Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____Олексій СМІРНОВ
“____” _____2025 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА

за першим (бакалаврським) рівнем вищої освіти

на тему

“ Програмне забезпечення системи кібербезпеки захисту приватності розмови шляхом генерування шуму”

Виконав здобувач вищої освіти
IV курсу, групи КБ-21
ОПП «Кібербезпека»
спеціальності 125
«Кібербезпека»

_____Задорожний К.О.
«____» _____2025 р.

Керівник проекту кандидат
технічних наук, доцент

_____Дрєєв О.М.
«____» _____2025 р.

Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет
Факультет *Механіко-технологічний*
Кафедра *Кібербезпеки та програмного забезпечення*
Рівень вищої освіти *бакалавр*
Галузь знань ._12 *“*Інформаційні технології*”*
Спеціальність *125 “*Кібербезпека*”*
Освітньо-професійна (освітньо-наукова) програма _*“* Кібербезпека *”*_

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____Олексій СМІРНОВ
“____” _____2025 року

ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ ЗА ПЕРШИМ (БАКАЛАВРСЬКИМ) РІВНЕМ ВИЩОЇ ОСВІТИ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Задорожний Костянтин Олександрович
(прізвище, ім’я, по батькові)

1. Тема роботи *Програмне забезпечення системи кібербезпеки захисту приватності розмови шляхом генерування шуму*
2. Керівник роботи *Дрєєв Олександр Миколайович, канд. техн. наук, доцент*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом закладу вищої освіти від “ 1 7 ” січня 2025 року №57-02
3. Строк подання роботи до захисту *22.05.2025 р.*
4. Мета та завдання випускної кваліфікаційної роботи *Метою роботи є розробка програмного забезпечення системи кібербезпеки захисту приватності розмови шляхом генерування шуму*
5. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
   1. *Призначення та область використання.*
   2. *Перегляд аналогічних існуючих систем.*
   3. *Опис і обґрунтування проектних рішень.*
   4. *Етапи програмування системи.*
   5. *Впровадження системи в промислову експлуатацію*
   6. *Висновки.*
6. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

Структурна схема системи	*1 аркуш*
Функціональна схема системи	*1 аркуш*
Діаграма процесів	*1 аркуш*
Принципова схема	*1 аркуш*
Блок-схема алгоритму роботи додатку	*5 аркушів*

7. Дата видачі завдання « 17 » січня 2025р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Строк виконання етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Примітка
1.	Аналіз існуючих систем керування	10.03.2025 р.	
2.	Постановка задачі, оформлення ТЗ	15.03.2025 р.	
3.	Розробка моделі компонента	20.03.2025 р.	
4.	Розробка структур даних	25.03.2025 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.03.2025 р.	
6.	Програмування алгоритмів	10.04.2025 р.	
7.	Оформлення ПЗ	17.05.2025 р.	
8.	Попередній захист роботи	22.05.2025 р.	

Дата видачі завдання
« 17 » січня 2025 р.

Підпис керівника

_____ _____
(прізвище та ініціали)

Завдання прийнято до виконання
« 17 » січня 2025 р.

Підпис здобувача

_____ _____
(прізвище та ініціали)

АНОТАЦІЯ

Задорожний К.О. Програмне забезпечення системи кібербезпеки захисту приватності розмови шляхом генерування шуму. 125 Кібербезпека. Центральноукраїнський національний технічний університет. Кропивницький. 2025.

У даній кваліфікаційній бакалаврській було розроблено програмне забезпечення системи кібербезпеки, призначене для захисту приватності розмови шляхом генерування шуму.

Метою розробки є система кібербезпеки та її програмного забезпечення для захисту приватності розмови шляхом генерування шуму.

В процесі роботи було вивчено нормативно-правову базу забезпечення захисту інформації з обмеженим доступом в Україні, проаналізовано канали витоку акустичної інформації та вимоги до технічних засобів захисту інформації через акустичні канали. В повній мірі описані всі компоненти розробленого програмного забезпечення.

Система працює шляхом генерування спеціально сформованого шумового сигналу, який додається до звукового поля в приміщенні, маскуючи мову. Реалізація захисту базується на мікроконтролері ESP32 та сучасних технологіях обробки сигналів, а програмне забезпечення розроблено з використанням мови програмування C++.

В ході виконання роботи було досягнуто поставленої мети та вирішено ряд важливих завдань. Розроблена система кібербезпеки забезпечує ефективний захист приватності розмови шляхом генерування шуму. Система може бути використана в різних сферах діяльності, де потрібен захист конфіденційності усного спілкування.

Ключові слова: кібербезпека, акустична інформація, захист приватності, генерування шуму, програмне забезпечення.

ABSTRACT

Zadorozhnyi K.O. Software for a Cybersecurity System for Protecting Conversation Privacy by Generating Noise. 125 Cybersecurity. Central Ukrainian National Technical University. Kropyvnytskyi. 2025.

This bachelor's qualification thesis presents the development of software for a cybersecurity system designed to protect the privacy of conversations through the generation of noise.

The aim of the project is to develop a cybersecurity system and its software to ensure conversation privacy by masking speech with generated noise.

The research involved an analysis of the legal and regulatory framework for the protection of restricted-access information in Ukraine, identification of acoustic information leakage channels, and examination of technical requirements for information protection through acoustic channels. All components of the developed software are described in detail.

The system operates by generating a specially shaped noise signal that is added to the sound field in a room, effectively masking speech. The solution is based on the ESP32 microcontroller and modern signal processing technologies, with the software developed in C++.

The objectives of the work were successfully achieved, and several important tasks were solved. The developed cybersecurity system provides effective protection of conversation privacy through noise generation. It can be applied in various fields where oral communication confidentiality is required.

Keywords: cybersecurity, acoustic information, privacy protection, noise generation, software.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ І СКОРОЧЕНЬ

ЗЗІ – Засоби захисту інформації

ІБ – інформаційна безпека

ПЗ – Програмне забезпечення

ТЗІ – Технічний захист інформації

ОІД – об’єкт інформаційної діяльності

КІ – конфіденційна інформація

НПА – нормативно-правовий акт

IoT – Інтернет речей (Internet of Things)

ESP32 - Мікроконтролер з інтегрованим Wi-Fi та Bluetooth

SD-карта - Карта пам’яті формату SD

I^2S - Inter-IC Sound

WAV-файл - Файл аудіо формату WAV

VCC - Voltage at the Common Collector

GND - Ground

MISO - Master In Slave Out

MOSI - Master Out Slave In

SCK - Serial Clock

CS - Chip Select

ВСТУП

Актуальність теми. Володіння та обмін інформацією являються ключовими факторами прогресу сучасної держави, оскільки дозволяє приймати оптимальні рішення; здійснювати аналіз даних з метою своєчасного реагування на різноманітні виклики; проводити моніторинг важливих соціально-економічних показників та прогнозувати майбутні кроки розвитку різних сфер діяльності. Але не менш важливою є проблема захисту інформації від несанкціонованого втручання.

Засобами передачі інформації являються усні, писемні, графічні відомості та відеоповідомлення, які реалізуються за допомогою сучасних інформаційно-комунікаційних технологій.

Незважаючи на широке впровадження автоматизованих та комп'ютеризованих систем обробки даних, людська мова залишається одним з найважливіших способів інформаційної взаємодії. А в умовах децентралізації економічних і політичних систем, а також збільшення частки оперативної інформації, мовний обмін набуває ще більшої значущості, особливо під час потреби швидкого реагування на певні виклики, що загрожують цілісності, стабільності та розвитку як державного устрою, так і окремих приватних, фінансових чи адміністративних структур. Тому захист акустичної (мовної) інформації потребує особливої уваги.

До 47% всього витоку конфіденційної інформації здійснюється через технічні канали витоку інформації, що робить проблему технічного захисту інформації надзвичайно актуальною [18].

Однією з основних загроз безпечному спілкуванню є перехоплення мовної інформації різними методами, включаючи прослуховування телефонних дзвінків, нарадчих кімнат, розмов на переговорах чи співбесідах, доповідей на закритих конференціях тощо.

На сьогодні існує велика кількість пристроїв для перехоплення мовної інформації, таких як: спрямовані мікрофони, диктофони, стетоскопи, радіозакладки тощо. Ці прилади важко виявити через їхню непомітність. Відтак, протидія цим пристроям у вигляді створення системи кібербезпеки захисту акустичної інформації є важливим аспектом забезпечення інформаційної безпеки приватних розмов. Отже, тема «**Програмне забезпечення системи кібербезпеки захисту приватності розмови шляхом генерування шуму**» є надзвичайно актуальною в сучасних умовах інформаційної політики держави.

Мета роботи полягає у розробці системи кібербезпеки та її програмного забезпечення для захисту приватності розмови шляхом генерування шуму. Для досягнення цієї мети було проведено вивчення нормативно-правової бази забезпечення захисту інформації з обмеженим доступом в Україні, здійснено аналіз каналів витоку акустичної інформації та вимог до технічних засобів захисту інформації через акустичні канали. Також було спроєктовано систему кібербезпеки технічного захисту мовної інформації шляхом генерування шуму, а на завершальному етапі розроблено та протестовано програмне забезпечення цієї системи.

Об'єктом дослідження являються методи та засоби технічного захисту акустичної інформації.

Предметом дослідження є програмне забезпечення технічного засобу захисту акустичної інформації приватної розмови, який побудований на базі генерації шуму.

Методи дослідження охоплюють аналіз існуючих методів захисту акустичної інформації, проектування, програмування та реалізацію генератора шуму, а також тестування його ефективності в умовах, наближених до реальних сценаріїв використання.

Структура роботи включає вступ, чотири основні розділи, висновки, додатки та список використаних джерел. Загальний обсяг роботи становить 75 сторінки.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

Розроблена система кібербезпеки має на меті забезпечення захисту особистих даних шляхом збереження конфіденційності голосових розмов.

В умовах стрімкого розвитку технологій прослуховування зростає актуальність розробки ефективних засобів протидії несанкціонованому доступу до інформації. Запропоноване рішення ґрунтується на застосуванні технологій генерації шуму та технологій Інтернет речей (IoT - Internet of Things) для зручного управління відтворенням акустичного фону в дистанційному режимі, що значно ускладнюють можливість запису та прослуховування, тим самим підвищуючи рівень приватності спілкування.

1.1 Призначення системи

Рішення у сфері кібербезпеки, розроблене для збереження конфіденційності усного спілкування за рахунок застосування шумових перешкод, має за основне завдання створення звукового фону, подібного до людської мови, а також відтворення ультразвукових хвиль, що блокує можливість запису або аналізу розмов третіми особами чи пристроями. Завдяки спеціально генерованим акустичним сигналам, голосові повідомлення стають нерозбірливими та непридатними для подальшої обробки.

Призначення розробленої системи полягає в надійному рівні захищеності усної вербальної інформації під час спілкування в різноманітних умовах. Вона ефективно запобігає спробам прихованого прослуховування, тим самим зменшуючи ймовірність розголошення чутливої інформації. Завдяки сучасному технічному оснащенню, це рішення забезпечує високий рівень безпеки для приватного спілкування.

1.2 Область застосування

Область застосування технологій захисту інформації з обмеженим доступом визначається нормативно-правовими актами (НПА) України, які встановлюють вимоги й рекомендації щодо запобігання витоку, спотворення, блокування чи знищення інформації різними каналами.

Правову основу області застосування системи кібербезпеки захисту акустичної інформації при приватній розмові становлять:

1) Конституція України [29]: надання гарантій таємниці телефонних розмов (стаття 31); заборона зібрання, зберігання, використання та поширення конфіденційної інформації про особу без її згоди (стаття 32); гарантія права на вільний доступ до інформації, її зберігання та поширення за виключенням окремих випадків державної таємниці (стаття 34);

2) Закони України:

2.1) «Про інформацію» [20]. Цей закон встановлює загальні правові основи інформаційної діяльності та визначає види інформації з обмеженим доступом.

2.2) «Про доступ до публічної інформації» [22]: цей закон регулює порядок доступу та обмеження до публічної інформації. Зокрема, стаття 7 визначає, що розпорядники інформації можуть поширювати конфіденційну інформацію лише за згодою осіб, які обмежили доступ до неї, або в інтересах національної безпеки, економічного добробуту та прав людини.

2.3) «Про захист персональних даних». Цей закон визначає правові засади захисту персональних даних та інформації, що дозволяє ідентифікувати фізичну особу.

2.4) «Про державну таємницю» [31]: встановлюються категорії інформації, яка підлягає захисту та методи її захисту.

2.5) «Про науково-технічну інформацію» [32] та інші НПА та міжнародні договори у сфері інформаційних відносин.

Відповідно до зазначених законів України до відомостей, які мають обмежений доступ, а отже підлягають захисту, відносяться: таємна, конфіденційна та службова інформація [20, 22, 42].

В залежності від режиму доступу до інформації, розрізняють відкриту інформацію та інформацію з обмеженим доступом (рис.1.1).

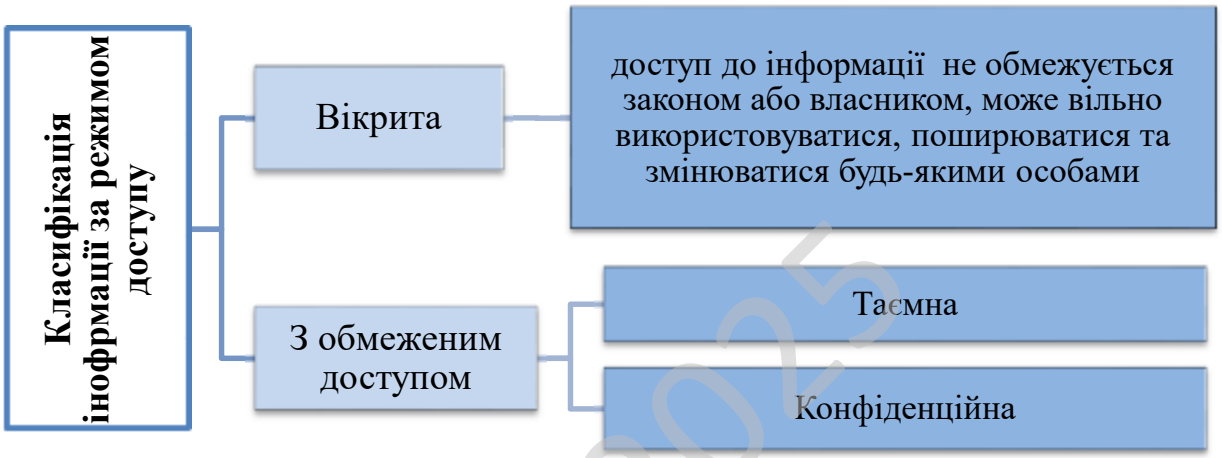

Рисунок 1.1 – Класифікація інформації за режимом доступу

До конфіденційної інформації (КІ) відносяться відомості різного характеру (ділового, банківського, виробничого, комерційного тощо), які знаходяться у володінні, користуванні або розпорядженні фізичних/юридичних осіб, які самостійно визначать інформацію як конфіденційну, поширюють її лише за власним бажанням та встановлюють для неї систему захисту [20].

