача.

Апаратна частина системи включає ініціалізацію основних модулів, а також керування точкою доступу, з’єднанням із мережею та обміном даними. Алгоритм враховує стан активності клієнтів і стан підключення, що дозволяє ефективно перемикатися між режимами роботи.

Серверна частина виконує обробку WebSocket-з’єднання, аутентифікацію, приймання аудіо та передачу команд назад на пристрій. Реалізовано механізми повторного з’єднання, контролю активності, черговості команд і обробки помилок, що забезпечує стабільність системи навіть у випадку короткочасних збоїв.

Загалом, алгоритми, описані у цьому розділі, демонструють високий ступінь інтеграції між користувацьким інтерфейсом, апаратними засобами та серверною логікою. Така структура дозволяє забезпечити надійність, масштабованість та інтуїтивну зручність експлуатації системи, що є критично важливим для її реального впровадження в інтелектуальне середовище.

4. ТЕСТУВАННЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ СИСТЕМИ

4.1. Тестування роботи веб-сайту налаштувань

Тут розглянемо реакцію веб-сайту налаштувань на заповнення та відправлення форм. Усі ці виводи до COM-порту відбуваються тільки в режимі дебагу (для демонстрації роботи системи).

За режим дебагу відповідає флаг:

```
bool isDebug = true;
```

Коректний результат заповнення першої форми (WI-FI), відправлення до ESP32, збереження в енергонезалежній пам’яті та виводу до монітору порту можна побачити на рис. 4.1.

```
Дані збережені!
Оновлений targetSSID: mus
Оновлений targetPsw: 7919
```

Рис. 4.1 – Результат відправлення форми «WI-FI»

Коректний результат спроби підключитись до WI-FI маршрутизатора можна побачити на рис. 4.1. Тут видно, що підключення відбувається не одразу і кожні 5 секунд виводиться статус підключення. Цей статус оновлюється в реальному часі на сайті налаштувань. Це можна подивитись на рис. 4.3 та рис. 4.4.

```
WIFI_STATUS: Підключення...
WIFI_STATUS: Підключення...
WIFI_STATUS: Підключено!
WIFI_STATUS: Підключено!
```

Рис. 4.2 – Результат тестового підключення до WI-FI

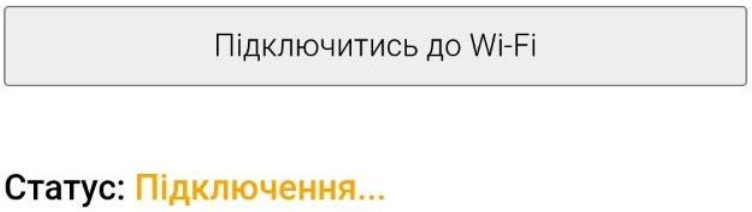


Рис. 4.3 – Зміна статусу підключення на «Підключення…»

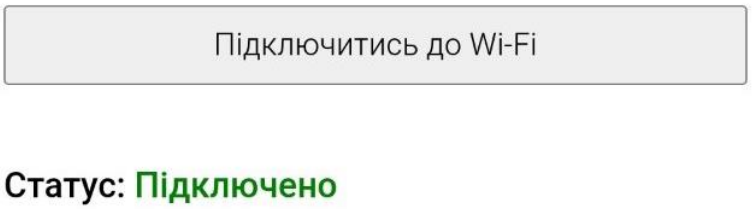


Рис. 4.4 – Зміна статусу підключення на «Підключено»

Коректний результат заповнення другої форми (Точка доступу), відправлення до ESP32, збереження в енергонезалежній пам’яті та виводу до монітору порту можна побачити на рис. 4.5.

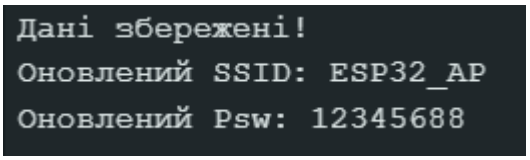


Рис. 4.5 – Результат відправлення форми «Точки доступу»

Коректний результат заповнення третьої форми (Конфігурація), відправлення до ESP32, збереження в енергонезалежній пам’яті та виводу до монітору порту можна побачити на рис. 4.6.

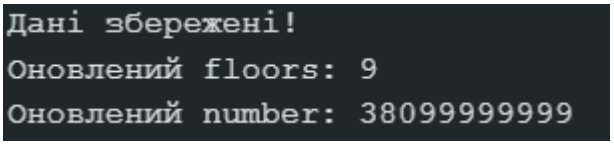


Рис. 4.6 – Результат відправлення форми «Конфігурація»

Коректний результат передчасного вимкнення точки доступу можна побачити на рис. 4.7.

Рис. 4.7 – Результат вимкнення точки доступу через сайт

4.2. Тестування випадків некоректного вводу даних налаштування

Форми працюють незалежно одна від одної. Для того щоб надіслати та зберегти дані в енергонезалежній пам’яті, необхідно, щоб кожне поле певної форми було заповнене (це проілюстровано на рис. 4.8) або, у випадку з паролями, містило щонайменше 8 символів (це проілюстровано на рис. 4.9).

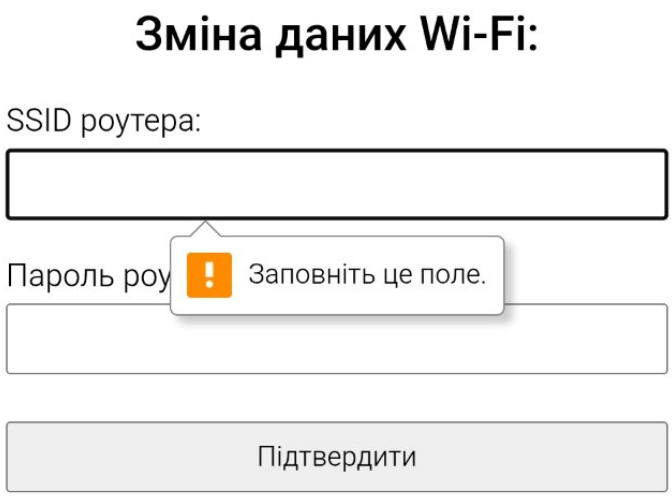


Рис. 4.8 – Пусте поле

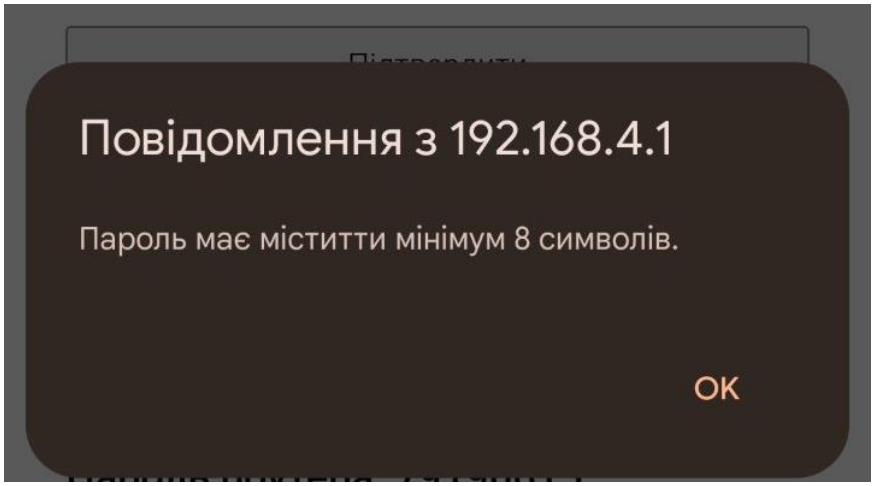


Рис. 4.9 – Пароль коротше за 8 символів

В конфігураційній формі кількість поверхів може бути в межах від 1 до 100 включно. Це показано на рис. 4.10 та рис. 4.11. А номер телефону має бути довжиною 12 символів рівно (рис. 4.12).

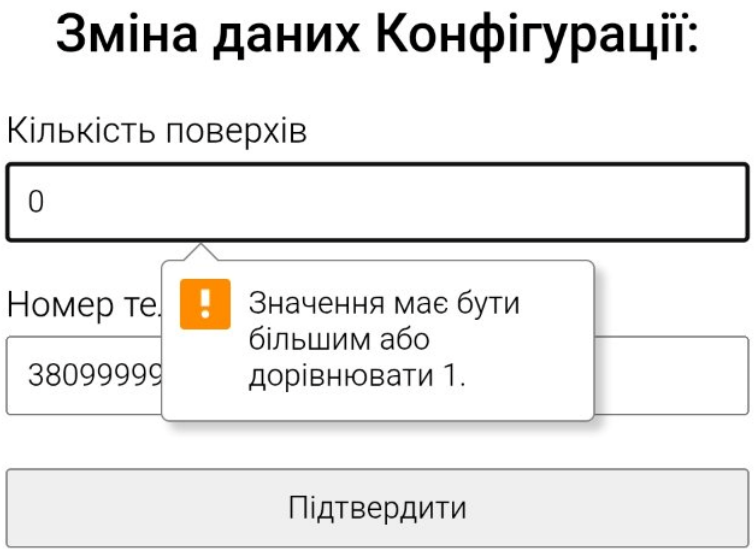


Рис. 4.10 – Кількість поверхів не може бути менше 1

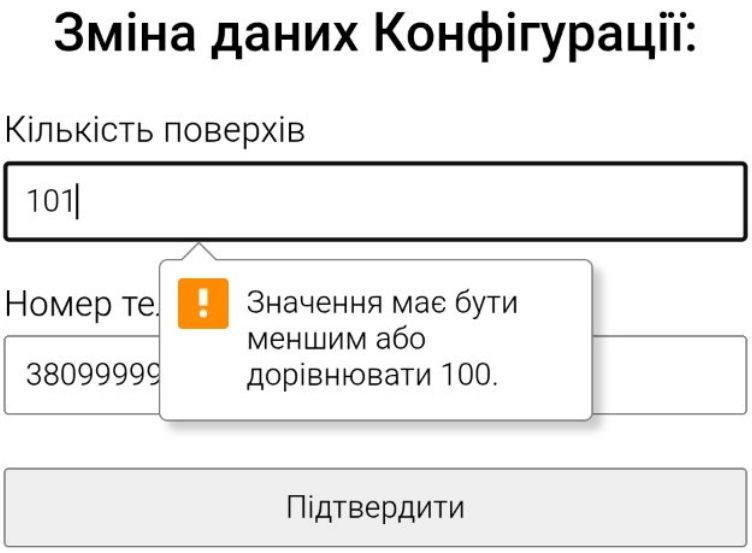


Рис. 4.11 – Кількість поверхів не може бути більше 100

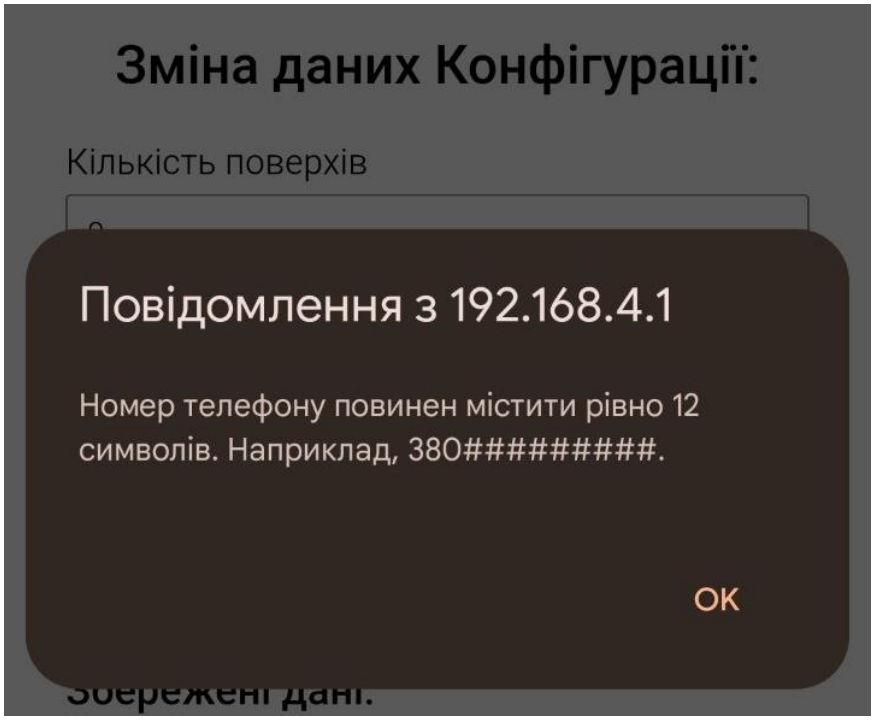


Рис. 4.12 – Номер телефону має бути довжиною 12 символів

Також варто приділити увагу випадку, коли користувач ввів неправильні дані для підключення до WI-FI та спробував під’єднатись. Як видно з рис. 4.13, було виконано 12 спроб підключення, на 13-у статус змінився на «Відключено.». На рис. 4.15 видно як відображається невдале підключення. Якщо вимкнути точку доступу, то ESP32 буде знову ж таки робити 12 спроб підключення, а на 13-у починати новий набір з 12-ти спроб (рис. 4.14).

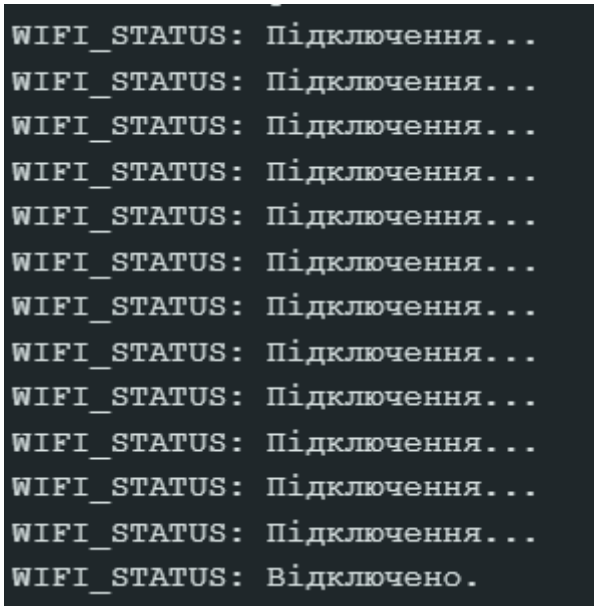


Рис. 4.13 – Випадок з невірними даними підключення до маршрутизатора

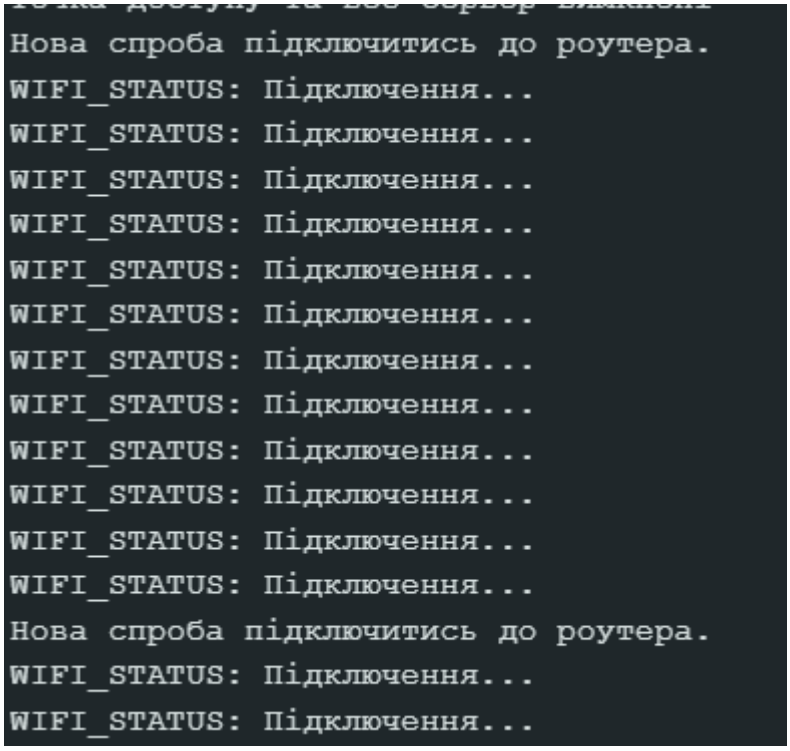


Рис. 4.14 – Випадок з невірними даними підключення до маршрутизатора після вимкнення точки доступу

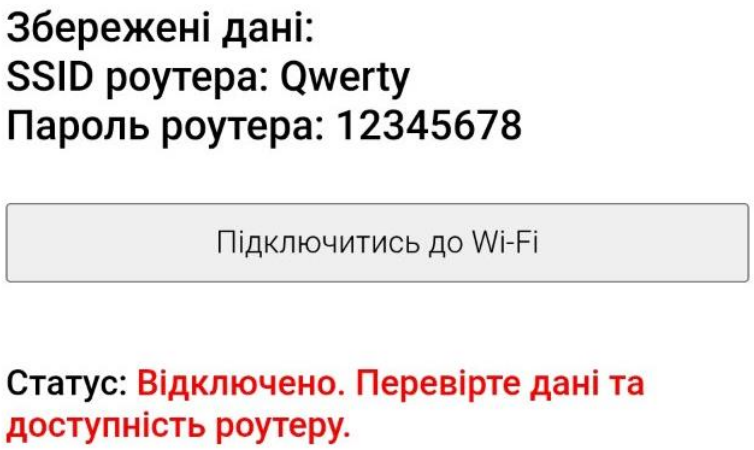


Рис. 4.15 – Зміна статусу на «Відключено. Перевірте дані та доступність роутеру»

Розглянемо випадок спроби підключення до сервера через WebSocket, якщо були введені неправильні конфігураційні дані (рис. 4.16). Як видно з

рисунка, сервер повернув «__AUTH_FAILED», що свідчить про невдалу авторизацію. Після цього WebSocket-з'єднання закривається, а ESP32 переходить у режим очікування та намагається перепідключитися кожні 15 хвилин замість 15 секунд.

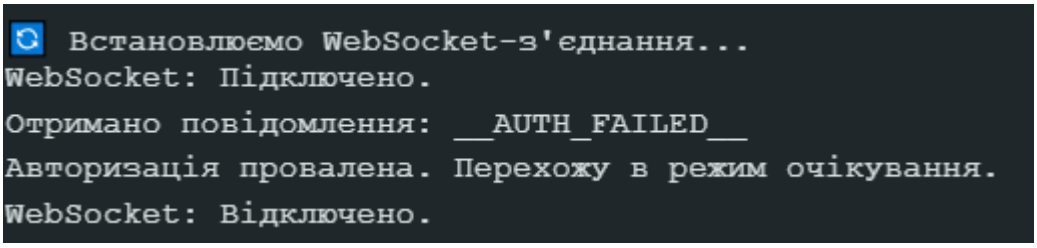


Рис. 4.16 – Випадок з неправильними конфігураційними даними

4.3. Тестування реакції на розрив WebSocket-з'єднання

Якщо розрив з'єднання відбудеться через серверу, то ESP32 одразу повідомить користувача в ліфті про відсутність зв'язку та неможливість у даний час оброблювати команди. І почне кожні 15 секунд намагатись перепідключитись. Це показано на рис. 4.17. Якщо зв'язок відновиться, то користувач в ліфті також буде повідомлений.

Якщо розрив відбудеться через відсутність інтернету ESP32 та сервер це зрозуміють через 40 секунд максимум, а то і швидше. Знову ж таки ESP32 повідомить користувача про відсутність зв'язку та неможливість у даний час оброблювати команди. І почне кожні 15 секунд намагатись перепідключитись. А сервер повідомить ліфтера про проблему зі зв'язком.

Якщо розрив відбудеться через зникнення живлення на ESP32, то сервер повідомить про проблему ліфтера.

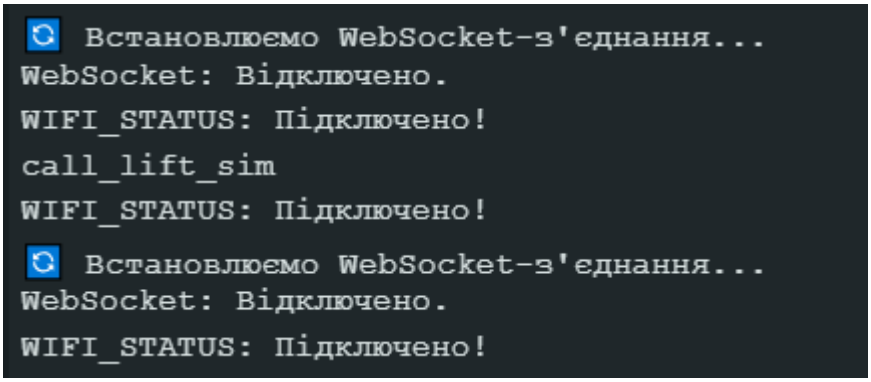


Рис. 4.17 – Випадок з вимкненим сервером

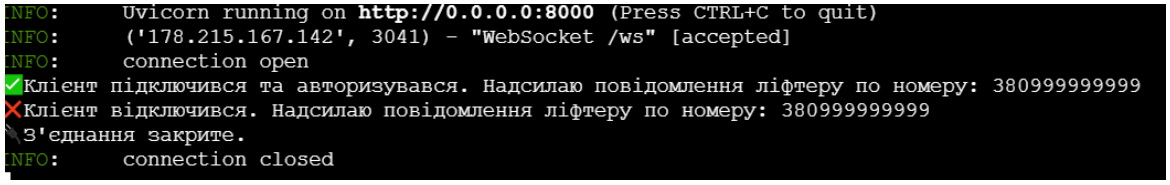


Рис. 4.18 – Випадок з відсутністю зв’язку з ESP32 на стороні серверу

4.4. Тестування точності розпізнавання команд

Було проведено 20 спроб, з яких успішних 15, а помилкових 5. Як спроби також вважались підтвердження (фрази «Так»). Відсоток правильності розпізнавання команд зображено на рис. 4.19. З діаграми видно, що в середньому 75% команд успішно розпізнається. Як для дешевого прототипу я вважаю, що це досить непоганий результат.

Рис. 4.19 – Відсоткова секторна діаграма успішності розпізнавання команд

4.5. Оцінка затримки між передачею аудіо та отриманням відповіді від сервера

Час запису пакетів – це час, який витрачається на збір даних з мікрофону.

Час обробки аудіо – це час, який рахується на сервері та показує як довго сервер обробляв дані.

Час реакції серверу – це Час обробки аудіо + Час передачі даних (загальний). Це по суті фактичний час очікування користувача.

Було проведено 10 тестувань, які зазначені в таблиці 4.1.