До таємної інформації належить інформація, що містить відомості, які становлять державну та іншу передбачену законом таємницю, розголошення якої завдає шкоди особі, суспільству або державі. Таємною визнається інформація, яка містить державну, професійну, банківську, розвідувальну таємницю, таємницю досудового розслідування та іншу, передбачену законом таємницю (лікарську, адвокатську, військову, таємницю нарадчої кімнати та інші) [22].

До службової інформації відноситься [22, ст. 9]:

а) інформація документів внутрішньовідомчої службової кореспонденції та доповідних записок;

б) інформація, зібрана в процесі оперативно-розшукової, розвідувальної діяльності, у сфері оборони країни, яку не віднесено до державної таємниці.

Усними формами ділового конфіденційного (або таємного) спілкування являються: бесіда, прийом, доповідь, розмова по телефону, нарада та переговори, які в своїй сукупності складають велику частку робочого часу [25].

Дослідження, які висвітлюють окремі аспекти питань частки усної ділової комунікації та відсотка витоків КІ через мовленнєві канали, показують, що менеджери витрачають на розмови з людьми 50% - 90% свого робочого часу [24, С.265], а також зазначається, що перевагою усних комунікацій являється забезпечення негайного зворотного зв'язку та взаємного обміну інформацією, а це, в свою чергу, підвищує ризик несанкціонованого прослуховування.

Ці дані свідчать про значну роль усної комунікації в діловому середовищі та підкреслюють важливість захисту усної мовної інформації для забезпечення особистої, корпоративної та державної безпеки в цілому.

Отже, опираючись на вище зазначені положення нормативно-правової бази забезпечення ЗКІ та зміст поняття інформації з обмеженим доступом, можна відмітити те, що область застосування системи кібербезпеки захисту усного спілкування за допомогою створення шумових перешкод є надзвичайно широкою та охоплює різні сфери, де конфіденційність відіграє вирішальну роль, а саме:

– Бізнес та підприємництво: охорона комерційної таємниці, фінансової інформації, витоку стратегічних планів. У корпоративному середовищі цей інструмент може застосовуватись під час ділових зустрічей, стратегічного планування, презентацій нових розробок або під час обговорення фінансових таємниць — ситуацій, де витік даних може мати серйозні наслідки та загрожувати конкурентній позиції компанії.

– Банківська справа: охорона банківської таємниці та інформації з приводу фінансових операцій клієнтів.

– Страхування та соціальний захист: охорона персональних даних клієнтів, страхові випадки.

– Юридична практика: охорона адвокатської таємниці та інформації під час конфіденційних консультацій. Розроблена система стане важливим елементом безпеки під час особистих консультацій клієнта з адвокатом, обговорення справ або підготовки до слухань, де суворе збереження таємниці є стандартом професійної етики.

– Медицина: охорона лікарської таємниці та медичних даних пацієнтів. Розроблена система допоможе забезпечити приватність діалогу між лікарем і пацієнтом, зокрема при обговорені чутливої медичної інформації – діагнозу, результатів аналізів або узгодженні методів лікування.

– Державне управління: охорона службової інформації та державної таємниці. Державні структури та урядові органи можуть використовувати цю систему для захисту внутрішніх обговорень, формування стратегій або ведення переговорів міжнародного рівня, де розголошення інформації може становити загрозу національній безпеці.

– Судова гілка влади: процес обговорення та ухвалення рішення в суді є конфіденційним (таємниця нарадчої кімнати). Розроблена система може слугувати додатковим захистом приватності процесу ухвалення рішення.

– Освіта та наука: охорона інтелектуальної власності: розроблена система може слугувати захистом наукової інформації при обговоренні результатів наукових досліджень, лабораторних випробувань тощо на закритих конференціях, семінарах чи зустрічах.

– Міжособистісне спілкування. Пристрій буде корисним для окремих осіб, які хочуть убезпечити своє приватне спілкування в публічних місцях, у транспорті або під час подорожей, де існує ризик зовнішнього втручання.

Завдяки зручній формі, мобільності та високій ефективності, система легко інтегрується в різні середовища забезпечуючи надійний захист голосових комунікацій в умовах сучасних викликів цифрової безпеки.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1 Огляд існуючих систем, технологій, архітектур та програмних рішень

В умовах зростаючої важливості інформаційної безпеки, забезпечення конфіденційності усного спілкування набуває особливої актуальності. Сучасні технології, зокрема спрямовані мікрофони та лазерні системи зчитування, суттєво знижують ефективність традиційних засобів захисту. Це створює потребу у впровадженні інноваційних підходів, серед яких — активні рішення на основі акустичних перешкод, що перешкоджають несанкціонованому запису або прослуховуванню розмов.

Огляд основних методів та засобів технічного захисту акустичної інформації від витоків

Технічний захист інформації (ТЗІ) здійснюється за допомогою інженерно-технічних заходів з метою запобігання витоку інформації та збереження її конфіденційності, цілісності та доступності (рис.2.1) [19, 23].

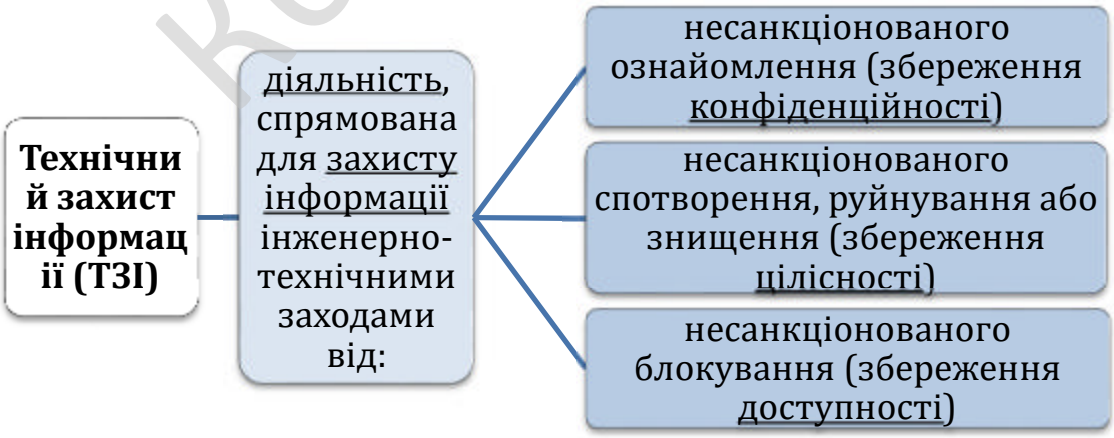

Рисунок 2.1 – Технічний захист інформації

До основних засобів ТЗІ відносять фізичні засоби, апаратні засоби, програмні засоби, криптографічні та їх поєднання [36, 37] (рис.2.2).

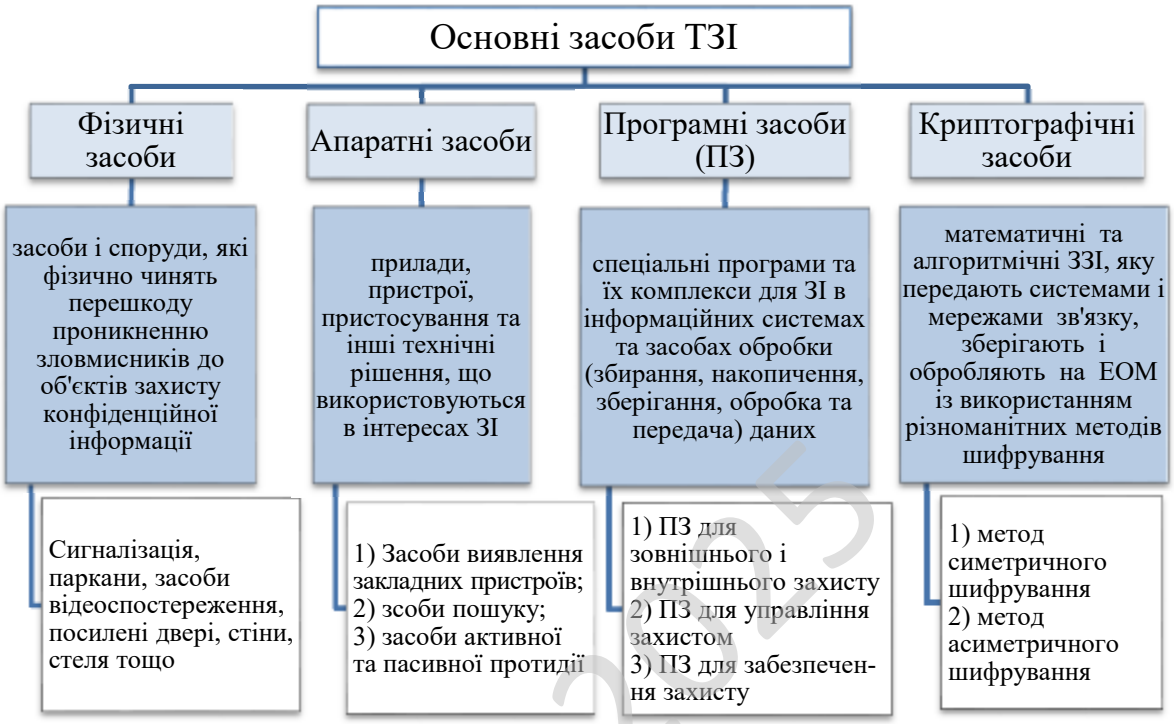

Рисунок 2.2 – Засоби інженерно-технічного захисту інформації

Для захисту усної мовної інформації розробляють комплексну систему захисту, яка складається з цілого ряду спеціальних методів активної та пасивної протидії витокам акустичної інформації (рис.2.3) [42].

Активні методи захисту акустичної інформації включають в себе:

1) Просторове зашумлення – передбачає створення рівномірного звукового або електромагнітного фону в приміщенні. Досягається використанням наступних засобів:

– генераторів просторового електромагнітного шуму;

– технічних засобів акустичного маскування з використанням генераторів та випромінювачів акустичних і віброакустичних завад, які створюють різні види шумів – білий, рожевий та інші;

– генераторів інфразвукових або ультразвукових коливань, які заважають запису звуків на диктофони.

Рисунок 2.3 – Система захисту мовної інформації

2) Лінійне зашумлення:

– технічні засоби захисту інформації, яка передається по провідних лініях (мережах електроживлення, кіл заземлення, телефонних, аудіо-, відео- та інформаційних кабелях). Даний вид захисту запобігає перехопленню сигналів із проводів або через радіочастотне випромінювання та маскує корисний сигнал в провідній лінії широкосмуговим електричним шумом чи високочастотними сигналами, які ускладнюють отримання мовної інформації.

3) Заходи щодо перетворення мовних сигналів у каналах зв'язку:

– цифрові або аналогові скремблери – змінюють частоти, біти або порядок мовного сигналу, роблячи його нерозбірливим;

– апаратне чи програмне шифрування даних;

4) Знешкодження закладних пристроїв: використання генераторів імпульсів та засобів заглушення:

– акустичні, ультразвукові, електромагнітні заглушувачі – для заглушення мікрофонів;

– електромагнітні блокіратори;

– заглушувачі для бездротових каналів – для блокування роботи радіомікрофонів і прослуховучих пристроїв.

Технічний засіб активного захисту мовної інформації (ТЗ АЗМІ) – це засіб, основними складовими частинами якого є:

1) генератор електричного коливання;

2) випромінювач, за допомогою якого електричні коливання засобу активного захисту перетворюються в акустичні та віброакустичні [39].

Класифікація засобів активного захисту мовної інформації відбувається за різними ознаками (табл. 2.1) [39].

Таблиця 2.1 – Класифікація засобів активного захисту мовної інформації

Ознака класифікації	Види засобів активного захисту мовної інформації
Тип випромінювача	Акустичні, Віброакустичні
Характер розподілення спектра маскуючого сигналу	Засоби білого шуму, Засоби рожевого шуму Засоби генерації інших видів шуму
Метод формування шумового сигналу	Засоби аналогові, Засоби цифрові
Характер напрямленості маскуючого сигналу	Засоби напрямлені, Засоби ненапрямлені

Активні заходи захисту здійснюють акустичне маскування інформації, знижуючи співвідношення сигнал/завада до норми шляхом створення акустичної або віброакустичної завади на межі огороджувальних конструкцій приміщення [16].

Акустичне маскування проводиться шляхом генерування спеціального шумового середовища, яке ускладнює розбірливість голосу та його обробку пристроями запису (диктофонами, мікрофонами, системами акустичної розвідки).

Основні типи акустичного маскування зазначені в табл. 2.2.

Таблиця 2.2 – Види акустичного маскування

Тип маскування	Принцип роботи	Приклади пристроїв
Акустичне шумове маскування	Випромінювачі ТЗ АЗМІ передають шумові акустичні сигнали в повітря, створюючи білий, рожевий чи псевдовипадковий шум, який накладається на голос людини і знижує її розбірливість	Шуман-1, Noise Defender, Базальт-4ГА, МАРС ТЗО-4-2
Вібро-акустичне маскування	Випромінювачі ТЗ АЗМІ передають шумові вібрації на тверді поверхні (стіни, вікна, меблі), створюючи на будівельних конструкціях резонансне зашумлення та роблячи їх джерелами хаотичних звуків, які заважають перехопленню мовлення через контактні мікрофони	Акустик-Блок, VibraShield, Базальт-4ГА, МАРС ТЗО-4-2
Ультра-звукове маскування	Випромінювачі ТЗ АЗМІ генерують ультразвукові хвилі, які впливають на роботу мікрофонів та порушують їхню здатність записувати звук.	Ultrasonic Guard, AntiMic

Також можливе і комбіноване маскування, яке використовує одночасно кілька методів акустичного маскування.