Таблиця 4.1 - Результати вимірювання часу

№	Час запису пакетів, с	Час реакції серверу, с	Час передачі даних, с	Час обробки аудіо, с	Загальний час запису, передачі та аналізу, с
1.	4.19	2.46	0.19	2.27	6.66
2.	3.59	3.36	0.24	3.12	6.94
3.	4.45	2.05	0.10	1.95	6.50
4.	3.18	2.06	0.14	1.92	5.25
5.	4.19	4.47	0.13	4.34	8.93
6.	2.66	2.27	0.11	2.16	4.93
7.	4.10	3.90	0.09	3.81	8.00
8.	3.23	1.28	0.18	1.10	4.51
9.	3.66	2.18	0.13	2.05	5.84
10.	3.55	2.30	0.11	2.19	5.85

З таблиці видно, що середній фактичний час очікування користувача на обробку становить 2.633 секунди.

Висновок

На основі проведеного тестування можна зробити висновок, що веб-сайт налаштувань працює стабільно та ефективно у всіх запланованих сценаріях. Всі форми (Wi-Fi, Точка доступу, Конфігурація) успішно передають дані на ESP32, забезпечуючи їх збереження в енергонезалежній пам’яті, що підтверджується виводами в режимі налагодження (debug).

Система адекватно реагує на некоректний ввід: перевіряється заповненість обов’язкових полів, довжина паролів та номери телефонів, а також діапазон допустимих значень. Це дозволяє уникати помилок у роботі ще на етапі налаштування, підвищуючи загальну надійність системи.

Обробка з’єднання з маршрутизатором реалізована з урахуванням повторних спроб підключення, а користувач отримує зворотний зв’язок як через сайт, так і через відповідні повідомлення у порту. Аналогічно реалізовано логіку обробки помилок WebSocket-з’єднання — ESP32 автоматично повідомляє користувача про відсутність зв’язку, а сервер, своєю чергою, інформує адміністратора про збої.

У тестуванні точності розпізнавання голосових команд система досягла 75% успішності, що є достатньо добрим результатом для недорогого прототипу. Це свідчить про реальну придатність системи до використання в умовах обмежених ресурсів.

Середній час реакції становив приблизно 2.63 секунди, що є цілком прийнятним показником з огляду на специфіку задачі. Система демонструє задовільний рівень швидкодії для інтерфейсу, орієнтованого на голосове керування.

Загалом, система веб-налаштування, зв’язку з сервером і обробки голосових команд показала високу стабільність, передбачувану поведінку в умовах помилок, а також задовільну продуктивність, що дозволяє рекомендувати її для використання в системах голосового керування ліфтом.

ВИСНОВКИ

У результаті виконаної роботи було всебічно досліджено, розроблено та протестовано комп'ютерну систему голосового управління ліфтом на основі мікроконтролера ESP32. Під час дослідження проведено аналіз сучасного стану ліфтових систем, особливо в контексті інклюзивності та адаптації до потреб маломобільних груп населення. Було встановлено, що більшість існуючих ліфтів в Україні залишаються технологічно застарілими, з фізичним інтерфейсом, не пристосованим до людей з інвалідністю, осіб похилого віку чи користувачів, які не мають змоги взаємодіяти з кнопками через зайняті руки.

У контексті війни в Україні та зростання кількості осіб з інвалідністю через бойові дії, проблема доступності набуває особливої актуальності. Тому розробка бюджетного, автономного та локалізованого рішення на основі ESP32 є не лише інженерним викликом, а й соціально важливою ініціативою.

На апаратно-програмному рівні було створено модульну систему, що включає:

- мікрофон INMP441 для зчитування аудіо;
- DFPlayer Mini для відтворення звукових повідомлень;
- мікроконтролер ESP32 з підтримкою Wi-Fi;
- веб-інтерфейс для налаштування параметрів збережених у енергонезалежній пам'яті.

Усі елементи системи працюють у синергії завдяки реалізованим алгоритмам взаємодії з користувачем, апаратною частиною та серверною логікою. Забезпечено надійну передачу аудіопотоку через WebSocket, адаптивну обробку помилок, повторні спроби підключення та інформування користувача у випадку збоїв.

Результати тестування продемонстрували, що система здатна стабільно функціонувати у реальних умовах. Рівень точності розпізнавання команд становить близько 75%, а середній час реакції — 2.63 секунди, що є прийнятним для задач голосового інтерфейсу з обмеженими ресурсами. Система

продемонструвала передбачувану поведінку, стійкість до помилок та зручність користування.

Таким чином, запропонована система:

- реалізована на доступних компонентах;
- підтримує українську мову;
- відповідає вимогам інклюзивності;
- не потребує радикальної модернізації ліфтової інфраструктури.

Розроблений прототип може стати основою для масштабованих проєктів модернізації житлового фонду, а також слугувати прикладом ефективного поєднання IoT-рішень з реальними потребами суспільства в умовах воєнного та післявоєнного відновлення.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Інтернет речей [Електронний ресурс] // Вікіпедія : вільна енциклопедія. – Режим доступу: https://uk.wikipedia.org/wiki/Інтернет_речей – Дата доступу: квітень 2025.
2. I2S Communication Protocol [Електронний ресурс]. – 2020. – Режим доступу: https://en.wikipedia.org/wiki/I%C2%B2S – Дата доступу: квітень 2025.
3. WebSocket Protocol [Електронний ресурс]. – 2023. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/API/WebSocket – Дата доступу: квітень 2025.
4. UART Communication Protocol Explained [Електронний ресурс]. – Режим доступу: https://learn.sparkfun.com/tutorials/serial-communication/all – Дата доступу: квітень 2025.
5. I²C [Електронний ресурс] // Вікіпедія : вільна енциклопедія. – Режим доступу: https://uk.wikipedia.org/wiki/I²C – Дата доступу: квітень 2025.
6. SPI Protocol Tutorial [Електронний ресурс]. – Режим доступу: https://www.analog.com/en/analog-dialogue/articles/introduction-to-spi-interface.html – Дата доступу: квітень 2025.
7. API Design Guidelines [Електронний ресурс]. – Режим доступу: https://docs.microsoft.com/en-us/azure/architecture/best-practices/api-design – Дата доступу: квітень 2025.
8. OpenAI API Documentation [Електронний ресурс]. – 2024. – Режим доступу: https://platform.openai.com/docs – Дата доступу: квітень 2025.
9. ESP32 Overview [Електронний ресурс]. – 2020. – Режим доступу: https://www.espressif.com/en/products/socs/esp32/overview – Дата доступу: квітень 2025.

10. OTA Web Configuration for IoT Devices [Електронний ресурс]. – 2022. – Режим доступу: https://randomnerdtutorials.com/esp32-access-point-ap-web-server/ – Дата доступу: квітень 2025.
11. Secure Digital [Електронний ресурс] // Вікіпедія : вільна енциклопедія. – Режим доступу: https://uk.wikipedia.org/wiki/Secure_Digital – Дата доступу: квітень 2025.
12. DFPlayer Mini MP3 Player Module [Електронний ресурс]. – 2019. – Режим доступу: https://wiki.dfrobot.com/DFPlayer_Mini_SKU_DFR0299 – Дата доступу: квітень 2025.
13. INMP441 MEMS Microphone Datasheet [Електронний ресурс]. – Режим доступу: https://www.invensense.com/products/audio/inmp441/ – Дата доступу: квітень 2025.
14. Amazon Web Services Documentation – EC2 [Електронний ресурс]. – Режим доступу: https://docs.aws.amazon.com/ec2/index.html – Дата доступу: квітень 2025.
15. Introducing JSON [Електронний ресурс]. – Режим доступу: https://www.json.org/json-en.html – Дата доступу: квітень 2025.
16. Otis Gen3 [Електронний ресурс]. – Режим доступу: https://elevation.fandom.com/wiki/Otis_Gen3 – Дата доступу: квітень 2025.
17. KONE KDS Home V [Електронний ресурс]. – Red Dot Design Award, 2021. – Режим доступу: https://www.red-dot.org/project/kone-kds-home-v-57602 – Дата доступу: квітень 2025.
18. Schindler CleanCall HoloVoice [Електронний ресурс]. – Режим доступу: https://www.schindler.com/en/elevators/cleanmobility/cleancall-holovoice.html – Дата доступу: квітень 2025.

19. TK Elevator MAX: Predictive Maintenance Solution [Електронний ресурс]. – Режим доступу: https://www.tkelevator.com/global-en/products/digital-solutions/max/ – Дата доступу: квітень 2025.
20. Vosk Speech Recognition Toolkit [Електронний ресурс]. – 2021. – Режим доступу: https://alphacephei.com/vosk/ – Дата доступу: квітень 2025.
21. WiFi.h – Підключення ESP32 до Wi-Fi [Електронний ресурс]. – Режим доступу: https://github.com/espressif/arduino-esp32/tree/master/libraries/WiFi – Дата доступу: квітень 2025.
22. WebServer.h – Створення HTTP-сервера на ESP32 [Електронний ресурс]. – Режим доступу: https://github.com/espressif/arduino-esp32/tree/master/libraries/WebServer – Дата доступу: квітень 2025.
23. Preferences.h – Робота з енергонезалежною пам'яттю ESP32 (NVS) [Електронний ресурс]. – Режим доступу: https://github.com/espressif/arduino-esp32/tree/master/libraries/Preferences – Дата доступу: квітень 2025.
24. driver/i2s.h – Документація ESP-IDF для інтерфейсу I2S [Електронний ресурс]. – Режим доступу: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/i2s.html – Дата доступу: квітень 2025.
25. WebSocketsClient.h – Бібліотека WebSocket-клієнта для Arduino [Електронний ресурс]. – Режим доступу: https://github.com/Links2004/arduinoWebSockets – Дата доступу: квітень 2025.
26. ArduinoJson.h – Документація по роботі з JSON у мікроконтролерах [Електронний ресурс]. – Режим доступу: https://arduinojson.org/ – Дата доступу: квітень 2025.
27. queue – C++ STL: Очереді (std::queue) [Електронний ресурс]. – Режим доступу:

https://en.cppreference.com/w/cpp/container/queue – Дата доступу: квітень 2025.

28. vector – C++ STL: Вектори (std::vector) [Електронний ресурс]. – Режим доступу: https://en.cppreference.com/w/cpp/container/vector – Дата доступу: квітень 2025.
29. HardwareSerial.h – Послідовний порт ESP32 (UART) [Електронний ресурс]. – Режим доступу: https://github.com/espressif/arduino-esp32/blob/master/cores/esp32/HardwareSerial.h – Дата доступу: квітень 2025.
30. DFRobotDFPlayerMini.h – Бібліотека для керування DFPlayer Mini [Електронний ресурс]. – Режим доступу: https://github.com/DFRobot/DFRobotDFPlayerMini – Дата доступу: квітень 2025.
31. FastAPI – Офіційна документація FastAPI [Електронний ресурс]. – Режим доступу: https://fastapi.tiangolo.com/ – Дата доступу: квітень 2025.
32. vosk – Документація Vosk API для Python [Електронний ресурс]. – Режим доступу: https://alphacephei.com/vosk/ – Дата доступу: квітень 2025.
33. json – Стандартна бібліотека Python (модуль json) [Електронний ресурс]. – Режим доступу: https://docs.python.org/3/library/json.html – Дата доступу: квітень 2025.
34. wave – Робота з WAV-файлами в Python [Електронний ресурс]. – Режим доступу: https://docs.python.org/3/library/wave.html – Дата доступу: квітень 2025.