Для надійного захисту КІ від витоку створюється система технічного захисту, під якою розуміється не лише використання матеріально-технічних

засобів захисту КІ на об’єкті інформаційної діяльності (ОІД), але й комплекс організаційних, нормативно-правових та програмних заходів, які запобігають несанкціонованому доступу, витоку, модифікації чи блокуванню КІ [1].

Під об’єктом інформаційної діяльності (ОІД) розуміється приміщення, транспортний засіб, де здійснюється озвучення та (або) обробка технічними засобами інформації з обмеженим доступом [42].

Організаційні заходи регламентують доступ до ОІД, політику безпеки, здійснення контролю та моніторингу діяльності на ОІД – аналіз журналів безпеки, виявлення аномальної активності в інформаційних системах; навчання персоналу з безпечної роботи з інформацією тощо [40].

Огляд технологій генерування та програмування шуму

Для виключення витоку акустичної конфіденційної інформації (КІ) успішно використовуються генератори шуму. Це пристрої, що виробляють звукові перешкоди, рівень шуму яких здатний перекрити сигнал, що захищається. Шпигунські засоби розвідки, які працюють в акустичному та віброакустичному каналі витоку інформації залишаються в робочому стані, але замість мовної інформації особа, яка підслуховує, приймає лише «білий шум». Очистити та застосувати таку інформацію практично неможливо. Головне завдання генератора шуму – створення випадкових і непередбачуваних сигналів та шумів для захисту приватності розмови [8]. Найчастіше акустичні генератори шуму генерують білий, рожевий чи червоний шум.

Шум класифікується за спектральною густиною потужності (Power Spectral Density або скорочено PSD), яка обернено пропорційна частоті сигналу (f), піднесеній до степені $\beta \geq 0$:

$$PSD \sim \frac{1}{f^{\beta}} . \qquad (2.1)$$

Спектральна густина потужності PSD (Вт/Гц) є функцією, що описує розподіл потужності сигналу (Вт) в залежності від його частоти (Гц).

Кожне β-значення відповідає різному кольору шуму. При $\beta = 0$, шум описується як білий; при $\beta = 1$ – як рожевий; при $\beta = 2$ – як червоний [10].

Відповідно до формули 2.1 й умови $\beta = 0$, отримаємо, що білий шум не залежить від частоти сигналу, тобто має однаковий рівень спектральної щільності потужності у всьому діапазоні частот вимірювань (рис.2.4), а також є не корельованим (не залежним) за часом. Інакше говорячи, білий шум має плоский спектр частот у лінійному просторі. Оскільки всі частоти білого шуму мають енергію, то загальна потужність такого сигналу буде дорівнювати нескінченності, що в реальності є неможливим.

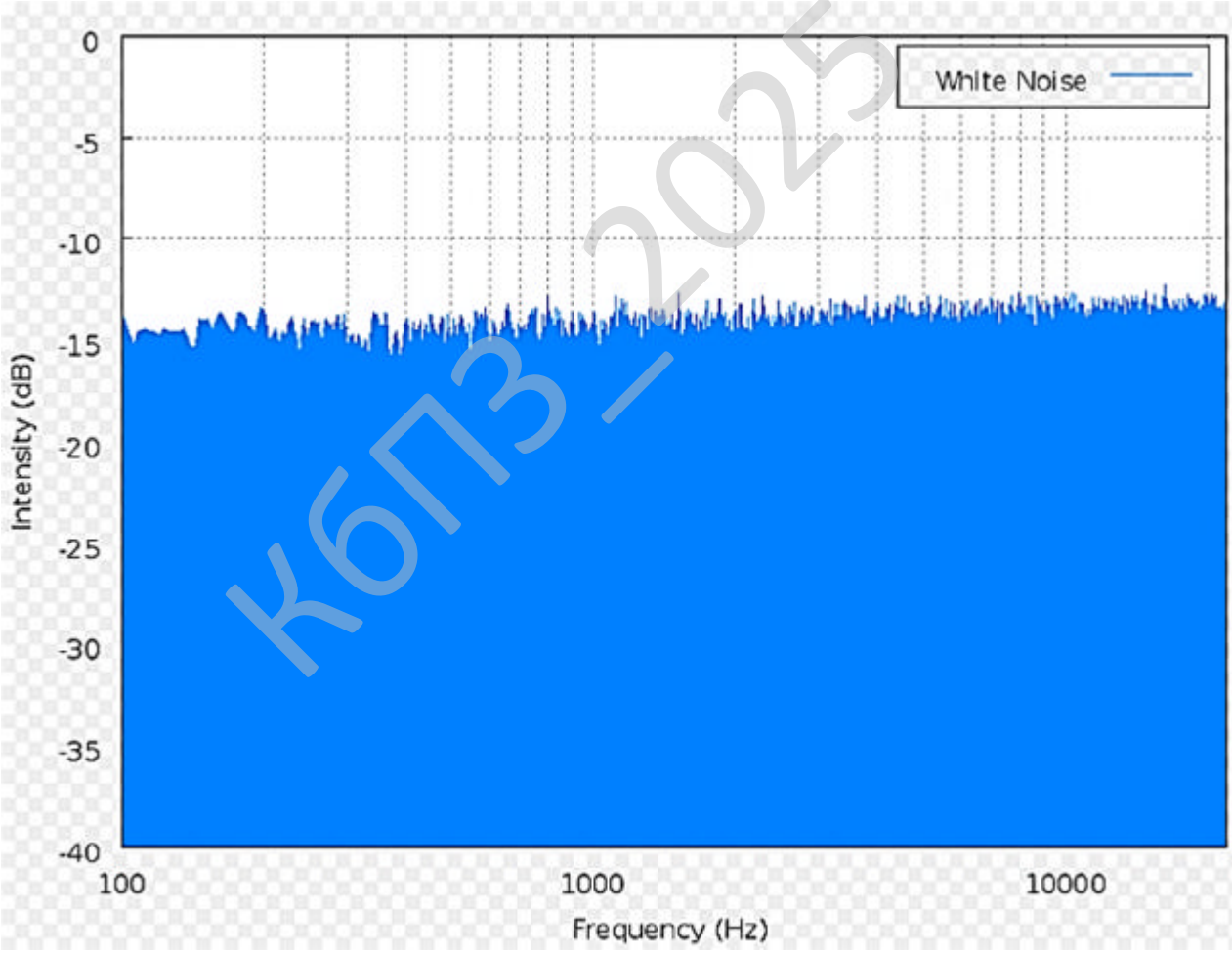

Рисунок 2.4 – Частотний спектр білого шуму [11]

Поняття «білий шум» зазвичай використовується для опису сигналу, автокореляційна функція якого відповідає дельта-функції Дірака в усіх вимірах

багатовимірного простору, де аналізується цей сигнал. Сигнали з такою характеристикою відносять до білого шуму. Ця статистична ознака є визначальною для подібних сигналів. У реальних умовах ідеальний білий шум може бути реалізований лише в обмеженому частотному діапазоні. Білий шум охоплює весь спектр гармонічних частот, що включає частоти, характерні для людського голосу.

Програмна реалізація генерації білого шуму заснована на використанні генератора випадкових чисел, значення (шуми) якого виводяться на цифрові або аналогові піни. Приклад програми для мікроконтролера ESP32, наведений у додатку Б.

При умові $\beta = 1$ в формулі 2.1, отримаємо, що для рожевого шуму спектральна густина потужності обернено пропорційна частоті сигналу (рис.2.5):

$$PSD \sim \frac{1}{f},$$

що означає, що рожевий шум має меншу інтенсивність високих частот порівняно з білим шумом і створює більш природний звуковий фон.

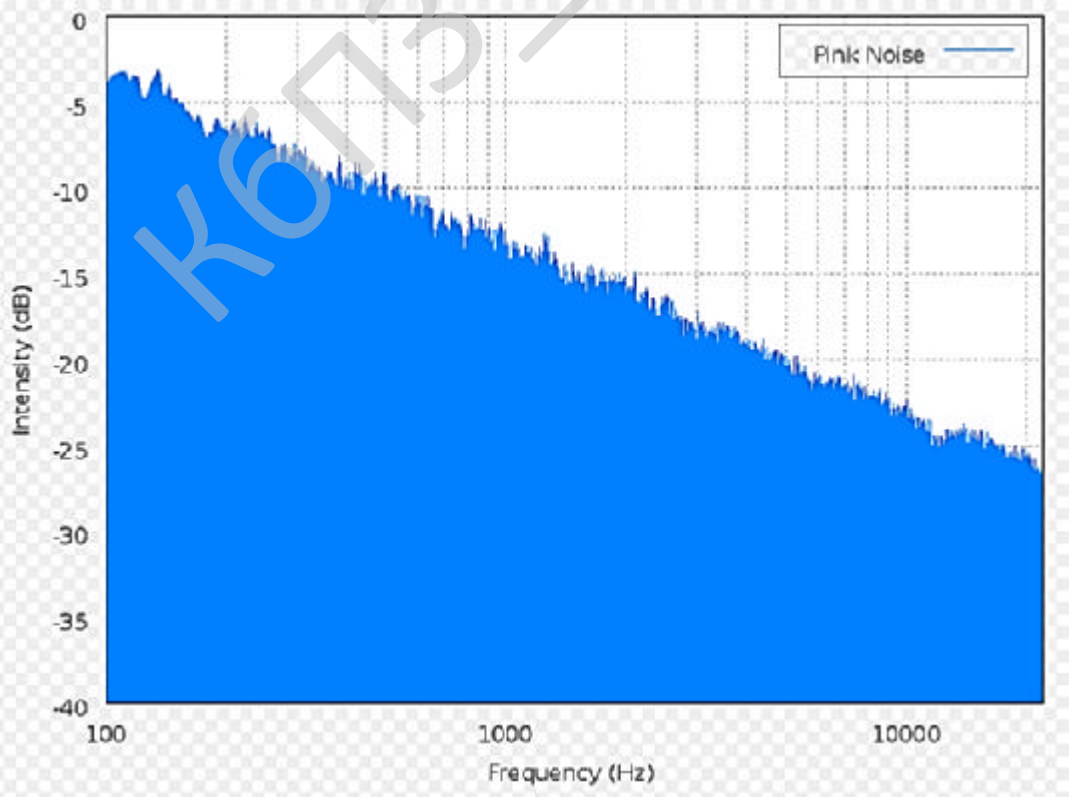

Рисунок 2.5 – Частотний спектр рожевого шуму [12]

Тому рожевий шум підходить для використання в житлових приміщеннях і офісах, де необхідно забезпечити комфортний рівень шуму.

Рожевий шум має плоский спектр частот у логарифмічному просторі. Це означає, що потужність сигналу для проміжку частот 30-40 Гц дорівнює потужності сигналу на частотах 3000–4000 Гц. Спектральна густина такого сигналу в порівнянні з білим шумом загасає на 3 дБ на кожну октаву у бік високих частот. Рожевий шум генерується з білого шуму за допомогою перетворень Фур'є [12].

Програмна реалізація генерації рожевого шуму заснована на алгоритмі Voss-McCartney, який можна описати на мові **C** та використати для мікроконтролерів STM32, ESP32, Raspberry Pi Pico або вбудованих систем.

Даний метод дозволяє отримати рожевий шум шляхом зміни значень випадкових чисел у кількох незалежних потоках із різною швидкістю оновлення. Приклад програми для мікроконтролера ESP32, наведений у додатку Б.

Для червоного шуму спектральна густина потужності обернено пропорційна квадрату частоти сигналу (рис.2.6):

$$PSD \sim \frac{1}{f^2},$$

що означає, що червоний шум має більше потужності на низьких частотах та ще менше потужності на високих частотах у порівнянні з рожевим шумом, а отже являється більш м'яким і затухаючим. Спектральна густина сигналу в порівнянні з білим шумом загасає на 6 дБ на кожну октаву у бік високих частот. Червоний шум підходить для дитячих кімнат або залів для релаксації.

Червоний шум може бути отриманий, якщо проінтегрувати білий шум, або за допомогою алгоритму, що симулює броунівський рух.

Перешкоди від зовнішніх джерел, таких як вібрації від сусідніх машин або фонове світло, в показаннях приладів зазвичай мають червоний малюнок шуму.

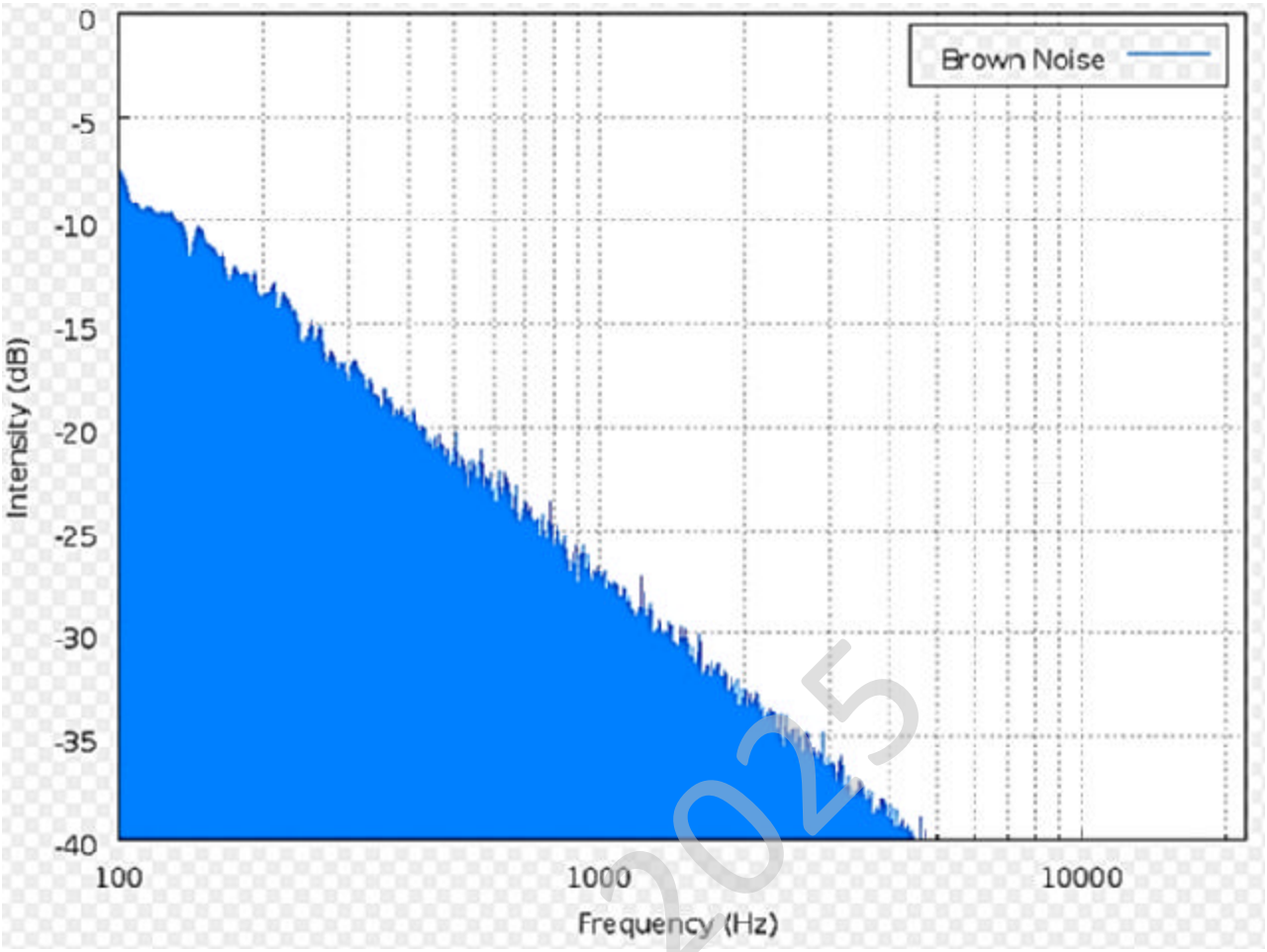

Рисунок 2.6 – Частотний спектр червоного шуму [13]

Програмна реалізація генерації червоного шуму заснована на використанні генератора випадкових чисел мікроконтролера та його інтегруванні, після чого його значення (шуми) виводяться на цифрові або аналогові піни. Приклад програми для мікроконтролера ESP32, наведений у додатку Б.