35. os – Стандартний модуль для роботи з ОС [Електронний ресурс]. – Режим доступу: https://docs.python.org/3/library/os.html – Дата доступу: квітень 2025.
36. tempfile – Створення тимчасових файлів і каталогів [Електронний ресурс]. – Режим доступу: https://docs.python.org/3/library/tempfile.html – Дата доступу: квітень 2025.
37. openai – Клієнтська бібліотека OpenAI для Python [Електронний ресурс]. – Режим доступу: https://github.com/openai/openai-python – Дата доступу: квітень 2025.
38. dotenv (load_dotenv) – Робота з .env-файлами в Python [Електронний ресурс]. – Режим доступу: https://github.com/theskumar/python-dotenv – Дата доступу: квітень 2025.
39. uvicorn – Легкий ASGI-сервер для FastAPI [Електронний ресурс]. – Режим доступу: https://www.uvicorn.org/ – Дата доступу: квітень 2025.
40. time – Стандартна бібліотека Python для роботи з часом [Електронний ресурс]. – Режим доступу: https://docs.python.org/3/library/time.html – Дата доступу: травень 2025.

ДОДАТОК 1

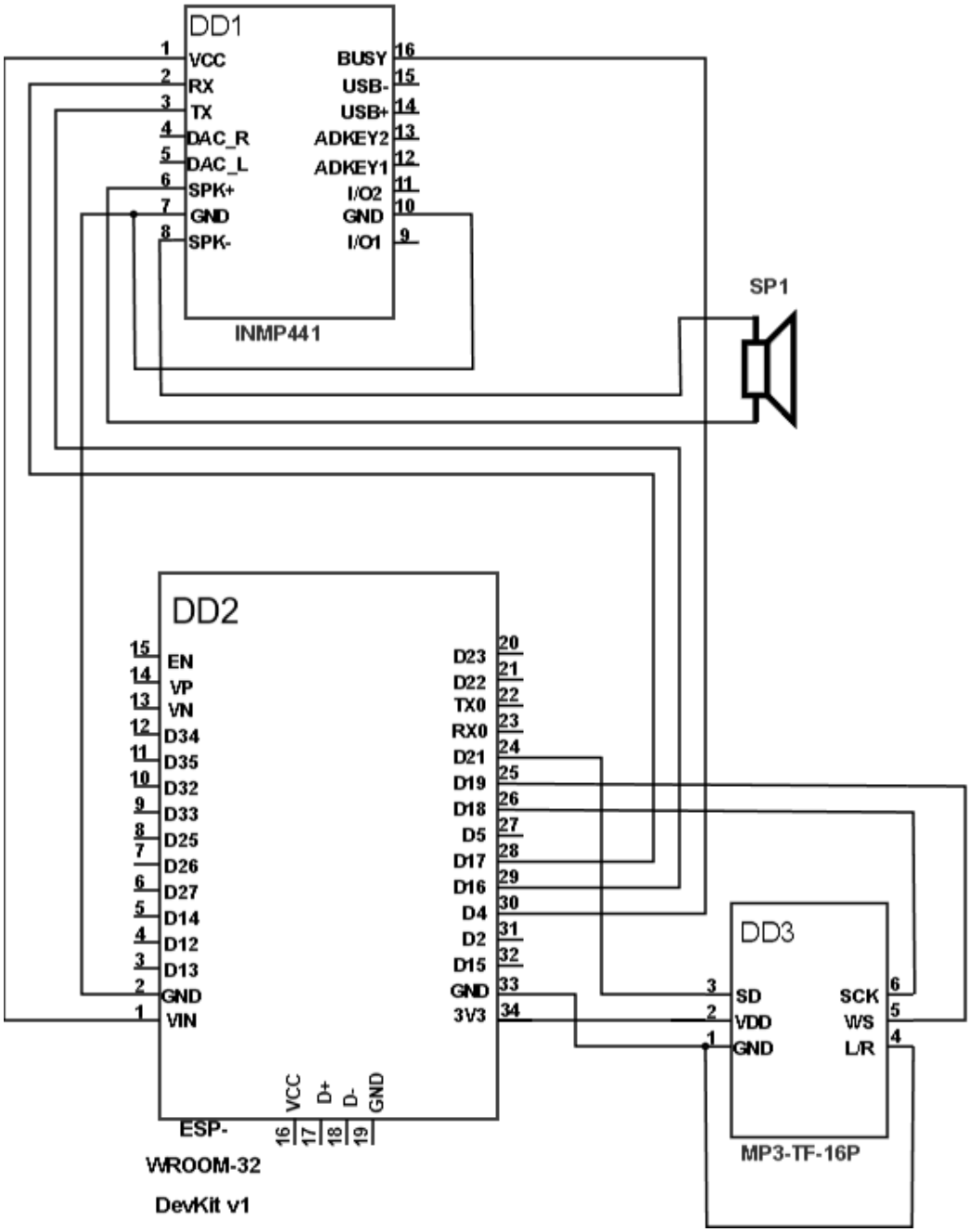
Комп'ютерна система голосового управління ліфтом на основі мікроконтролера ESP32

Комп'ютерна система голосового управління ліфтом на основі мікроконтролера ESP32. Схема електрична принципова

ІАЛЦ. 045200.005 Е3

Аркушів 1

Київ - 2025



					ІАЛЦ. 045200.005 Е3	
Зм	*Лист*	*№ докум.*	*Підп.*	*Дата*		
Розроб.		Гусельніков А. О.			Комп'ютерна система голосового управління ліфтом на основі мікроконтролера ESP32	*Літ.* / *Лист* / *Листів*: 1 / 1
Перев.		Петрашенко А. В.				
Н. контр.		Клятченко Я. М.				**НТУУ «КПІ ім. Ігоря Сікорського», ФПМ, КВ-12**
Затв.		Романкевич В.О.			*Схема електрична принципова*	

Позначення	Найменування	Кільк.	Прим.
	Мікросхеми		
DD1	INMP441	1	
DD2	ESP-WROOM-32 (30pin) DevKit v1	1	
DD3	DFPlayer Mini (MP3-TF-16P)	1	
	Динаміки		
SP1	Динамік 3Вт 8Ом	1	

Зм	Лист	№ докум.	Підп.	Дата	ІАЛЦ. 045200.006 ПЕ		
Розроб.		Гусельніков А. О.			Комп'ютерна система голосового управління ліфтом на основі мікроконтролера ESP32. Схема електрична принципова	Літ.	Лист 1 / Листів 1
Перев.		Петрашенко А. В.					
Н. контр.		Клятченко Я. М.			Перелік елементів	НТУУ «КПІ ім. Ігоря Сікорського», ФПМ, КВ-12	
Затв.		Романкевич В. О.					

ДОДАТОК 2

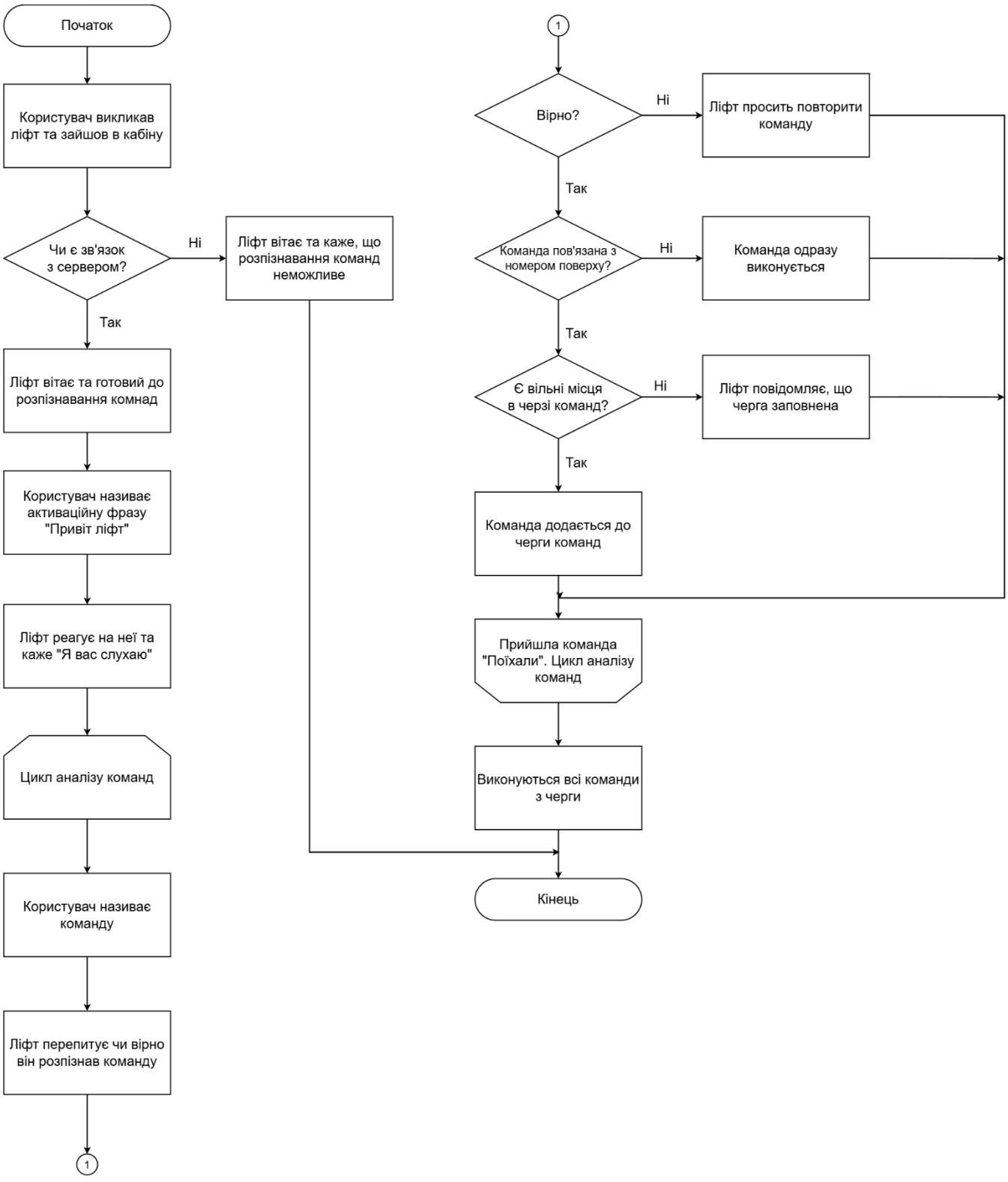
Комп'ютерна система голосового управління ліфтом на основі мікроконтролера ESP32

Алгоритм взаємодії користувача з системою. Блок-схема

ІАЛЦ. 045200.007 Д1

Аркушів 1

Київ - 2025



					ІАЛЦ. 045200.007 Д1	
Зм	***Лист***	***№ докум.***	***Підп.***	***Дата***		
Розроб.		Гусельніков А. О.			Комп'ютерна система голосового управління ліфтом на основі мікроконтролера ESP32 ***Алгоритм взаємодії користувача з системою. Блок-схема***	***Літ.*** / ***Лист*** 1 / ***Листів*** 1
Перев.		Петрашенко А. В.				
Н. контр.		Клятченко Я. М.				**НТУУ «КПІ ім. Ігоря Сікорського», ФПМ, КВ-12**
Затв.		Романкевич В. О.				