Ще один важливий і досить ефективний тип генерації шуму – це «мовленевоподібна» перешкода, яка може формуватися двома способами [49]:

1) створюється з фрагментів акустичного сигналу КІ при багатократному накладанні їх з різними рівнями;

2) створюється з некорельованих з акустичним сигналом мовних фрагментів, наприклад:

а) з фрагментів мови 3-х дикторів радіомовних станцій при рівних амплітудах змішуваних сигналів;

б) з фрагменту одного домінуючого мовного сигналу або музичного фрагменту і суміші фрагментів радіопередач із шумом.

Дослідження довели, що найбільш ефективним способом маскування акустичної КІ приватної розмови є перший варіант створення мовленевоподібної перешкоди (реалізований у приладі «PSP-2A AUTO»).

Для захисту приватних розмов, створення мовоподібного шуму на основі мікроконтролера можна реалізувати, використовуючи методи цифрової обробки сигналів (DSP). Це передбачає виконання наступних кроків: 1) Мікроконтролер з потужним DSP (STM32, ESP32, Raspberry Pi Pico тощо) генерує білий або рожевий шум; 2) пропускає його через формантний фільтр; 3) додає варіації тембру, пауз, модуляцій; 4) відтворює отриманий шум через динамік або аудіовихід. Щоб зробити шум схожим на мову, потрібно фільтрувати частоти в спектрі людської мови, а саме в діапазоні 200–6000 Гц.

Фільтрація за допомогою FIR-фільтра реалізується через коефіцієнти формантних фільтрів, які підсилюють частоти, характерні для людської мови.

Програмна реалізація генерації мовленевоподібного шуму для мікроконтролера ESP32, наведена у додатку Б.

Огляд існуючих технічних рішень захисту акустичної інформації шляхом генерування шуму

В залежності від того, який вид каналу витоку акустичної інформації потрібно знешкодити – акустичний, віброакустичних чи оптоелектронний, проводяться відповідні заходи із зниження рівня акустичних або віброакустичних сигналів (тобто озвучення інформації) до певного співвідношення сигнал/завада шляхом створення завади – акустичного шуму та вібрацій. Ступень захисту інформації визначається відповідними нормами.

Основним методом боротьби з несанкціонованим радіоелектронним прослуховуванням являється створення перешкод відповідної інтенсивності у всьому можливому спектрі частот для приймального пристрою.

Універсальною перешкодою являється створення шумових сигналів. Тому апаратура радіопротидії (витяг, обробка, спотворення і придушення інформації) має складатися з наступних елементів: генератор шуму; антенна система.

Генератори шуму в мовному діапазоні використовуються для захисту від несанкціонованого прийому акустичної інформації шляхом маскування безпосередньо корисного звукового сигналу. Маскування проводиться шумом різного кольору.

Одним з найпоширеніших типів є системи, що генерують білий шум, який складається з усіх частот, що звучать одночасно, і створює ефективний бар'єр для маскування розмов. Білий шум часто використовується в офісах і медичних закладах, де важливо забезпечити приватність розмов.

Білий шум зазвичай розглядається у випадках, коли спектр завади охоплює ширший діапазон частот, ніж смуга пропускання використовуваної апаратури, а амплітуди в цьому діапазоні залишаються майже незмінними.

До таких типів завад належать, зокрема, шумові флуктуації та завади, що виникають у багатоканальних системах передачі інформації. [14].

Іншим типом є системи, що генерують рожевий шум. Генератор рожевого шуму – це електронний пристрій або програмне забезпечення, яке створює рожевий шум.

Існує два основних типи генераторів рожевого шуму: аналоговий і цифровий. Аналогові генератори виробляють рожевий шум, імітуючи природні процеси, тоді як цифрові генератори покладаються на алгоритми та фільтри для створення бажаного частотного спектру.

В табл.2.3 наведено типове обладнання для захисту акустичної інформації на основі генерації шуму [16].

Таблиця 2.3 – Обладнання захисту акустичної КІ на основі генерації шуму

Найменування	Опис обладнання
Набір засобів захисту на основі генератору шуму «МАРС ТЗО-4-2»	
 Генератор шуму «МАРС ТЗО-4-2»	Генератор має 2 канали, кожен канал формує два незалежні шумові сигнали. На кожний канал підключається 12-16 випромінювачів. Робочі частоти шумового сигналу – (180 –5600) Гц, звуковий тиск від колонок на відстані 1 м – не менше 80 дБ, споживана потужність – не більше 40 Вт, напруга живлення генератора – (100–240) В частотою 50, 60 Гц [46].
 Вібровипромінювач ВИ-3	Встановлюється по одному (якщо скло великих розмірів – по два) на кожне віконне скло. Робочі частоти –(180 – 5600) Гц, опір обмотки – 50 Ом, максимальна потужність – 1 Вт, рівень звуку на відстані 1 м – 40 дБА [43].
 Вібровипромінювач ВИ-4	Встановлюється, як правило, на радіатори опалення по одному на вхідну/вихідну трубу, а також на стіни, колони та інші масивні конструкції. Робочі частоти –180 – 5600 Гц, опір обмотки – 50 Ом, максимальна потужність – 2 Вт, рівень звуку на відстані 1 м – 40 дБА[44].
 Акустичний випро-мінювач МАРС АК	Встановлюється в закритих порожнинах приміщення. Діапазон частот шумового сигналу: 180 Гц – 5600 Гц; звуковий тиск, що створюється на відстані 1 м в діапазоні робочих частот – не менше 80 дБ, діапазон регулювання рівнів шумових сигналів на виходах – не менше 20 дБ, потужність споживання – до 40 ВА [45].
Акустичний випромінювач МАРС АКЗ	Аналогічний АК, але у захищеному варіанті (захищений від витоку інформації каналом акустоелектричних перетворень) [45].

Продовження таблиці 2.3

Найменування	Опис обладнання
Типовий набір засобів захисту на основі генератору шуму «DNG-2300»	
 Генератор шуму DNG-2300	Трьох канальний генератор шуму: 2 вихідні канали для вібровипромінювачів і 1 канал для акустичних колонок. Канали «вібро» можуть живити від 12 до 24 перетворювачів, встановлених на важких або легких конструкціях відповідно. Канал «акусто» може живити до 12 динаміків. Призначений для захисту периметра приміщення від знімання інформації через стетоскопи, контактні та провідні мікрофони та від витоку через ЗП. Захищає від передавачів, що використовують для передачі інформації мережу 220 В і віконних систем, принцип дії яких заснований на відображенні лазерних, інфрачервоних чи мікрохвильових променів. Недоліки: вага 2,2 кг, ефективний лише для стаціонарного використання; висока вартість – 19276 грн; відсутня можливість автономної роботи.
 Вібровипромінювач TRN-2000	TRN-2000 використовується для захисту від витоку КІ по вібраційному каналу шляхом формування перешкоди в стінах, вікнах, стелі, підлозі і трубах. Один вібровипромінювач захищає цегляну стіну розміром 3×3 м, 1 скло або 1 трубу водопроводу або опалення. Діапазон частот: 250-5000 Гц.
 Акустичний випромінювач OMS-2000	Система з 3-х колонок, які можуть встановлюватися в 1-му місці або ж розподілятися по приміщенню. Створює перешкоду для диктофонів і ЗП в приміщенні. Діапазон частот 250-6500 Гц. Має канал зворотного зв'язку для налаштування рівня звукового тиску [3]. Містить усі гармонічні частоти, присутні в спектрі людського голосу [2]. Захищає простір підвісних стель, вентиляційних коробів, ніш, шаф; встановлюється в вентиляції, за підвісними стелями та в інших порожнинах.

Продовження таблиці 2.3

Найменування	Опис обладнання
Набір засобів захисту на основі генератору шуму «Базальт-4ГА»	
 Генератор шуму «Базальт-4ГА»	Двоканальний генератор електричних шумових сигналів з можливістю поокставного корегування частотної характеристики.
 Вібровипромінювач «Базальт-4ДВМ»	Використовується для віброакустичного зашумлення для захисту об'єктів від витоку мовної інформації віброакустичним каналом. Відсутні зворотні акустоелектричні перетворення. Має засоби кріплення на труби і на скло віконних фрамуг.
 Акустичний випромінювач «Базальт-4ДА»	«Базальт-4ДА» заповнює порожнини приміщення акустичним шумом для запобігання витоку КІ через повітря. Перетворює електричні сигнали, створювані джерелом шуму, в акустичні. Рівень звукового тиску – не менше 63 дБ для октави частотою 4000 Гц та не менше 62 дБ для октави 250 Гц при підведенні шумової напруги 1 В зі спадаючою спектральною щільністю потужності 3 дБ/октаву. Звуковий тиск – до 84 дБ [47].
Типовий набір засобів захисту на основі генератору шуму «ANG-2200	
 Генератору шуму «ANG-2200»	Генератор має 2 канали, кожен з яких контролює високі і низькі частоти з випадковими частотами шуму для перешкоджання роботи дротових мікрофонів в стінах, контактних мікрофонів, аудіопередавачів, розташованих в розетках змінного струму, і лазерних чи мікрохвильових пристроїв. Система маскування генерує шум по периметру приміщення, а не в глиб кабінету, перешкоджаючи ЗП. Може працювати з випромінювачами OMS-2000, «Базальт-4ДА» [48].

Продовження таблиці 2.3

Найменування	Опис обладнання
Набір засобів захисту на основі генератору шуму «Соната-АВ»	
 Генератору шуму «Соната-АВ»	Генератор має 2 канали, на кожен канал можна підключити 12 вібровипромінювачів високої потужності, 16 вібровипромінювачів низької потужності та 16 аудіовипромінювачів. Діапазон частот 300-5000 Гц. Електроживлення з мережі 220 В. Перевищення вібро- та аудіозавади над рівнем мовного сигналу не менш 10 дБ.
Набір засобів захисту на основі генератору шуму «РИАС-2ГС»	
 «РИАС-2ГС»	Стаціонарний генератор акустичного шуму. Призначений для захисту КІ від витоку акустичними та віброакустичними каналами шляхом генерації аналогового шумового сигналу за допомогою лавинного пробою p-n переходу малопотужного транзистора зі зворотним включенням. Генератор забезпечує акустичні сигнали в діапазоні частот 180 Гц – 5,6 кГц. Діапазон регулювання: низько- і високочастотні складові шумового сигналу – не менше 20 дБ; сигнали октавних частот – не менше 12 дБ [51]. Ціна – 22116 грн.
 Випромінювач акустичний «РИАС-2ВА»	«РИАС-2ВА» призначений для захисту об'єктів від витоку КІ віброакустичними каналами шляхом перетворення електричних сигналів в акустичні коливання в діапазоні частот 180-5600 Гц. Вихідна напруга акустичного каналу при навантаженні 6 Ом - не менше 5 В. Максимальний рівень вихідного акустичного сигналу (звукового тиску) в діапазоні робочих частот на відстані 1 м від випромінювача – не менше 70 дБ відносно нульового значення.

Продовження таблиці 2.3

Найменування	Опис обладнання
Портативні генератори шуму	
 Генератор шуму MNG-300 Rabbler	Розмір генератора 83x51x22 мм. Шумоподібні перешкоди створюються за допомогою реальних мовних компонентів і ідентичні шуму натовпу в громадських місцях. Цей тип шуму найбільш ефективний при створенні глушіння диктофонів і ЗП. Недоліки: невеликий діапазон частот: 300 – 2000 Гц; джерело живлення від крони 9В, що означає, що пристрій швидко розряджається; ціна – 8 906 грн.
 Генератор шуму USPD	Генератор розміром 18x10cx4см. Генерує ультразвукові і звукові перешкоди, які корелюються з мовою людини. Пристрій має 16 ультразвукових випромінювачів по 120 дБ потужності, які придушують запис на мікрофонах сучасних телефонів та диктофонах ЗП. Акустичний шум потужністю 3 Вт з'являється з акустичного динаміка та глушить аналогові мікрофони телефонів, а також будь-які сучасні диктофони і мікрофони. Ультразвукові випромінювачі повинні бути спрямовані на відвідувача. Рівень шумових перешкод встановлюється контролером автоматично. Вартість – 36300 грн [5].
 Генератор WNG-023	Генератор WNG-023 розмірами 111×70×22 мм створює діапазон шумових частот 100-12000 Гц.. Захищає від витоку акустичної КІ у невеликому замкнутому просторі, створює перешкоду потужністю до 1 Вт, що знижує розбірливість записаної або переданої по радіоканалі мови. Джерело живлення - крона 9 В.