ДОДАТОК З

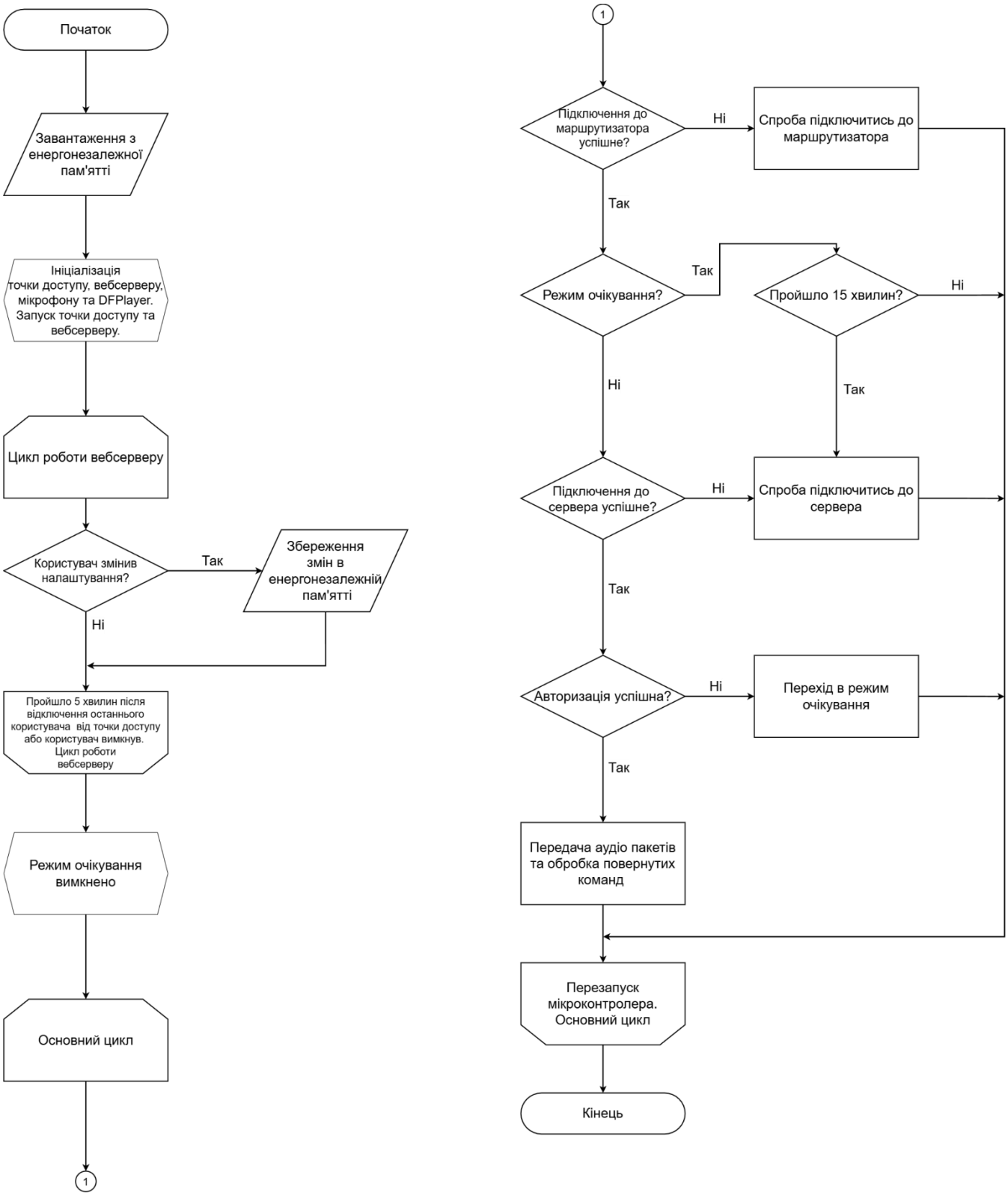
Комп'ютерна система голосового управління ліфтом на основі мікроконтролера ESP32

Алгоритм роботи апаратної частини системи. Блок-схема

ІАЛЦ. 045200.008 Д2

Аркушів 1

Київ - 2025



Зм	Лист	№ докум.	Підп.	Дата	ІАЛЦ. 045200.008 Д2		
Розроб.		Гусельніков А. О.			Комп'ютерна система голосового управління ліфтом на основі мікроконтролера ESP32 *Алгоритм роботи апаратної частини системи. Блок-схема*	Літ.	Лист 1 / Листів 1
Перев.		Петрашенко А. В.				НТУУ «КПІ ім. Ігоря Сікорського», ФПМ, КВ-12	
Н. контр.		Клятченко Я. М.					
Затв.		Романкевич В. О.					

ДОДАТОК 4

Комп'ютерна система голосового управління ліфтом на основі мікроконтролера ESP32

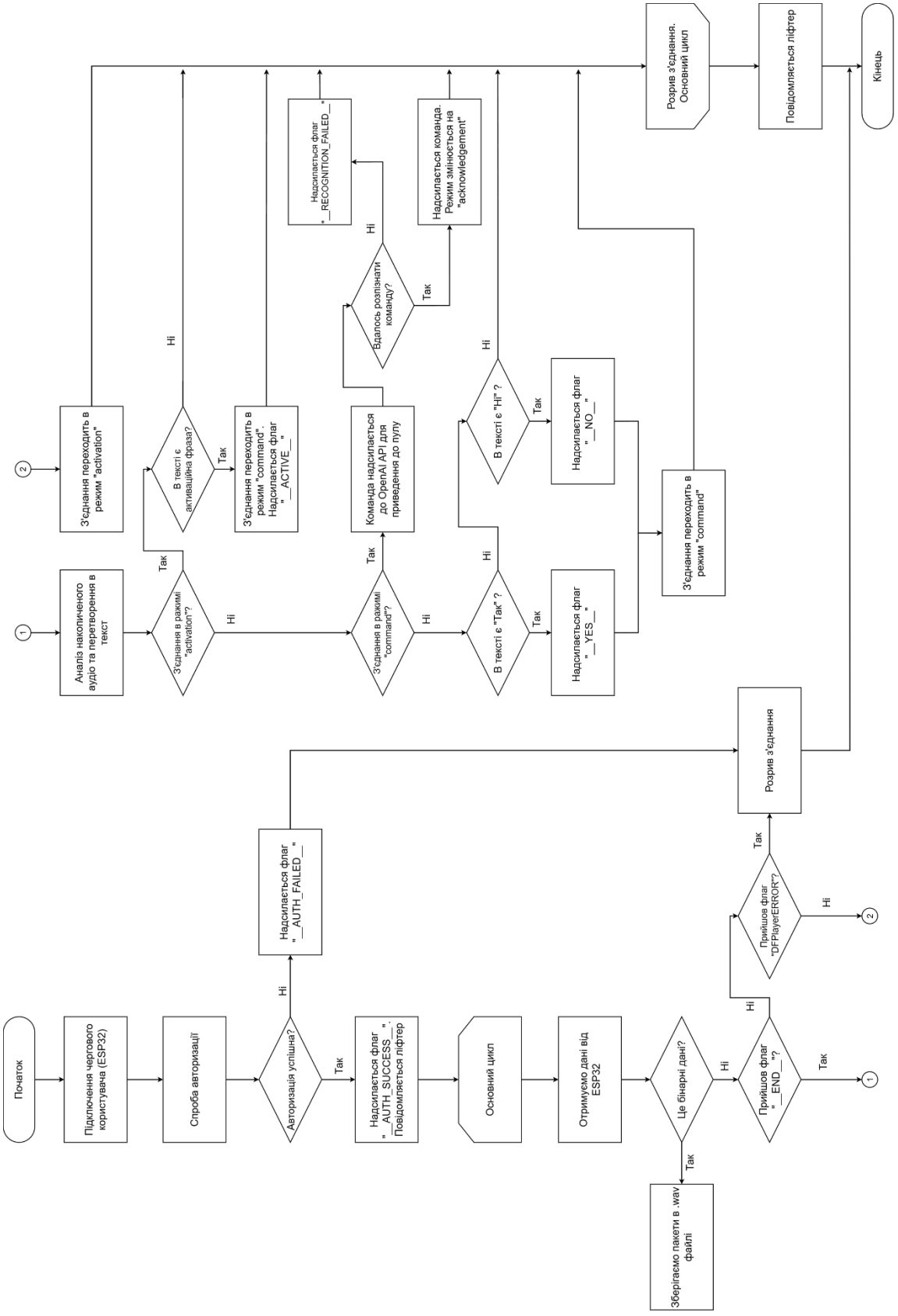
Алгоритм роботи серверної частини системи.

Блок-схема

ІАЛЦ. 045200.009 Д3

Аркушів 1

Київ - 2025



Зм	Лист	№ докум.	Підп.	Дата
Розроб.		Гусельніков А. О.		
Перев.		Петрашенко А. В.		
Н. контр.		Клятченко Я. М.		
Затв.		Романкевич В. О.		

ІАЛЦ. 045200.009 Д3

Комп'ютерна система голосового управління ліфтом на основі мікроконтролера ESP32

Алгоритм роботи серверної частини системи

Літ.	Лист	Листів
	1	1

НТУУ «КПІ ім. Ігоря Сікорського», ФПМ, КВ-12

ДОДАТОК 5

Комп'ютерна система голосового управління ліфтом на основі мікроконтролера ESP32

Текст програмного коду

ІАЛЦ. 045200.010 Д4

Аркушів 1

Київ - 2025

main.ino

```
#include <WiFi.h>
#include <WebServer.h>
#include <Preferences.h>
#include <driver/i2s.h>
#include <WebSocketsClient.h>
#include <ArduinoJson.h>
#include <queue>
#include <vector>
#include "ElevatorCommands.h"
#include <HardwareSerial.h>
#include "PlayerController.h"


// Режим дебагу (вивід усього в COM)

bool isDebug = true;
unsigned long start_listening;
unsigned long end_listening;
unsigned long send;
unsigned long receive;
bool start = true;

// Режим дебагу (вивід усього в COM)

//
int errCode = -1;
bool blockReconnect = false;
//

// Точка доступу

bool isAPActive = false;
const unsigned long timeout = 1 * 60 * 5000;  // 5 хвилин
unsigned long lastCheckTime = 0;

// Точка доступу

// Інтернет

bool isConnectedToInternet = false;
bool startCheckStatus = false;
bool firstTry = true;
unsigned int errCnt = 0;
unsigned long timerToShowStatus = 0;

// Інтернет

// Дані та флеш-пам'ять

Preferences preferences;
String targetSSID = "";
String targetPsw = "";
String connectionStatus = "Відключено.";
String ssid = "";
String password = "";
String command_buf = "";
int floors = 9;
String number = "";

// Дані та флеш-пам'ять

// WebServer

WebServer server(80);

// WebServer
```

```
// Мікрофон
#define I2S_SCK 18  //зелений
#define I2S_WS 19   //помаранчевий
#define I2S_SD 21   //синій
#define I2S_PORT I2S_NUM_0

#define bufferLen 512
#define NOISE_THRESHOLD_FACTOR 1.3
#define CONTINUE_SENDING_TIME 1500

int16_t sBuffer[bufferLen];
int16_t sBuffer2[bufferLen];

float baselineNoise = 100;
bool isSpeaking = false;
bool sendEND = false;
unsigned long lastSpeechTime = 0;

// Мікрофон

// WebSocket

WebSocketsClient client;
unsigned long WEBSOCKET_RECONECT_TIME = 15000;
int websocket_tries = 0;
bool isWebSocketConnected = false;
bool reconecting = false;
unsigned long wsReconnectTimer = 0;
bool waiting = false;
unsigned long lastWebSocketActivity = 0;
const unsigned long WS_ACTIVITY_TIMEOUT = 45000;

// WebSocket

// Черга команд до ліфту

std::queue<ElevatorCommand*> commandQueue;
ElevatorCommand* cmd_buff;
int cmd_limit = 8;

// Черга команд до ліфту

// Динамік

#define PIN_MP3_RX 16  // RX ESP32 ← TX DFPlayer
#define PIN_MP3_TX 17  // TX ESP32 → RX DFPlayer
HardwareSerial dfSerial(1);
PlayerController myPlayer(dfSerial, PIN_MP3_RX, PIN_MP3_TX);

// Динамік

// Симуляція запуску

bool ff = true;
unsigned long configEND = 0;

// Симуляція запуску

void msgToCom(String msg) {
  if (isDebug) {
    Serial.println(msg);
  }
}

void updateConnectionStatus(String newStatus) {
  connectionStatus = newStatus;
}
```

```
String generateHTML() {
  String html = "<html><head>";
  html += "<meta charset='UTF-8'>";
  html += "<meta name='viewport' content='width=device-width, initial-
scale=1.0'>";
  html += "<title>Налаштування</title>";
  html += "<style>";
  html += "h2 {text-align: center;}";
  html += "body { font-family: Arial, sans-serif; font-size: 16px;
text-align: left; padding: 20px; }";
  html += "form { display: inline-block; text-align: left; margin-
bottom: 20px; width: 100%; }";
  html += "input, button { padding: 8px; font-size: 14px; width: 100%;
margin-top: 5px; }";
  html += "</style>";
  html += "</head><body>";

  html += "<h2>Зміна даних Wi-Fi:</h2>";


  html += "<form id='wifiForm'>";
  html += "SSID роутера:<br><input type='text' id='ssidInput' value='"
+ targetSSID + "' required><br><br>";
  html += "Пароль роутера:<br><input type='text' id='pswInput' value='"
+ targetPsw + "' required><br><br>";
  html += "<button type='submit'>Підтвердити</button>";
  html += "</form>";

  html += "<br><br><hr><br><br>";

  html += "<form id='connectForm'>";
  html += "<h3>Збережені дані: <span id='savedWIFIData' style='font-
weight:bold;'><br>SSID роутера: " + targetSSID + "<br>Пароль роутера: "
+ targetPsw + "</span></h3>";
  html += "  <button type='submit'>Підключитись до Wi-Fi</button>";
  html += "</form>";
  String statusColor = "black";
  if (connectionStatus.indexOf("Підключено") >= 0) {
    statusColor = "green";
  } else if (connectionStatus.indexOf("Відключено") >= 0) {
    statusColor = "red";
  } else if (connectionStatus.indexOf("Підключення") >= 0) {
    statusColor = "orange";
  }

  html += "<p style='font-weight:bold;'>Статус: <span id='statusText'
style='color:" + statusColor + ";'>" + connectionStatus +
"</span></p>";

  html += "<br><br><hr><br><br>";

  html += "<h2>Зміна даних Точки доступу:</h2>";

  html += "<form id='apForm'>";
  html += "SSID Точки доступу:<br><input type='text' id='ssidApInput'
value='" + ssid + "' required><br><br>";
  html += "Пароль Точки доступу:<br><input type='text' id='pswApInput'
value='" + password + "' required><br><br>";
  html += "   <button type='submit'>Підтвердити</button>";
  html += "</form>";

  html += "<h3>Збережені дані: <span id='savedAPData' style='font-
weight:bold;'><br>SSID Точки доступу: " + ssid + "<br>Пароль точки
доступу: " + password + "</span></h3>";
  html += "<br><br><hr><br><br>";

  html += "<h2>Зміна даних Конфігурації:</h2>";
```

```
  html += "<form id='configForm'>";
  html += "Кількість поверхів<br><input type='number' id='floorsInput'
min='1' max='100' value='" + String(floors) + "' required><br><br>";
  html += "Номер телефону ліфтера:<br><input type='number'
id='phoneInput' value='" + number + "' required><br><br>";
  html += "   <button type='submit'>Підтвердити</button>";
  html += "</form>";

  html += "<h3>Збережені дані: <span id='savedCONFIG' style='font-
weight:bold;'><br>Кількість поверхів: " + String(floors) + "<br>Номер
телефону ліфтера: " + number + "</span></h3>";
  html += "<br><br><hr><br><br>";


  html += "<form id='endForm' method='POST' action='/END'>";
  html += "  <button type='submit'>Вимкнути точку доступу</button>";
  html += "</form>";

  // JavaScript
  html += "<script>";
  html +=
"document.getElementById('connectForm').addEventListener('submit',
function(e) {";
  html += "  e.preventDefault();";
  html += "  fetch('/connect')";
  html += "    .then(response => response.text())";
  html += "    .then(data => {";
  html += "      const statusText =
document.getElementById('statusText');";
  html += "      statusText.innerText = data;";
  html += "      if (data.includes('Підключено')) {";
  html += "        statusText.style.color = 'green';";
  html += "      } else if (data.includes('Відключено')) {";
  html += "        statusText.style.color = 'red';";
  html += "      } else if (data.includes('Підключення')) {";
  html += "        statusText.style.color = 'orange';";
  html += "      } else {";
  html += "        statusText.style.color = 'black';";
  html += "      }";
  html += "    });";
  html += "});";

  html +=
"document.getElementById('wifiForm').addEventListener('submit',
function(e) {";
  html += "  e.preventDefault();";
  html += "  const ssid = document.getElementById('ssidInput').value;";
  html += "  const psw = document.getElementById('pswInput').value;";
  html += "  if (psw.length < 8) {";
  html += "    alert('Пароль має містити мінімум 8 символів.');";
  html += "    return;";
  html += "  }";
  html += "  fetch('/submit', {";
  html += "    method: 'POST',";
  html += "    headers: { 'Content-Type': 'application/x-www-form-
urlencoded' },";
  html += "    body: 'targetSSID=' + encodeURIComponent(ssid) +
'&targetPsw=' + encodeURIComponent(psw)";
  html += "  }).then(response => {";
  html += "    if (response.ok) {";
  html += "      document.getElementById('savedWIFIData').innerHTML =
`<br>SSID роутера: ${ssid}<br>Пароль роутера: ${psw}`;";
  html += "    }";
  html += "  });";
  html += "});";

  html += "document.getElementById('apForm').addEventListener('submit',
function(e) {";
```

```
  html += "  e.preventDefault();";
  html += "  const ssid =
document.getElementById('ssidApInput').value;";
  html += "  const pass =
document.getElementById('pswApInput').value;";
  html += "  if (pass.length < 8) {";
  html += "    alert('Пароль має містити мінімум 8 символів.');";
  html += "    return;";
  html += "  }";
  html += "  fetch('/submitAP', {";
  html += "    method: 'POST',";
  html += "    headers: { 'Content-Type': 'application/x-www-form-
urlencoded' },";
  html += "    body: 'ssid=' + encodeURIComponent(ssid) + '&pass=' +
encodeURIComponent(pass)";
  html += "  }).then(response => {";
  html += "    if (response.ok) {";
  html += "      document.getElementById('savedAPData').innerHTML =
`<br>SSID Точки доступу: ${ssid}<br>Пароль точки доступу: ${pass}`;";
  html += "    }";
  html += "  });";
  html += "});";

  html +=
"document.getElementById('configForm').addEventListener('submit',
function(e) {";
  html += "  e.preventDefault();";
  html += "  const floors =
document.getElementById('floorsInput').value;";
  html += "  const phone =
document.getElementById('phoneInput').value;";
  html += "  if (phone.length !== 12) {";
  html += "    alert('Номер телефону повинен містити рівно 12 символів.
Наприклад, 380#########.');";
  html += "    return;";
  html += "  }";
  html += "  fetch('/submitCONFIG', {";
  html += "    method: 'POST',";
  html += "    headers: { 'Content-Type': 'application/x-www-form-
urlencoded' },";
  html += "    body: 'floors=' + encodeURIComponent(floors) + '&phone='
+ encodeURIComponent(phone)";
  html += "  }).then(response => {";
  html += "    if (response.ok) {";
  html += "      document.getElementById('savedCONFIG').innerHTML =
`<br>Кількість поверхів: ${floors}<br>Номер телефону ліфтера:
${phone}`;";
  html += "    }";
  html += "  });";
  html += "});";


  html += "function updateStatus() {";
  html += "  fetch('/status')";
  html += "    .then(response => response.text())";
  html += "    .then(data => {";
  html += "      const statusText =
document.getElementById('statusText');";
  html += "      statusText.innerText = data;";
  html += "      if (data.includes('Підключено')) {";
  html += "        statusText.style.color = 'green';";
  html += "      } else if (data.includes('Відключено')) {";
  html += "        statusText.style.color = 'red';";
  html += "      } else if (data.includes('Підключення')) {";
  html += "        statusText.style.color = 'orange';";
  html += "      } else {";
  html += "        statusText.style.color = 'black';";
  html += "      }";
```

```
  html += "    });";
  html += "}";
  html += "setInterval(updateStatus, 2000);";

  html += "</script>";

  html += "</body></html>";
  return html;
}



void handleRoot() {
  server.send(200, "text/html", generateHTML());
}



void handleSubmit() {
  if (server.hasArg("targetSSID") && server.hasArg("targetPsw")) {
    targetSSID = server.arg("targetSSID");
    targetPsw = server.arg("targetPsw");

    preferences.putString("targetSSID", targetSSID);
    preferences.putString("targetPsw", targetPsw);
    startCheckStatus = false;

    msgToCom("Дані збережені!");
    msgToCom("Оновлений targetSSID: " + targetSSID);
    msgToCom("Оновлений targetPsw: " + targetPsw);
    server.send(200, "text/plain", "OK");
  }
}

void handleAPSubmit() {
  if (server.hasArg("ssid") && server.hasArg("pass")) {
    ssid = server.arg("ssid");
    password = server.arg("pass");

    preferences.putString("ssid", ssid);
    preferences.putString("pass", password);

    msgToCom("Дані збережені!");
    msgToCom("Оновлений SSID: " + ssid);
    msgToCom("Оновлений Psw: " + password);
    server.send(200, "text/plain", "OK");
  }
}

void handleCONFIGSubmit() {
  if (server.hasArg("floors") && server.hasArg("phone")) {
    floors = server.arg("floors").toInt();
    number = server.arg("phone");

    preferences.putInt("floors", floors);
    preferences.putString("number", number);

    msgToCom("Дані збережені!");
    msgToCom("Оновлений floors: " + String(floors));
    msgToCom("Оновлений number: " + number);

    server.send(200, "text/plain", "OK");
  }
}

void checkWiFiStatus() {
```

```
  int status = WiFi.status();
  if (status == WL_CONNECTED) {
    msgToCom("WIFI_STATUS: Підключено!");
    updateConnectionStatus("Підключено");