Продовження таблиці 2.3

Найменування	Опис обладнання
 Генератору шуму «PSP-2A AUTO»	Використовується для захисту приватних розмов в приміщеннях та салонах автомобілів, а також під час розмов по телефону. «PSP-2A AUTO» формує мовоподібну заваду, яка корелюється по рівню, спектру і часу з акустичним сигналом КІ за допомогою перетворення мовного сигналу методом інверсії спектру і акустичної псевдоревербації через множення і ділення його частотних складових і багатократного накладання перевідображених акустичних сигналів, що приймаються [49]. Смуга частот шумового сигналу: 200-4500 Гц. На максимальній гучності працює до 10 год. Смислова розбірливість, що забезпечується приладом при максимальній гучності перешкоди, становить менше 15%. Ціна – 25800 грн [50].
Пригнічувач мікрофонів ПМ-2А	Безпосередньо впливає на мікрофони та стетоскопи будь-якого технічного засобу за допомогою накладання в самому мікрофоні реального акустичного сигналу і перешкоди. Перешкода створюється з мовного сигналу та корелюється з формантами людської мови. Формантне перетворення здійснюються шляхом хаотичної зміни декількох параметрів вихідного сигналу - амплітудних, спектральних і фазових, по двом незалежним каналам (високо- і низькочастотні). Незначна зміна рівня спектрального складу вихідного мовного сигналу призводить до повної втрати смислової розбірливості.
AD-11, AD-31	Генератори акустичного шуму, мобільні пригнічувачі диктофонів.
USPD Circulo	Генератор шуму, ультразвуковий акустичний пригнічувач диктофонів. Ціна 56700 грн.

Продовження таблиці 2.3

Найменування	Опис обладнання
USPD-3	Ультразвуковий акустичний пригнічувач диктофонів, розроблений для забезпечення конфіденційності переговорів.
 RAPTOR NG-1	Ультразвуковий пригнічувач диктофонів. Ціна 8900 грн.
 USPD папка	Пригнічувач диктофонів, ультразвуковий акустичний генератор шуму для захисту від прослуховування. Ціна 36300 грн.

Отже, генерація шуму є ефективним засобом для блокування витоку усної КІ під час приватної розмови через прямий акустичний канал та через ЗП.

Генератори шуму можуть встановлюватися в різних частинах об'єкта інформаційної діяльності (ОІД):

1. На столі або вбудований у меблі для маскування розмови від ЗП.
2. Біля дверей та вікон – для запобігання прослуховуванню через спрямовані мікрофони.
3. Портативний генератор шуму носиться при собі або ставиться поруч із джерелом запису, щоб блокувати мікрофони смартфонів.
4. Система ультразвукового блокування запису – встановлюється у приміщенні та спрямовує перешкоди на диктофони.

Проаналізувавши технічні засоби захисту приватності розмов, можна дійти висновку, що стаціонарні генератори шуму є громіздкими, а портативні — дорогими, з обмеженим функціоналом та недостатньо зручним управлінням.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Вибір засобів для побудови системи кібербезпеки та мови програмування з метою захисту конфіденційної інформації залежить від наступних чинників:

1) норм НПА України, які встановлюють рекомендації щодо захисту акустичної інформації;

2) каналів витоку акустичної інформації;

3) кінцевих технічних вимог щодо характеристик системи.

Правова основа технічного забезпечення захисту конфіденційної інформації.

Правова основа технічного забезпечення ЗКІ базується на наступних концепціях, положеннях та законах:

1) Концепція технічного захисту інформації в Україні [26]: закладається фундамент для створення ефективної системи ТЗІ, що передбачає комплекс організаційних, технічних та програмних заходів, спрямованих на запобігання несанкціонованому доступу, витоку, модифікації чи блокуванню інформації;

2) Закон України «Про захист інформації в інформаційно-телекомунікаційних системах» [30]: регулює питання ТЗІ;

3) «Положення про технічний захист інформації в Україні» [23]: визначаються правові та організаційні засади ТЗІ під охороною держави.

Комплексна система ЗКІ з обмеженим доступом, для якої захист встановлений законом, має використовувати сертифіковані засоби захисту інформації (ЗЗІ) або такі засоби, які мають позитивний експертний висновок державної експертизи у сфері технічного (криптографічного) ЗІ [21, 23].

Окрім зазначених вище НПА, здійснення комплексної системи захисту інформації з обмеженим доступом регламентується й іншими документами, зазначеними в табл.2.4.

Таблиця 2.4 – НПА організації, регулювання та здійснення технічного захисту інформації

Назва НПА	Номер НПА	Короткий опис
1	2	3
Державні стандарти	ДСТУ 3396.2-97 [27] Захист інформації. ТЗІ. Терміни та визначення	Визначає терміни з наступних розділів: 1) види інформації (інформація з обмеженим доступом, таємна та конфіденційна), які підлягають технічному захисту; 2) загрози для інформації: витік, порушення цілісності та блокування інформації, а також такі поняття як модель загроз, закладний пристрій, технічна розвідка тощо; 3) технічні канали витоку інформації (як сукупність носія інформації, середовища його поширення та засобу технічної розвідки); 4) технічний захист інформації та його організація.
	ДСТУ 3396.0-96 [28] Захист інформації. ТЗІ. Основні положення	Визначає основні етапи побудови системи захисту інформації (СЗІ): 1) визначення та аналіз загроз: оцінка об'єктів захисту, умов функціонування, потенційних загроз; 2) розробка СЗІ: планування організаційних та технічних заходів, визначення зон безпеки та вибір методів захисту; 3) реалізація плану захисту інформації: впровадження запланованих заходів, атестація засобів ТЗІ; 4) контроль функціонування та управління СЗІ.

Продовження таблиці 2.4

1	2	3
Державні стандарти	ДСТУ 3396.1-96 [38] Захист інформації. ТЗІ. Порядок проведення робіт	Встановлює порядок проведення робіт з ТЗІ, який за наказом керівника підприємства передбачає: – організацію обстеження об'єкту ЗІ; – організацію розроблення системи ЗІ; – реалізацію організаційних заходів ЗІ; – реалізацію первинних та основних технічних заходів захисту; – прийняття, визначення повноти та якості робіт.
Державні стандарти	ДСТУ 27001:2023 [34] ІБ, кібербезпека та захист конфіденційності. Системи керування ІБ. Вимоги	Встановлює вимоги до систем управління (СУ) інформаційною безпекою (ІБ), визначає підходи до управління ризиками ІБ, включаючи політики, процедури та технічні заходи для захисту конфіденційності, цілісності та доступності інформації. Є ідентичним міжнародному ISO/IEC 27001:2022 і використовується організаціями для впровадження ефективної СУ ІБ.
Нормативними документами (НД) Державної служби спеціального зв'язку та захисту інформації України	НД ТЗІ 2.5-004-2000	Керівництво з побудови систем ТЗІ, що включає засоби акустичного захисту.
	НД ТЗІ Р-001-2000 [39]	Засоби активного захисту мовної інформації з акустичними та віброакустичними джерелами випромінювання. Класифікація та загальні технічні вимоги. Рекомендації.
	НД ТЗІ 3.7-001-06	Вимоги до методів вимірювання рівня акустичних полів у приміщеннях, що містять конфіденційну інформацію
	НД ТЗІ 2.6-001-99	Керівництво щодо створення захищених приміщень та використання шумозахисних матеріалів
	Технічний захист усної мовної інформації в симетричних абонентських аналогових телефонних лініях:	
	НД ТЗІ 2.3-002-01	- засоби пасивного приховування мовної інформації. Нелінійні атенюатори та загороджувальні фільтри

Продовження таблиці 2.4

1	2	3
	НД ТЗІ 4.7-001-01	- засоби визначення наявності та віддаленості місця контактного підключення засобів технічної розвідки
	НД ТЗІ 2.3-003-01	- засоби активного приховування мовної інформації. Генератори спеціальних сигналів
Постанови Кабінету Міністрів України (КМУ)	КМУ №373 від 29.03.2006	«Про затвердження Порядку здійснення криптографічного захисту інформації в Україні», що передбачає комплексний підхід до ТЗІ
	КМУ №939 від 18.08.2017	"Про затвердження Порядку організації та забезпечення ЗІ в інформаційно-комунікаційних системах".
Міжнародні стандарти у сфері акустичного захисту інформації	ISO/IEC 27001	Стандарт управління інформаційною безпекою
	ISO/IEC 15408	Критерії оцінки безпеки ІТ-продуктів
	ISO 3382	Вимірювання акустичних характеристик приміщень для забезпечення конфіденційності розмов
	ISO/IEC 27002	Рекомендації щодо управління інформаційною безпекою, включаючи акустичний захист.
	NIST 800-53	Рекомендації США щодо захисту інформації, включаючи захист від витоку акустичних даних.

Ці та інші документи встановлюють вимоги та рекомендації щодо здійснення технічного захисту інформації, управління ризиками та забезпечення кібербезпеки, а також визначають вимоги щодо проектування, тестування та експлуатації технічних засобів, що запобігають витоку мовної інформації.

Повний перелік документів нормативно-правової бази, що забезпечує надання послуг у галузі технічного захисту інформації можна знайти на сайті Державної служби зв’язку та захисту інформації України [33].

Обґрунтування вибору засобів для побудови системи на основі каналів витоку акустичної інформації

Під витоком інформації розуміється неконтрольоване розповсюдження КІ за допомогою різних фізичних полів (акустичних, світлових, електромагнітних та ін.) і матеріальних об'єктів, яке призводить до отримання КІ особами, без права доступу до неї [19, 23].

В залежності від того, яким фізичним полем передається КІ та яким матеріальним об’єктом вона перехоплюється, говорять про той чи інший технічний канал витоку інформації (ТКВІ), зокрема, різні види акустико-перетворювальних каналів; електромагнітний канал; візуально-оптичний та матеріально-речовий (папір, фото, магнітні носії тощо) [19, с. 23; 35, с. 21].

Отже, під ТКВІ розумітимемо фізичний шлях від джерела КІ до зловмисника, по якому можливе неправомірне отримання відомостей.

Для реалізації ТКВІ потрібні дві умови: 1) наявність інформативного сигналу, 2) засоби перехоплення і фіксації інформації та 3) фізичне середовище поширення інформативного сигналу [35].

Технічні канали витоку мовної інформації поділяться на наступні: 1) прямий акустичний, 2) віброакустичний, 3) електроакустичний, 4) акустооптоелектронний (або лазерний акустичний), 5) відеоакустичний, 6) канали ВЧ нав’язування, 7) канали витоку на основі закладних пристроїв [19].

В нашому дослідженні створена система кібербезпеки для приватності розмови розрахована на захист приватності розмови прямим акустичним каналом та каналів витоку на основі закладних пристроїв.

Носієм інформації в прямому акустичному каналі виступають звукові хвилі різних діапазонів: ультразвукового (частотою більше 20 кГц), чутного звуку (частотою 16 Гц – 20 кГц) та інфразвукового (частотою менше 16 Гц). Діапазон звукових частот, що містяться в людській мові, коливається від 80 Гц до 1400 Гц [17].

Середовищем для поширення акустичних сигналів для прямого акустичного каналу являється дифузійне середовище – повітря.

При поширені акустичної хвилі молекули повітря здійснюють відповідні коливальні рухи, передаючи їх від однієї молекули до іншої однаково в усіх напрямках. При появі на шляху звукової хвилі перешкод (стін, вікон, дверей, стелі тощо), звукові хвилі здійснюють на них відповідний тиск, приводячи їх також в коливальний вібраційний рух. Молекули перешкод, в свою чергу, передають коливання (вібрації) від молекули до молекули в середині твердого тіла і потім далі – молекулам повітря за перешкодою (наприклад, за дверима). Це призводить до того, що звук знову випромінюється, але вже з меншою інтенсивністю (частина енергії поглинається і розсіюється всередині матеріалу стіни), тобто звукові хвилі проходять крізь недостатньо захищені конструкції (рис.2.7).

Прямий спосіб витоку акустичної інформації відбувається через повітропроводи (конструкції систем опалення, вентиляції та кондиціювання повітря для транспортування повітря до приміщень), щілини в стінах, наскрізні отвори тощо (рис. 2.7).

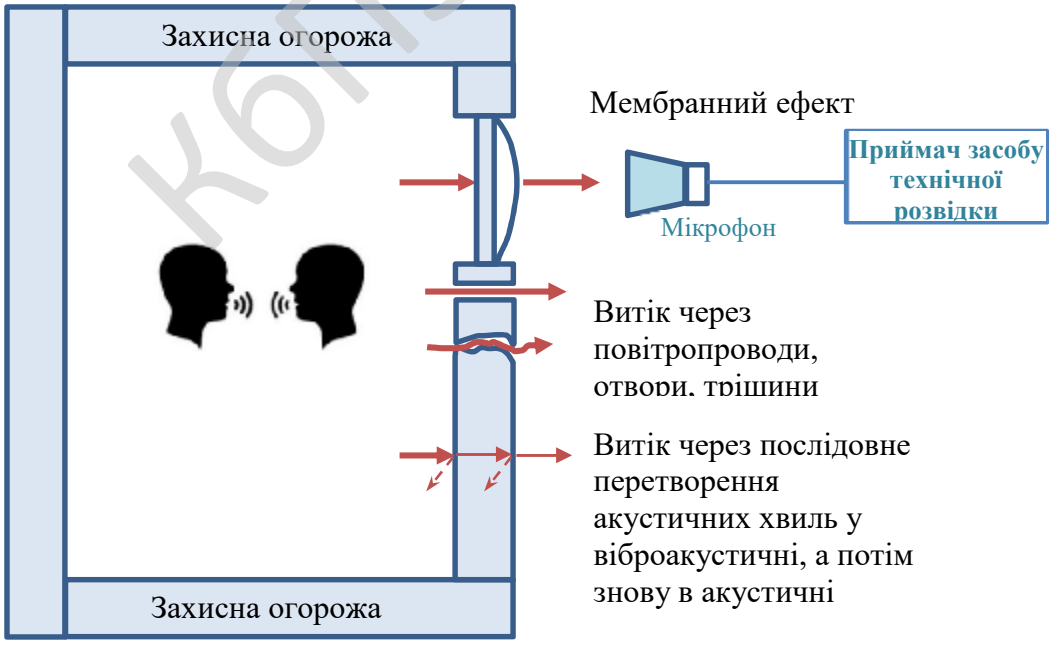

Рисунок 2.7 – Витік акустичної інформації прямим акустичним каналом

«Мембранний ефект» витоку акустичної інформації за межі приміщення обумовлений прогинанням тонких (відносно довжини) елементів огороджувальних конструкцій (віконного скла, фанерних чи пластикових перегородок тощо) під дією звукового тиску та створенням коливань з аналогічною звуковою частотою. Таким чином створюється фонограма розмови по аналогії з мембраною акустичної системи [19, 41].

Структурно-просторова модель акустичного каналу витоку інформації розроблена у дослідженні Є.В. Павленко (додаток Б), відповідно до якої модель акустичного каналу витоку мовної інформації доцільно розглядати як 4 взаємопов'язані моделі: джерела акустичного сигналу; акустичних перешкод та шумів; середовища поширення акустичного сигналу та акустичного приймача [35]. Для перехоплення акустичної КІ використовуються:

а) радіомікрофони дальністю дії 50-200 м без ретрансляторів;

б) високочутливі акустичні мікрофони спрямованої дії, а також акустичні антени засобів технічної розвідки, які встановлюються за межами ОІД в зоні, близькій до вікон, дверей та можливих щілин та отворів в стінах.

Спрямованість таких мікрофонів складає 2^0-5^0, середня дальність дії трубчастих мікрофонів становить ≈100 м, а для інших типів спрямованих мікрофонів складає в середньому до 275 м. На відкритій місцевості параболічний спрямований мікрофон може працювати на відстані до 1 км [15].

Робота каналів витоку акустичної інформації на основі закладних пристроїв (ЗП) базується на прихованому встановленні автономних розвідувальних пристроїв в зоні ОІД, які за своєю конструкцією складаються з мікрофона й передавача, та здатні розпізнавати та передавати акустичну інформацію без дозволу на це власника інформації. Такі автономні пристрої називають закладними пристроями перехоплення мовної інформації [16].

За способом передачі акустичної інформації ЗП поділяються на такі:

1) Активні радіозакладні пристрої – передають акустичну КІ, перетворюючи акустичні сигнали в електромагнітні хвилі радіоефіру.

Основні діапазони радіозакладних пристроїв складають 75-115 МГц, 115-200 МГц, 270-480 МГц, але сучасні технічні засоби розвідки мають діапазон вище 1000 МГц, що призводить до ускладнення їх виявлення. Крім діапазону радіозакладних пристроїв велику роль відіграє модуляція сигналу.

Нові класи модуляцій, такі як дельта-модуляція (кодування інформації в короткотривалі імпульси та передача їх в зазначений час), модуляція в шумоподібні сигнали чи модуляція з псевдовипадковою перестановкою несучої частоти створюють проблеми у виявленні закладних пристроїв [19].

2) Пасивні радіозакладні пристрої з перевипромінюванням ВЧ поля з модуляцією його акустичним сигналом на ОІД. Дані закладні пристрої складаються з 2-х резонаторів – ВЧ електромагнітного резонатора та акустичного. Акустичний резонатор реагує на мовний сигнал та своєю вібрацією викликає зміну добротності контура електромагнітного резонатора, що перевипромінює модульований сигнал;

3) ЗП з передачею акустичної інформації по стандартному телефонному каналу;

4) Мережеві ЗП. Передають акустичну інформацію по мережам електроживлення або ж самі живляться від мережі, але модульований акустичний сигнал передають по іншому каналу, наприклад, випромінюючи електромагнітний сигнал в радіоефір.

Також перехоплена ЗП мовна інформація може передаватися по оптичному каналу, інженерним комунікаціям в ультразвуковому діапазоні частот та іншими каналами.

Оскільки закладні пристрої знаходяться в середині ОІД, то одним з найбільш дієвих методів захищення являється метод недопущення установки ЗП в приміщенні ОІД. Також потрібний постійний контроль сигналів у лініях зв’язку, контроль радіовипромінювання та інфрачервоного випромінювання тощо. Для протидії ЗП використовують засоби електромагнітного, акустичного, вібраційного зашумлення.

Вимоги до характеристик системи кібербезпеки захисту приватності розмови шляхом генерування шуму

Блокування витоку акустичної КІ через динаміки смартфонів, які мають системи шумозаглушення, має використовувати методи, що ускладнюють фільтрацію шуму, та враховувати наступні положення:

1) Мікрофони смартфонів оснащені системами шумозаглушення, які розпізнають та відфільтровують фонові звуки, зокрема білий шум. Вони налаштовані на вловлювання мовлення навіть за наявності завад.

2) Програмований білий шум зазвичай відтворюється в діапазоні частот 200–8000 Гц. Для ефективного захисту потрібно охоплювати ширший спектр, включаючи ультразвукові частоти (понад 20 кГц). Ультразвукові хвилі перевантажують мембрану мікрофона, спотворюючи запис.

3) Для посилення захисту необхідно, щоб система ТЗІ рівномірно охоплювала всю зону переговорів, тобто мала не одне локальне джерело шуму, а кілька таких джерел, розташованих навколо зони переговорів.

4) Захист ефективніший, коли шум генерується в динамічному режимі у синхронізації з мовленням – змінюється інтенсивність і частота залежно від гучності голосу. Статичний шум легше відфільтрувати під час подальшої обробки запису, тому використання динамічних змін – різночастотного шуму, який змінюється у реальному часі, створюючи «рваний» або імпульсний шум та ускладнюють фільтрацію шуму для смартфонів.

5) Маскування мовлення акустичними сигналами, які схожі за структурою на мовлення, з метою ускладнення розпізнавання. Такий шум накладається на голос і унеможливлює подальшу обробку запису..

6) Щоб уникнути фільтрації шуму мікрофонами смартфонів, важливо, щоб джерело шуму було близько до мовця.

7) Використання віброакустичних систем, які створюють коливання та заважають мікрофонам знімати вібрації від звуків через різні поверхні.

Отже, для максимального захисту акустичної інформації від витоку акустичними каналами та запису на ЗП необхідно:

1) Комбінувати методи: шум різних частот та ультразвук.
2) Розміщувати засоби ТЗІ правильно: біля джерела мовлення.
3) Використовувати активне маскування: динамічні сигнали ефективніші за статичний шум.

Основний тон (фундаментальна частота) природної людської мови лежить в діапазоні від 60-70 Гц (низькі звуки баса) до 1200-1300 Гц (високі ноти сопрано), в середньому для середньостатистичної людини основний тон змінюється в межах 85-255 Гц. Крім основного тону, голосові зв'язки людини при вібрації створюють гармоніки – додаткові частоти, кратні основному тону. У природному мовленні гармоніки можуть досягати до 10-12 кГц. Крім гармонік також виникають резонансні частоти – форманти, які критично важливі для розбірливості мови. У приголосних і шумових звуках форманти також можуть містити частоти до 10-12 кГц.

Пристрої для активного захисту акустичної інформації можуть мати більш вузький діапазон частот – 300-3400 Гц, оскільки вони обмежені схемою та мають обмежений час роботи від батареї.

Однак, для забезпечення більш надійного захисту, рекомендується використовувати генератори шуму з ширшим діапазоном частот, що охоплює низькі частоти (100-300 Гц), які можуть бути важливими для приглушення низькочастотних звуків, таких як гул або вібрації, а також високі частоти (3,4-10 кГц), які можуть бути важливими для приглушення високочастотних звуків, таких як шипіння або свист.

Таким чином, оптимальний діапазон частот для генератора шуму для активного захисту акустичної інформації становить від 100 Гц до 10 кГц.

Згідно нормативного документу Р-001-2000 [39], діапазон робочих частот вихідного сигналу повинен бути не менше (180…5600) Гц.

Рівень шуму повинен бути достатнім для приглушення мовного сигналу, але не перевищувати допустимі санітарні норми. Згідно наказу МОЗ № 463 «Про затвердження Державних санітарних норм допустимих рівнів шуму в приміщеннях житлових та громадських будинків і на території житлової забудови», медико-санітарні нормативи допустимих рівнів звуку в приміщеннях житлових будинків становлять 35-55 дБА вдень та 25-35 дБА вночі, а на території житлової забудови – 50-60 дБА вдень та 40-50 дБА вночі.

Додатково, варто знати, що низькочастотний шум інтенсивністю до 100 дБ не викликає відчутної несприятливої дії на орган слуху. При постійному шумовому фоні до 70 дБ виникає порушення ендокринної та нервової систем, а при 90 дБ відбувається порушення слуху. Тому використання генераторів шуму з високим рівнем звукового тиску для захисту мовної інформації має бути строго регламентованим за часом.

Крім забезпечення стабільного рівня шуму впродовж певного часу, генератор шуму повинен мати можливість регулювання рівня звукового тиску та спектральних характеристик.

Нормативний документ Р-001-2000 [39] визначає основні вимоги до активних засобів захисту мовної інформації (генераторів шуму) та їх характеристики, до яких відносяться: 1) робоча площина акустичного випромінювання – площина випромінюючих отворів випромінювача; 2) коефіцієнт напрямленості – відношення рівнів звукового тиску, що розвиває випромінювач в напрямку робочої осі до рівня тиску, що розвивається під кутом до робочої осі в заданій площині та на заданій відстані від робочого центру випромінювача; 3) вихідний сигнал – звуковий сигнал у вигляді білого або рожевого шуму, який створюється випромінювачем.

Розподілення рівнів вихідного акустичного сигналу в третьоктавних смугах частот у діапазоні робочих частот повинно відповідати білому або рожевому шуму. Розподіл на третьоктавні смуги повинен відповідати ГОСТ 12090-80 в межах діапазону робочих частот.

В засобах з цифровим методом формування шумового сигналу період повторення псевдовипадкової (псевдошумової) послідовності повинен вибиратися з ряду: 8, 16, 24 години, … (далі з періодом 8 годин) [39].

Отже, враховуючи проведений аналіз в п.2.1 та 2.2 можна визначити основні вимоги до портативного генератора шуму зазначені в табл.2.5.

Таблиця 2.5 – Основні вимоги щодо технічного засобу захисту акустичної інформації методом генерації шуму.

Характеристика	Рівень за нормативними документами	Оптимальний рівень
Діапазон робочих частот	180…5600 Гц [39]	100 – 10000 Гц Ультразвукові частоти 20 кГц
Рівень вихідного звукового тиску, що розвивається акустичними колонками в заданому діапазоні частот у вільному просторі	Максимальний рівень вибирається з похибкою не > 6 дБ з ряду: 60; 70; 80; 90; 100 дБ – на відстані 1 м від джерела шуму [39].	Плавне регулювання рівня звукового тиску: 60-85 дБ
Діапазон ослаблення рівня звукового тиску вихідного акустичного сигналу по відношенню до усередненого максимального рівня	Вибирається з ряду: 0; 10; 20; 30; 40; 50 дБ	Плавне регулювання рівня звукового тиску:
Період повторення псевдошумової послідовності (для засобів з цифровим методом формування шумового сигналу).	Повинен вибиратися з ряду: 8, 16, 24, 32 год, … [39]	використання різночастотного шуму, який постійно змінюється
Розміри	Не нормовані	Орієнтовно 10x10x3 см
Вид шуму	Білий, рожевий	Мовленевоподібні перешкоди

Опис засобів для побудови системи кібербезпеки захисту приватності розмови шляхом генерування шуму та мови програмування

Для розробки системи кібербезпеки, яка забезпечує захист приватності розмови шляхом генерування шуму, було обрано низку апаратних і програмних засобів, а також мову програмування, які найкраще відповідають вимогам проекту. Обґрунтування вибору цих засобів базується на їхніх технічних характеристиках, функціональних можливостях, сумісності між собою та зручності використання в умовах розробки вбудованих систем.

Основою апаратної частини системи є мікроконтролер ESP32. Цей вибір обумовлений його високою продуктивністю, багатофункціональністю та доступністю. ESP32 має двоядерний процесор, що дозволяє виконувати багатозадачні операції, такі як обробка аудіосигналів, керування периферійними пристроями та обслуговування мережевих з'єднань. Крім того, ESP32 підтримує бездротові протоколи Wi-Fi і Bluetooth, що відкриває можливості для розширення функціоналу системи, наприклад, для віддаленого керування або передачі даних [6, 7]. Важливою перевагою ESP32 є його підтримка великої кількості бібліотек і фреймворків, що значно спрощує розробку програмного забезпечення [4].

Для запису голосу людини використовується аналоговий мікрофон. Цей вибір обумовлений його простотою інтеграції з ESP32 через вбудований АЦП (аналогово-цифровий перетворювач). Аналоговий мікрофон дозволяє отримувати аудіосигнал, який потім обробляється мікроконтролером для подальшого використання в генерації шуму. Висока чутливість мікрофона забезпечує якісний запис навіть у складних акустичних умовах.

Для відтворення звуку використовуються дві пари динаміків, підключених через підсилювач звуку [9]. Одна пара динаміків використовується для відтворення шуму, який генерується на основі записаного голосу, а друга пара — для генерації ультразвуку.

Використання підсилювача звуку забезпечує достатню гучність і якість відтворення, що є критично важливим для ефективного захисту приватності

розмови. Ультразвук, який генерується другою парою динаміків, створює додатковий рівень захисту, оскільки він не сприймається людським вухом, але може впливати на роботу пристроїв для запису звуку.

Для керування системою використовуються чотири кнопки, кожна з яких відповідає за певну функцію, наприклад, початок запису, відтворення шуму, перемикання між файлами або активацію ультразвукового режиму. Індикація натискання кнопок реалізована за допомогою чотирьох світлодіодів, які сигналізують про успішне виконання відповідної команди. Це забезпечує зручність використання системи та дозволяє користувачеві швидко орієнтуватися в її стані.

Для виведення інформації використовується невеликий екран, на якому відображаються повідомлення про стан системи, наприклад, назва поточного файлу, статус запису або відтворення, а також інші важливі дані. Екран забезпечує зручний інтерфейс для взаємодії з користувачем і дозволяє оперативно отримувати інформацію про роботу системи.

Усі компоненти системи зібрані на друкованій платі, яка була спеціально розроблена для цього проекту. Використання друкованої плати забезпечує компактність і надійність конструкції, а також спрощує монтаж і підключення компонентів. Для захисту електронних компонентів і забезпечення естетичного вигляду системи був надрукований корпус, який також виконує функцію ергономічного утримувача для користувача.

Щодо програмного забезпечення, для розробки було обрано мову програмування C++ у середовищі Arduino. Цей вибір обумовлений кількома факторами. По-перше, C++ є однією з найпоширеніших мов для розробки вбудованих систем завдяки своїй ефективності, низькорівневому доступу до апаратних ресурсів і широкій підтримці бібліотек.