    isConnectedToInternet = true;
    errCnt = 0;
  } else {
    if (errCnt < 12) {
      msgToCom("WIFI_STATUS: Підключення...");
      updateConnectionStatus("Підключення...");
      isConnectedToInternet = false;
      ++errCnt;
    } else {
      msgToCom("WIFI_STATUS: Відключено.");
      updateConnectionStatus("Відключено. Перевірте дані та доступність
роутеру.");
      isConnectedToInternet = false;
      startCheckStatus = false;
    }
  }
  timerToShowStatus = millis();
}

void connectToWiFi() {
  WiFi.disconnect(true);
  delay(100);
  errCnt = 0;
  WiFi.begin(targetSSID.c_str(), targetPsw.c_str());
  startCheckStatus = true;
  checkWiFiStatus();
}

void stopAP() {
  WiFi.softAPdisconnect(true);
  server.stop();
  isAPActive = false;
  msgToCom("Точка доступу та веб-сервер вимкнені");
  configEND = millis();
}

void startAP() {
  WiFi.mode(WIFI_AP_STA);
  WiFi.softAP(ssid.c_str(), password.c_str());

  isAPActive = true;
  lastCheckTime = millis();

  server.begin();
  msgToCom("Точка доступу та веб-сервер увімкнені");
}



void i2s_install() {
  // Set up I2S Processor configuration
  const i2s_config_t i2s_config = {
    .mode = i2s_mode_t(I2S_MODE_MASTER | I2S_MODE_RX),
    .sample_rate = 16000,
    .bits_per_sample = i2s_bits_per_sample_t(16),
    .channel_format = I2S_CHANNEL_FMT_ONLY_LEFT,
    .communication_format =
i2s_comm_format_t(I2S_COMM_FORMAT_STAND_I2S),
    .intr_alloc_flags = 0,
    .dma_buf_count = 8,
    .dma_buf_len = bufferLen,
    .use_apll = false
  };
```

```
  i2s_driver_install(I2S_PORT, &i2s_config, 0, NULL);
}

void i2s_setpin() {
  // Set I2S pin configuration
  const i2s_pin_config_t pin_config = {
    .bck_io_num = I2S_SCK,
    .ws_io_num = I2S_WS,
    .data_out_num = -1,
    .data_in_num = I2S_SD
  };

  i2s_set_pin(I2S_PORT, &pin_config);
}


void listening() {
  size_t bytes_read = 0;
  float sum = 0;

  memcpy(sBuffer, sBuffer2, bufferLen * sizeof(int16_t));

  if (i2s_read(I2S_PORT, sBuffer2, sizeof(sBuffer2), &bytes_read,
portMAX_DELAY) == ESP_OK) {
    size_t samples_read = bytes_read / sizeof(int16_t);

    for (size_t i = 0; i < samples_read; ++i) {
      sum += abs(sBuffer[i]);
    }

    float mean = sum / samples_read;
    float threshold = baselineNoise * NOISE_THRESHOLD_FACTOR;

    if (mean > threshold) {
      if (!isSpeaking) {
        msgToCom("Людина почала розмовляти!");
        if (start) {
          start = false;
          start_listening = millis();
        }
        isSpeaking = true;
      }
      lastSpeechTime = millis();
    } else if (isSpeaking && millis() - lastSpeechTime >
CONTINUE_SENDING_TIME) {
      msgToCom("Людина замовкла...");
      isSpeaking = false;
      sendEND = true;
    }

    if (isSpeaking || millis() - lastSpeechTime <=
CONTINUE_SENDING_TIME) {
      sendAudioDataToWebSocket(sBuffer, bytes_read);
    }

    if (sendEND && millis() - lastSpeechTime > CONTINUE_SENDING_TIME) {
      sendEND = false;
      msgToCom("Надсилаю END");
      end_listening = millis();
      start = true;
      send = millis();
      client.sendTXT("__END__");

    }
  }
}

void sendAudioDataToWebSocket(int16_t* buffer, size_t bytes) {
```

```
  client.sendBIN((uint8_t*)buffer, bytes);

  //msgToCom("Дані надіслані через WebSocket (" + String(bytes) + "
байт).");
}

void setupWebSocket() {
  client.disconnect();
  delay(500);
  client.begin("15.237.137.143", 8000, "/ws");
  client.onEvent(webSocketEvent);
}

void printLatency() {
  float recordTime = (end_listening - start_listening) / 1000.0;
  float responseTime = (receive - send) / 1000.0;
  float totalTime = recordTime + responseTime;

  msgToCom("Час запису пакетів: " + String(recordTime, 2) + " с");
  msgToCom("Час реакції серверу: " + String(responseTime, 2) + " с");
  msgToCom("Загальний час запису, передачі та аналізу: " +
String(totalTime, 2) + " с");
}

void webSocketEvent(WStype_t type, uint8_t* payload, size_t length) {
  receive = millis();
  switch (type) {
    case WStype_CONNECTED:
      {
        msgToCom("WebSocket: Підключено.");
        isWebSocketConnected = true;
        StaticJsonDocument<200> jsonDoc;
        jsonDoc["floors"] = floors;
        jsonDoc["number"] = number;

        String jsonString;
        serializeJson(jsonDoc, jsonString);
        client.sendTXT(jsonString);
        lastWebSocketActivity = millis();
        break;
      }
    case WStype_DISCONNECTED:
      {
        msgToCom("WebSocket: Відключено.");
        if (isWebSocketConnected) {
          reconecting = true;
          myPlayer.playTrack(2);
        }
        isWebSocketConnected = false;
        waiting = true;
        break;
      }
    case WStype_TEXT:
      {
        Serial.printf("Отримано повідомлення: %s\n", payload);
        String message = (const char*)payload;
        if (message == "__RECOGNITION_FAILED__") {
          myPlayer.playTrack(6);
        } else if (message == "__ACK_FAILED__") {
          myPlayer.playTrack(17);
        } else if (message == "__ACTIVE__") {
          myPlayer.playTrack(3);
        } else if (message == "__NOTHING__") {

        } else if (message == "__YES__") {


          if (command_buf == "список команд") {
```

```
            myPlayer.playTrack(9);
          } else if (command_buf == "поїхали") {
            myPlayer.playTrack(13);
            execute_all_commands();
            client.sendTXT("__RUN__");
          } else if (command_buf == "викликати ліфтера") {
            myPlayer.playTrack(10);
          } else if (command_buf == "скинути команди") {
            clear_all_commands();
            myPlayer.playTrack(11);
            client.sendTXT("__START__");
          } else if (command_buf == "відчинити двері") {
            myPlayer.playTrack(14);
            cmd_buff = new OpenDoorCommand();
            cmd_buff->execute();
            delete cmd_buff;
          } else if (command_buf == "зачинити двері") {
            myPlayer.playTrack(15);
            cmd_buff = new CloseDoorCommand();
            cmd_buff->execute();
            delete cmd_buff;
          } else {
            if (commandQueue.size() < cmd_limit) {
              commandQueue.push(cmd_buff);
              myPlayer.playTrack(8);
            } else {
              myPlayer.playTrack(12);
            }
          }
        } else if (message == "__NO__") {
          command_buf = "";
          myPlayer.playTrack(7);
        } else if (message == "__AUTH_FAILED__") {
          WEBSOCKET_RECONECT_TIME = 15*60*1000;  //15 хвилин
          isWebSocketConnected = false;
          wsReconnectTimer = millis();
          waiting = true;
          msgToCom("Авторизація провалена. Перехожу в режим
очікування.");
          client.disconnect();

        } else if (message == "__AUTH_SUCCESS__") {
          WEBSOCKET_RECONECT_TIME = 15000;  //15 секунд
          websocket_tries = 0;
          if (reconecting) {
            reconecting = false;
            clear_all_commands();
            myPlayer.playTrack(16);
            myPlayer.playTrack(1);
          }
          msgToCom("Авторизація пройдена. Починаю працювати.");
        } else {
          command_buf = message;
          command_recognizer();
        }

        if(message != "__AUTH_SUCCESS__" && message !=
"__AUTH_FAILED__") {
          printLatency();
        }
        lastWebSocketActivity = millis();
        break;
      }
    case WStype_PING:
      {
        lastWebSocketActivity = millis();
        msgToCom("WebSocket: Получен PING от сервера");
        break;
```

```
      }
    case WStype_ERROR:
      {
        msgToCom("WebSocket: Помилка.");
        break;
      }

  }
}

void call_lift_sim() {
  msgToCom("call_lift_sim");
  client.sendTXT("__START__");
  if (isConnectedToInternet && isWebSocketConnected) {
    myPlayer.playTrack(1);
  } else {
    myPlayer.playTrack(2);
  }
}

int findCommandIndex(const String& target) {
  std::vector<String> commands = {
    "скинути команди",
    "список команд",
    "поїхали",
    "відчинити двері",
    "зачинити двері",
    "викликати ліфтера"
  };
  int index = 0;
  for (auto it = commands.begin(); it != commands.end(); ++it, ++index)
{
    if (*it == target) {
      return index;
    }
  }
  return -1;
}


void command_recognizer() {
  int index;
  if (isDigit(command_buf.charAt(0))) {
    int floor = command_buf.toInt();
    msgToCom("FLOOR: " + String(floor));
    index = 23 + floor;
    cmd_buff = new GoToFloorCommand(floor);
  } else {
    int cmd = findCommandIndex(command_buf);
    index = cmd + 18;
  }

  myPlayer.playTrack(4);
  myPlayer.playTrack(index);
  myPlayer.playTrack(5);
}

void execute_all_commands() {
  while (!commandQueue.empty()) {
    ElevatorCommand* cmd = commandQueue.front();
    commandQueue.pop();
    cmd->execute();
    delete cmd;
  }
}

void clear_all_commands() {
  while (!commandQueue.empty()) {
```

```
    ElevatorCommand* cmd = commandQueue.front();
    commandQueue.pop();
    delete cmd;
  }
}

void sendError() {
  if (errCode == 1) {
    client.sendTXT("DFPlayerERROR");
  }
  blockReconnect = true;
}

void setup() {
  Serial.begin(115200);
  delay(1000);

  preferences.begin("storage", false);

  if (!preferences.isKey("ssid") || !preferences.isKey("pass")) {
    preferences.putString("ssid", "ESP32_AP");
    preferences.putString("pass", "12345688");
  }

  ssid = preferences.getString("ssid", "");
  password = preferences.getString("pass", "");


  targetSSID = preferences.getString("targetSSID", "");
  targetPsw = preferences.getString("targetPsw", "");

  floors = preferences.getInt("floors", 9);
  number = preferences.getString("number", "380#########");

  server.on("/", handleRoot);
  server.on("/submit", HTTP_POST, handleSubmit);
  server.on("/connect", HTTP_GET, []() {
    connectToWiFi();
    checkWiFiStatus();
    server.send(200, "text/plain", connectionStatus);
  });
  server.on("/status", HTTP_GET, []() {
    server.send(200, "text/plain", connectionStatus);
  });
  server.on("/submitAP", HTTP_POST, []() {
    handleAPSubmit();
  });
  server.on("/submitCONFIG", HTTP_POST, []() {
    handleCONFIGSubmit();
  });
  server.on("/END", HTTP_POST, []() {
    stopAP();
  });

  startAP();

  i2s_install();
  i2s_setpin();
  i2s_start(I2S_PORT);

  delay(500);
  myPlayer.setBusyPin(4);
  if(!myPlayer.begin()) {
    errCode = 1;
  }
}
```

```
void loop() {

  if (isAPActive) {
    static unsigned long lastLoopCheck = 0;
    static unsigned int lastNumClients = 0;

    server.handleClient();

    if (millis() - lastLoopCheck >= 5000) {
      lastLoopCheck = millis();

      int numClients = WiFi.softAPgetStationNum();

      if (numClients > 0) {
        lastCheckTime = millis();
      }

      if (numClients != lastNumClients) {
        lastNumClients = numClients;
        msgToCom("Кількість клієнтів змінилась. Зараз їх: " +
String(numClients));
      }

      if (millis() - lastCheckTime > timeout) {
        stopAP();
      }
    }
  } else if (!isConnectedToInternet) {
    if (errCnt >= 12 || firstTry) {
      msgToCom("Нова спроба підключитись до роутера.");
      connectToWiFi();
      if (firstTry) {
        firstTry = false;
      }
    }

  } else if (!isAPActive) {
    if (isConnectedToInternet && !waiting) {
      client.loop();
      if (isWebSocketConnected) {
        if (millis() - lastWebSocketActivity > WS_ACTIVITY_TIMEOUT) {
        isWebSocketConnected = false;
          wsReconnectTimer = millis();
          waiting = true;
          msgToCom("Зв'язок з сервером втрачено. Намагаюсь
перепідключитись.");
          client.disconnect();
        }
        else if(errCode != -1) {
          sendError();
        }
        else if(!myPlayer.isPlaying() && !ff) {
          listening();
        }

      }
    }

    if (isConnectedToInternet && !isWebSocketConnected &&
!blockReconnect) {

      if (millis() - wsReconnectTimer > WEBSOCKET_RECONECT_TIME) {  //
15 секунд
        wsReconnectTimer = millis();
        msgToCom("🔄 Встановлюємо WebSocket-з'єднання...");
        waiting = false;
        setupWebSocket();
        ++websocket_tries;
```

```
        if (websocket_tries >= 6) {
          ESP.restart();
        }
      }
    }

    if (ff && millis() - configEND >= 30000) {
      call_lift_sim();
      ff = false;
    }
  }



  if (startCheckStatus) {
    if (millis() - timerToShowStatus >= 5000) {
      checkWiFiStatus();
    }
  }
}
```