По-друге, середовище Arduino надає зручний інструментарій для роботи з ESP32, включаючи готові бібліотеки для роботи з периферійними пристроями, такими як SD-карта, дисплей, мікрофон і динаміки. Це значно скорочує час

розробки та дозволяє зосередитися на реалізації основної функціональності системи.

Використання бібліотек, таких як ESPAsyncWebServer для обробки HTTP-запитів, LittleFS для роботи з файловою системою та Inter-IC Sound (I^2S) для обробки аудіосигналів, забезпечує високу продуктивність і стабільність роботи системи. Крім того, підтримка FreeRTOS у середовищі Arduino дозволяє реалізувати багатозадачність, що є важливим для одночасного виконання кількох процесів, таких як запис, відтворення та обробка команд користувача.

Таким чином, вибір апаратних і програмних засобів для побудови системи кібербезпеки був обґрунтований їхньою відповідністю технічним вимогам проекту, зручністю інтеграції та ефективністю. Використання ESP32, аналогового мікрофона, динаміків, підсилювача звуку, кнопок, світлодіодів і дисплея в поєднанні з мовою програмування C++ і середовищем Arduino дозволило створити компактну, функціональну та зручну у використанні систему для захисту приватності розмови.

2.3 Розгорнута постановка завдання

Отримавши всі деталі реалізації цієї системи, потрібно виконати розгорнуту постановку завдання. Розробка системи передбачає створення фізично вбудованого програмно-апаратного комплексу на базі мікроконтролера ESP32, здатного в режимі реального часу ініціювати генерування маскуючого сигналу (у тому числі ультразвукового діапазону), а також забезпечувати візуальний і фізичний інтерфейс для зручного керування пристроєм. На відміну від традиційних підходів до захисту розмов – шифрування передачі або створення захищених приміщень – у цьому випадку акцент робиться саме на активне перешкоджання запису мови за рахунок акустичного шуму, що ускладнює або унеможливлює подальше її розпізнавання.

В апаратній реалізації передбачено використання двох пар динаміків: одна пара — для відтворення шуму у чутному діапазоні, інша — для випромінювання ультразвуку. Це дозволяє розширити спектр захисного впливу на потенційні пристрої стеження. Для забезпечення адекватної потужності звучання використовується підсилювач звуку, який виводить оброблений сигнал на обидві пари динаміків. Шум формується на основі вхідного аудіосигналу, що подається з аналогового мікрофона. Цей мікрофон фіксує мовну активність користувача, і на його основі запускається або коригується алгоритм генерування шуму. Таким чином, система може функціонувати як у постійному режимі, так і в режимі, активованому голосом.

Програмна логіка пристрою реалізується на мові C++ у середовищі Arduino, що забезпечує повну інтеграцію з апаратною частиною й ефективне використання обчислювальних можливостей ESP32. Завданням мікроконтролера є керування вхідними/вихідними потоками даних, обробка мікрофонного сигналу, запуск шумогенератора, керування відображенням на екрані та індикацією, а також обробка натискань кнопок. Розробка включає реалізацію паралельних задач, які працюють у реальному часі, зокрема: генерація шуму, виведення інформації, відстеження станів кнопок та світлодіодів.

Також потрібно забезпечити інтеграцію з вебінтерфейсом, щоб надати додатковий рівень гнучкості та зручності. Користувач має можливість керувати пристроєм дистанційно через локальну мережу за допомогою браузера. Через вебінтерфейс можна запускати режими генерації шуму або ультразвуку, переглядати лог подій, змінювати налаштування, а також отримувати статусні повідомлення. Для реалізації цієї функціональності вбудовано вебсервер на основі бібліотеки ESPAsyncWebServer, який обробляє HTTP-запити та забезпечує швидкий обмін даними між користувачем і пристроєм.

Також, щоб забезпечити взаємодію користувача з пристроєм без використання вебінтерфейсу, важливо забезпечити інтуїтивно зрозумілий інтерфейс. Для цього передбачено використання чотирьох кнопок, кожна з яких

виконує конкретну функцію, наприклад: запуск генерації шуму, увімкнення/вимкнення ультразвуку, перемикання режимів роботи або перезапуск пристрою. Для підтвердження дій та контролю поточного стану системи реалізована світлодіодна індикація — чотири світлодіоди сигналізують про успішне натискання та виконання відповідних команд. Додаткову візуальну інформацію (назви режимів, помилки, статуси) користувач отримує з OLED-дисплея, що вбудований у корпус пристрою. Цей екран дозволяє оперативно стежити за функціонуванням системи без потреби у додаткових пристроях.

Завдання проєкту також включає відлагодження та тестування функціональності системи в умовах, максимально наближених до реальних. Це передбачає перевірку спрацювання пристрою в умовах фонового шуму, тестування ефективності маскування мовного сигналу в різних режимах генерації, а також перевірку дії ультразвукових хвиль на записуючі пристрої. Особлива увага приділяється стабільності роботи пристрою протягом тривалого часу, здатності швидко реагувати на команди користувача та відсутності збоїв у логіці перемикання режимів.

У результаті очікується отримання повноцінного, самостійного пристрою, який відповідає сучасним вимогам у галузі захисту мовної інформації, є компактним, мобільним, простим у використанні та ефективним у своїй основній функції — забезпеченні приватності розмови шляхом акустичного маскування. Реалізація цього завдання відкриває перспективи подальшого вдосконалення розробки, зокрема — впровадження адаптивної генерації шуму, віддаленого керування або інтеграції з іншими кіберзахисними системами.

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

Системи кібербезпеки захисту приватності розмови шляхом генерування шуму повинна забезпечувати конфіденційність голосових розмов шляхом створення маскувального шуму, який перешкоджає несанкціонованому прослуховуванню. Основою рішення є плата ESP32, два комплекти динаміків, звуковий підсилювач, аналоговий мікрофон, чотири кнопки управління, чотири світлодіоди для індикації, невеликий дисплей і виготовлений 3D-друкований корпус. Взаємодія цих компонентів і програмна логіка реалізують звуковий генератор маскувального шуму, здатний працювати в ультразвуковому діапазоні.

3.1 Опис функціонування системи

При ввімкненні живлення ESP32 проходить ініціалізацію периферійних модулів: налаштовуються цифрові виводи для кнопок, світлодіодів та динаміків, активується інтерфейс ADC для роботи з мікрофоном, ініціалізується дисплей для відображення параметрів роботи, а також піднімається вбудований файл-сервер для надання веб-інтерфейсу. Одночасно запускається веб-сервер, розміщений у папці data плати, де зберігається зібраний білд UI. Це дозволяє підключитися до ESP32 з будь-якого пристрою в локальній мережі та керувати всіма параметрами дистанційно.

Програмне забезпечення складається з головного циклу обробки подій і декількох допоміжних модулів: обробник вводу з кнопок, модуль індикації світлодіодів, підсистема керування генератором шуму, веб-інтерфейс та дисплейний інтерфейс. Обробник натискань кнопок відслідковує логіку перемикання режимів, регулювання інтенсивності шуму та встановлення додаткових параметрів. При натисканні кожної з чотирьох кнопок на відповідний

цифровий вивід подається сигнал і запускається обробка, після чого загоряється відповідний світлодіод, що підтверджує спрацювання.

Для реєстрації голосу співрозмовника використовується аналоговий мікрофон, підключений через вхід звукового підсилювача до ADC ESP32. Сигнал від мікрофона перетворюється на цифровий і обробляється з метою визначення рівня гучності та спектральних характеристик голосу. На підставі отриманих даних формується параметричний опис мовного сигналу, необхідний для синтезу відповідного маскувального шуму.

Модуль генерації шуму використовує два динаміки: перший відтворює звуковий шум у чутному діапазоні, другий призначений для ультразвукового випромінювання. Генерація шуму базується на програванні файлу, який генерується на основі голосу людини. Ультразвуковий шум створює додатковий бар'єр для запису за допомогою відомих методів перехоплення, оскільки такі частоти не відтворюються звичайними пристроями, але за певних умов впливають на мікрофон шкідливим чином.

Управління підсилювачем звуку та динаміками здійснюється через ШІМ-канали ESP32. В залежності від обраного алгоритму генеровані звукові хвилі формуються програмно, а потім перетворюються ШІМ у аналоговий сигнал для підсилення. Це дозволяє динамічно змінювати амплітуду та форму шуму без додаткового апаратного перемикання. Завдяки гнучкому програмному підходу можна оперативно адаптувати систему до умов довкілля та рівня голосу співрозмовника.

Дисплей виконує роль інформаційного вікна для користувача: на екрані відображається поточний режим робот. Інтерфейс реалізований за допомогою простих графічних елементів, що оновлюються в реальному часі. Взаємодія з дисплеєм дозволяє вибрати один із декількох алгоритмів генерації шуму: рівномірний білий шум, адаптивний шум на основі спектральних характеристик мови, ультразвуковий імпульсний шум.

Веб-інтерфейс надає альтернативний спосіб керування системою через браузер. Використовуючи набір HTML, CSS та JavaScript, зібраний у папці data на ESP32, користувач може віддалено моніторити стан системи. Це робить пристрій зручним для інтеграції в корпоративні мережі або IoT-екосистеми.

Програмна архітектура побудована за принципом поділу на задачі: в окремому потоці обробляється вхідний сигнал мікрофона, в іншому — здійснюється синтез шуму та керування ШІМ, у третьому — відбувається обробка вводу користувача, оновлення індикації та обслуговування веб-запитів. Завдяки цьому система зберігає стабільність та швидкодію при одночасному виконанні всіх функцій.

Матеріальна частина зібрана на одній друкованій платі з урахуванням мінімізації розмірів та зручності монтажу. Корпус, надрукований на 3D-принтері, надійно фіксує всі модулі та забезпечує доступ до кнопок, дисплея й динаміків. Висока мобільність системи дозволяє використовувати її в різних умовах: від кабінетів переговорів до відкритих просторів.

Після запуску система проходить коротку самодіагностику: тестуються всі датчики, діоди індикації, перевіряється коректність роботи дисплея, стабільність веб-інтерфейсу та здоровий стан ШІМ. У разі виявлення критичних несправностей відображається повідомлення на екрані або у браузері, після чого генерація шуму призупиняється для запобігання некоректної роботи.

3.2 Розробка структурної схеми

У процесі створення системи кібербезпеки, що забезпечує захист приватності розмови шляхом генерування шуму, важливим етапом є розробка структурної схеми пристрою, вона зображена на рисунку 3.1. Структурна схема слугує візуалізацією логічних взаємозв’язків між апаратними компонентами системи та дозволяє чітко окреслити функціональні ролі кожного з них. У даному розділі описується побудова системи, її складові та логіка взаємодії елементів.

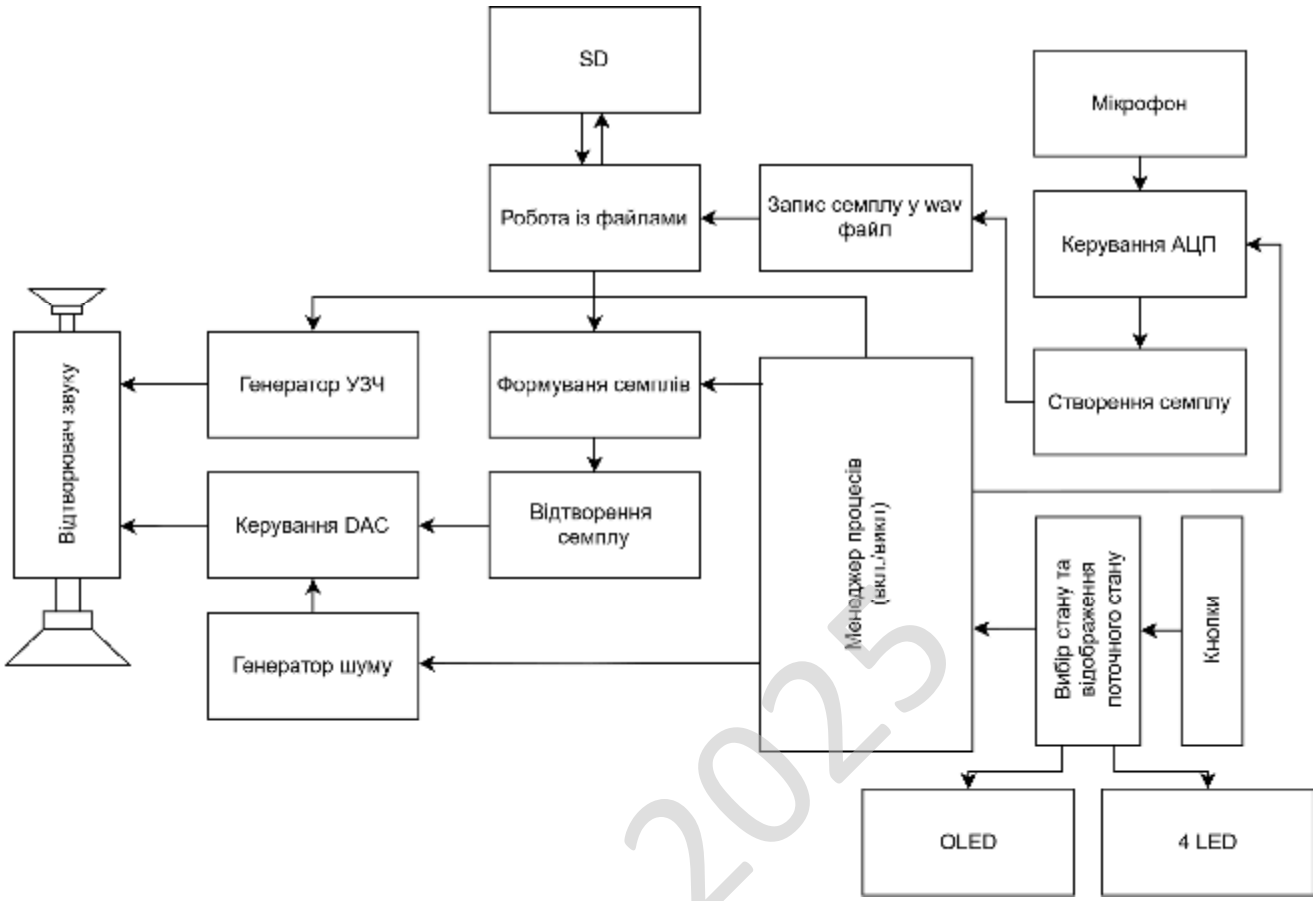

Рисунок 3.1 - Структурна схема системи кібербезпеки захисту приватності розмови шляхом генерування шуму

Центральним елементом всієї системи є мікроконтролер ESP32, який виступає як основний обчислювальний модуль, на якому реалізовано керування всією периферією та логіку роботи системи.