PlayerController.h

```
#ifndef PLAYER_CONTROLLER_H
#define PLAYER_CONTROLLER_H

#include <DFRobotDFPlayerMini.h>

class PlayerController {
public:
    PlayerController(HardwareSerial &serial, uint8_t rxPin, uint8_t
txPin);

    bool begin();
    void playTrack(uint16_t trackNumber);
    void setVolume(uint8_t volume);
    void setBusyPin(uint8_t pin);
    bool isPlaying();

private:
    HardwareSerial &serialPort;
    DFRobotDFPlayerMini player;
    uint8_t rx;
    uint8_t tx;
    uint8_t busyPin = 255;
};

#endif
```

PlayerController.cpp

```
#include "PlayerController.h"

PlayerController::PlayerController(HardwareSerial &serial, uint8_t
rxPin, uint8_t txPin)
    : serialPort(serial), rx(rxPin), tx(txPin) {}

bool PlayerController::begin() {
    serialPort.begin(9600, SERIAL_8N1, rx, tx);
    delay(500);
    if (!player.begin(serialPort, true, false)) {
        return false;
    }
    player.volume(5);
```

```
    if (busyPin != 255) {
        pinMode(busyPin, INPUT);
    }
    return true;
}

void PlayerController::playTrack(uint16_t trackNumber) {
    while (isPlaying()) {
        delay(100);
    }
    player.playMp3Folder(trackNumber);
    delay(1000);
}

void PlayerController::setVolume(uint8_t volume) {
    if (volume >= 0 && volume <= 30)
    {
        player.volume(volume);
    }
    else {
        player.volume(20);
    }

}

void PlayerController::setBusyPin(uint8_t pin) {
    busyPin = pin;
    pinMode(busyPin, INPUT);
}

bool PlayerController::isPlaying() {
    if (busyPin == 255) return false;
    return digitalRead(busyPin) == LOW;
}
```

ElevatorCommands.h

```
#ifndef ELEVATOR_COMMANDS_H
#define ELEVATOR_COMMANDS_H

#include <Arduino.h>

class ElevatorCommand {
public:
  virtual void execute() = 0;
  virtual ~ElevatorCommand() {}
};

class OpenDoorCommand : public ElevatorCommand {
public:
  void execute() override;
};

class CloseDoorCommand : public ElevatorCommand {
public:
  void execute() override;
};

class GoToFloorCommand : public ElevatorCommand {
private:
  int targetFloor;
public:
  GoToFloorCommand(int floor);
  void execute() override;
};

#endif
```

ElevatorCommands.cpp

```cpp
#include "ElevatorCommands.h"

void OpenDoorCommand::execute() {
  Serial.println("Відкриваю двері...");
}

void CloseDoorCommand::execute() {
  Serial.println("Закриваю двері...");
}

GoToFloorCommand::GoToFloorCommand(int floor) : targetFloor(floor) {}

void GoToFloorCommand::execute() {
  Serial.print("Емулюю натискання клавіші поверху ");
  Serial.println(targetFloor);
}
```

main.py

```python
from fastapi import FastAPI, WebSocket, WebSocketDisconnect
from vosk import Model, KaldiRecognizer
import json
import wave
import os
import tempfile
import time

from llm.llm import LLM

app = FastAPI()

SAMPLE_RATE = 16000
SAMPLE_WIDTH = 2
CHANNELS = 1

model = Model("vosk-model-small-uk-v3-small")
llm = LLM()

def print_latency(start_time: float):
    elapsed = time.perf_counter() - start_time
    print(f"Час обробки аудіо: {elapsed:.2f} секунд")

@app.websocket("/ws")
async def websocket_endpoint(websocket: WebSocket):
    await websocket.accept()

    mode = "activation"
    wf = None
    number = None

    # Створення першого тимчасового файлу
    tmp_file = tempfile.NamedTemporaryFile(suffix=".wav", delete=False)
    temp_audio_file = tmp_file.name
    tmp_file.close()

    try:
        wf = wave.open(temp_audio_file, 'wb')
        wf.setnchannels(CHANNELS)
        wf.setsampwidth(SAMPLE_WIDTH)
        wf.setframerate(SAMPLE_RATE)

        try:
            # Авторизация
            message = await websocket.receive_text()
```

```
            try:
                data = json.loads(message)
                floors = data.get("floors")
                number = data.get("number")

                if not floors or not number or number ==
"380#########":
                    print(f"❌ Невірний формат авторизаційних данних:
{data}")
                    await websocket.send_text("__AUTH_FAILED__")
                    await websocket.close()
                    return

                await websocket.send_text("__AUTH_SUCCESS__")

            except json.JSONDecodeError as e:
                print(f"❌ Помилка декодування JSON: {e}")
                await websocket.send_text("__AUTH_FAILED__")
                await websocket.close()
                return
            print(f"✅ Клієнт підключився та авторизувався. Надсилаю
повідомлення ліфтеру по номеру: {number}")
            # Основний цикл
            while True:
                try:
                    message = await websocket.receive()
                except RuntimeError:
                    raise WebSocketDisconnect

                if "bytes" in message:
                    wf.writeframes(message["bytes"])

                elif "text" in message:
                    text_data = message["text"].strip()

                    if text_data == "__END__":
                        start_analyze = time.perf_counter()
                        print("📥 Завершення аудіо. Обробка...")

                        wf.close()
                        recognizer = KaldiRecognizer(model,
SAMPLE_RATE)
                        recognizer.SetWords(True)

                        with open(temp_audio_file, "rb") as f:
                            while chunk := f.read(4000):
                                recognizer.AcceptWaveform(chunk)

                        result = json.loads(recognizer.FinalResult())
                        text = result.get("text", "").strip()
                        if text:
                            print(f"📝 Розпізнано: {text}")

                            if mode == "activation":
                                if "привіт ліфт" in text.lower():
                                    print("📝 Активаційна фраза
розпізнана. Увімкнено режим команди.")
                                    print_latency(start_analyze)
                                    await
websocket.send_text("__ACTIVE__")
                                    mode = "command"
                                else:
                                    print_latency(start_analyze)
                                    await
websocket.send_text("__NOTHING__")
                            elif mode == "command":
```

```
                                res = llm.get_command(floors, text)
                                print(f"📝 Команда: {res}")
                                print_latency(start_analyze)
                                if "Помилка" in res:
                                    await
websocket.send_text("__RECOGNITION_FAILED__")
                                else:
                                    await websocket.send_text(res)
                                    mode = "acknowledgement"

                            elif mode == "acknowledgement":
                                if "так" in text.lower():
                                    print(f"📝 Команду підтверджено")
                                    print_latency(start_analyze)
                                    await
websocket.send_text("__YES__")
                                    mode = "command"
                                elif "ні" in text.lower():
                                    print(f"📝 Команду відхилено")
                                    print_latency(start_analyze)
                                    await websocket.send_text("__NO__")
                                    mode = "command"
                                else:
                                    print(f"📝 Невірне підтвердження")
                                    print_latency(start_analyze)
                                    await
websocket.send_text("__ACK_FAILED__")
                            else:
                                print(f"📝 Слів не розпізнано")

                                # Видалити старий файл і створити новий
                            os.remove(temp_audio_file)
                            tmp_file =
tempfile.NamedTemporaryFile(suffix=".wav", delete=False)
                            temp_audio_file = tmp_file.name
                            tmp_file.close()

                            wf = wave.open(temp_audio_file, 'wb')
                            wf.setnchannels(CHANNELS)
                            wf.setsampwidth(SAMPLE_WIDTH)
                            wf.setframerate(SAMPLE_RATE)

                        elif text_data == "__RUN__":
                            mode = "activation"
                        elif text_data == "__START__":
                            mode = "activation"

                        elif text_data == "DFPlayerERROR" :
                            print(f"❌ Помилка підключення до DFPlayer.
Надсилаю повідомлення ліфтеру по номеру: {number}")
                            await websocket.close()
                            return

                except WebSocketDisconnect:
                    print(f"❌ Клієнт відключився. Надсилаю повідомлення
ліфтеру по номеру: {number}")

                except Exception as e:
                    print(f"⚠️ Сталася помилка: {e}")

            finally:
                if wf:
                    try:
                        wf.close()
                    except:
```

```
                pass
        if os.path.exists(temp_audio_file):
            os.remove(temp_audio_file)
        print("🔌 З'єднання закрите.")
```

llm.py

```
from openai import OpenAI
import os
from dotenv import load_dotenv

class LLM:
    def __init__(self, model: str = "gpt-4o-mini"):
        load_dotenv()
        self.model = model
        self.client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))

    def get_command(self, floors, text):
        COMMANDS = [
            "скинути команди",
            "список команд",
            "поїхали",
            "відчинити двері",
            "зачинити двері",
            "викликати ліфтера",
            *[f"{i}-й поверх" for i in range(1, floors+1)]
        ]

        prompt = (
            f"Тобі надано список допустимих команд:\n"
            f"{'\n'.join(COMMANDS)}\n\n"
            f"Проаналізуй текст користувача та вибери з цього списку ту
команду, "
            f"на яку він найбільше схожий. Відповідай **лише самою
командою** зі списку, без пояснень. "
            f"Якщо жодна команда не підходить — поверни 'Помилка'.\n\n"
            f"Текст: {text}"
        )

        try:
            response = self.client.chat.completions.create(
                model=self.model,
                messages=[
                    {"role": "system", "content": "Ти помічник, що
керує ліфтом."},
                    {"role": "user", "content": prompt}
                ],
                temperature=0.5,
            )
            return response.choices[0].message.content
        except Exception as e:
            return f"Помилка: {str(e)}"
```