Для захисту розмови передбачено використання двох пар динаміків. Одна пара — це основні динаміки, які безпосередньо генерують звукову маскувальну хвилю на основі записаного мікрофоном сигналу. Інша пара використовується для відтворення ультразвукових коливань, які не чутні для людського вуха, однак здатні впливати на роботу мікрофонів сторонніх пристроїв (наприклад, мобільних телефонів або диктофонів), ускладнюючи запис приватних розмов. У структурній схемі це рішення представлено окремими каналами звукового виходу, що підключені до підсилювачів.

Підсилювачі звуку в системі виконують роль посилення вихідного сигналу, отриманого від ESP32, до рівня, достатнього для подачі на динаміки. Підсилення є необхідним, оскільки вихідна потужність мікроконтролера є недостатньою для ефективної генерації як звукового, так і ультразвукового сигналу. У конструкції передбачено окреме живлення для підсилювачів, що дозволяє зменшити вплив шумів та збурень на основну логіку ESP32.

Ще одним ключовим елементом є аналоговий мікрофон, який зчитує голос людини у режимі реального часу. Цей сигнал надходить на ESP32, де проходить первинну цифрову обробку та аналіз. На основі частотного та амплітудного аналізу мікроконтролер формує шум, який модулюється з урахуванням параметрів вхідного голосу. Такий підхід дозволяє досягти адаптивного маскування, коли шум є достатньо схожим за структурою з голосом, що ускладнює його відокремлення при спробі запису або подальшої обробки.

Для реалізації інтерфейсу керування було використано чотири кнопки. Кожна з них відповідає за окрему функцію: запуск/зупинка шуму, запис голосу, перемикання файлів для програвання. Кожна кнопка має відповідну світлодіодну індикацію — при натисканні певної кнопки загоряється світлодіод, що сигналізує про активацію функції. Це забезпечує зворотний зв’язок з користувачем та дає розуміння поточного стану пристрою.

Крім кнопок та світлодіодів, у пристрої також присутній OLED-дисплей, який використовується для виведення інформації про режим роботи, стан пристрою, обраний файл та інші параметри. Дисплей дозволяє користувачеві орієнтуватися у функціональності пристрою без використання зовнішніх додатків, що значно покращує зручність користування.

3.3 Розробка функціональної схеми

На цьому етапі було спроєктовано функціональну схему пристрою, яка зображена на рисунку 3.2, призначеного для забезпечення захисту приватності розмови шляхом генерації акустичного шуму. Схема ілюструє принцип роботи системи та взаємодію її апаратних компонентів із програмними модулями, що забезпечують повний цикл обробки голосових сигналів — від їх виявлення до формування перешкод. Завдяки використанню операційної системи реального часу (RTOS), система здатна оперативно реагувати на події, що надходять від користувача або через інтерфейс API.

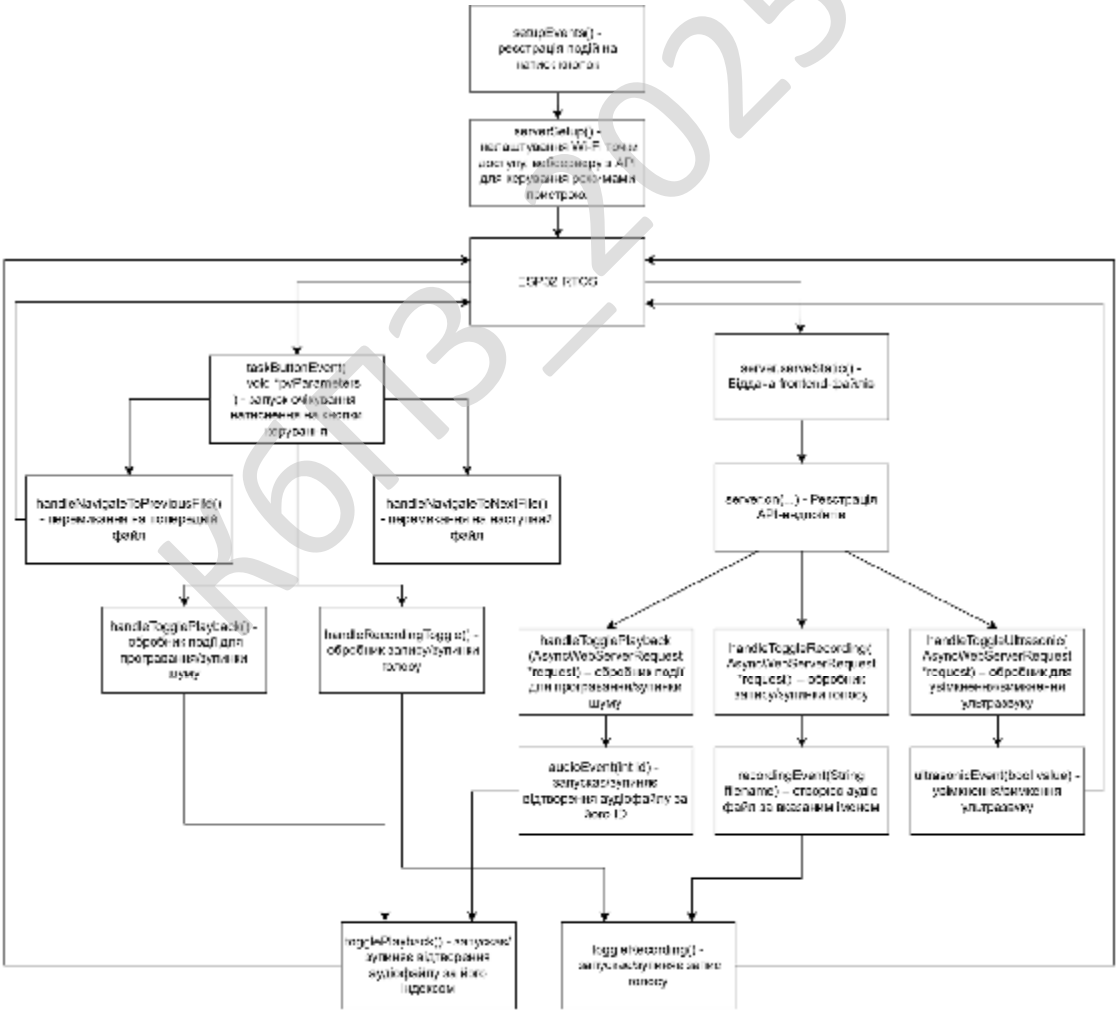

Рисунок 3.2 - Функціональна схема системи кібербезпеки захисту приватності розмови шляхом генерування шуму

Основою функціональної логіки є механізм подій, зокрема функція taskButtonEvent(void *pvParameters), яка ініціює відповідну задачу та обробляє взаємодію з користувачем. Після ініціалізації пристрій переходить у режим очікування команд. Користувач має змогу переглядати наявні аудіофайли за допомогою функцій handleNavigateToNextFile() та handleNavigateToPreviousFile(), які дозволяють перемикатися між файлами. Для запуску або зупинки відтворення вибраного файлу використовується функція handleTogglePlayback(), а у разі необхідності створення нового аудіофайлу викликається handleRecordingToggle().

Крім взаємодії з фізичними кнопками, система підтримує керування через вебдодаток завдяки реалізованому API, який ініціалізується функцією serverSetup(). API надає три основні ендпоінти: /toggle-playback, /toggle-recording та /toggle-ultrasonic. Взаємодія з цими маршрутами дозволяє керувати відповідними режимами пристрою віддалено.

Ендпоінт /toggle-playback викликає функцію handleTogglePlayback(AsyncWebServerRequest *request), яка перевіряє параметр id у запиті, що відповідає індексу аудіофайлу для відтворення. У разі успішної перевірки система викликає подію audioEvent(id) і здійснює запуск файлу. Якщо параметр некоректний або відсутній, повертається відповідний код помилки.

Ендпоінт /toggle-recording використовує функцію handleToggleRecording(AsyncWebServerRequest *request), яка очікує параметр filename — ім’я файлу для збереження запису. За його наявності запускається або зупиняється процес запису через подію recordingEvent(filename). У разі відсутності параметра сервер повертає помилку.

Ендпоінт /toggle-ultrasonic викликає функцію handleToggleUltrasonic(AsyncWebServerRequest *request), що керує параметром isActive, який визначає, чи потрібно активувати ультразвуковий режим. Успішне виконання змінює стан змінної ultrasonicMode та ініціює подію ultrasonicEvent(toggle).

Ключовими внутрішніми функціями є togglePlayback() та toggleRecording(), які виконуються як під час виклику API, так і при обробці внутрішніх подій. togglePlayback() перемикає стан аудіовідтворення, зупиняючи або відтворюючи вибраний файл, оновлює індикатори та інформацію про поточний файл. У свою чергу, toggleRecording() відповідає за зміну стану запису: при активному записі зупиняє процес і зберігає файл, при неактивному — створює новий файл та розпочинає запис.

3.4 Розробка діаграми процесів

Діаграма процесів слугує ефективним засобом для візуального представлення, аналізу та фіксації всіх процедур, що відбуваються в межах певної системи чи організації. Вона дозволяє чітко відобразити взаємозв'язки між окремими етапами виконання робіт і зрозуміти вплив кожного процесу на кінцевий результат. Графічне зображення процесів програмного забезпечення, яке реалізує функції кібербезпеки шляхом забезпечення приватності розмов за допомогою генерації шуму, наведене на рисунку 3.3.

Після початку діаграми відбувається ініціалізація компонентів системи: карти пам'яті, аудіо системи, мікрофону, дисплею, реєстрація подій, роздачі точки доступу та запуст веб сервера.

Після цього програма переходить в режим очікування команд. Команди можуть надходити або від веб-додатку або за допомогою ручного керування. Основні процеси, які реалізовані в системі, є запис звуку, відтворення звукових файлів, генерація ультразвукових сигналів. Управління режимами роботи відбувається за допомогою чотирьох кнопок. Кожна кнопка відповідає за конкретний режим: вмикання / вимикання генерації шуму, перемикання типів шуму та вмикання / вимикання запису голосу. Натискання кнопки супроводжується індикацією відповідного світлодіода та виводу повідомлення на екран, що підтверджує виконання дії. Така зворотна візуальна реакція підвищує

зручність використання пристрою, дозволяючи користувачу оперативно реагувати на зміну параметрів.

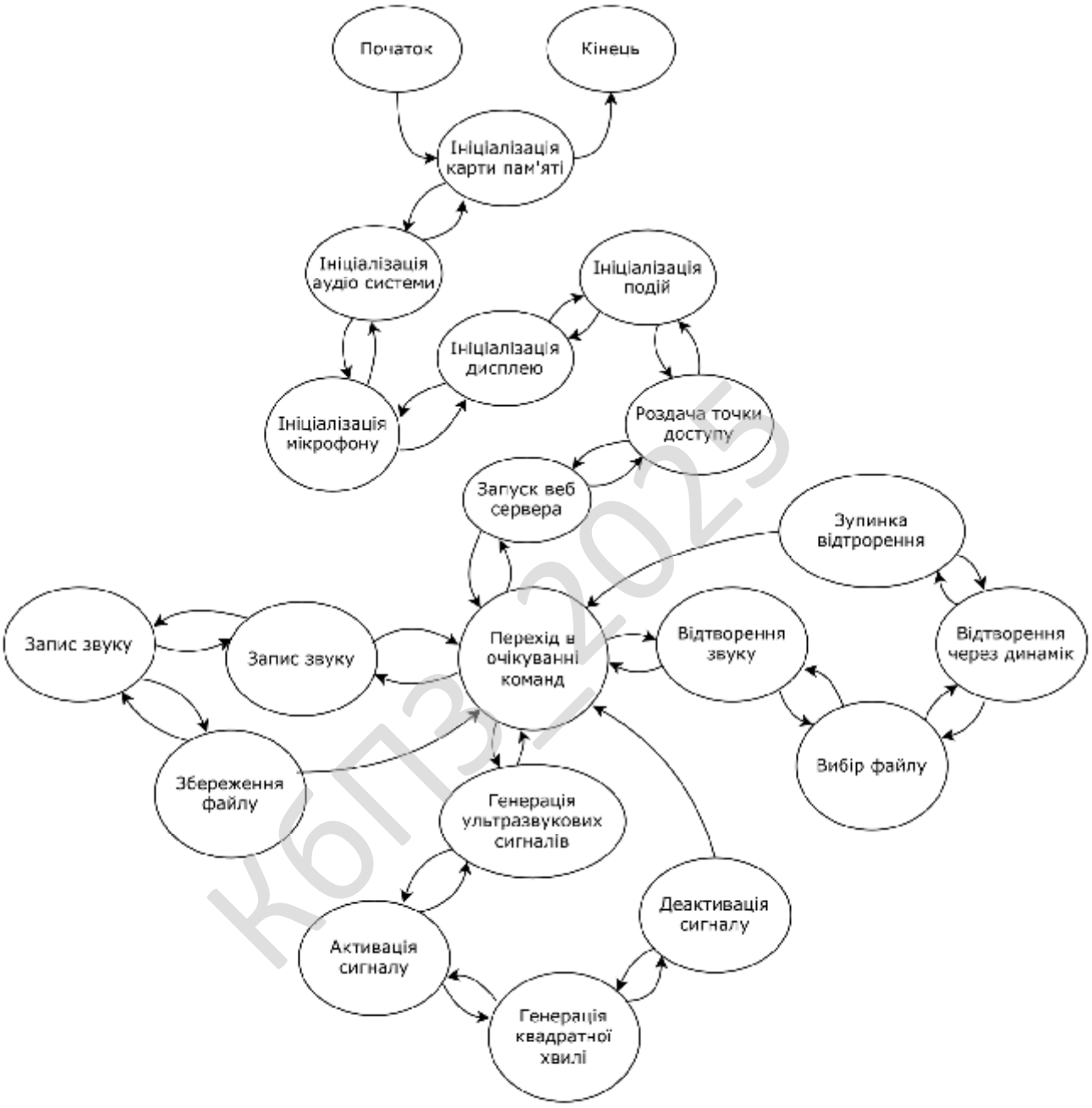

Рисунок 3.3 - Діаграма процесів забезпечення системи кібербезпеки захисту приватності розмови шляхом генерування шуму

Процес запису звуку починається з натисканням кнопки. Система перевіряє, чи запис уже не виконується, і якщо ні, створює новий WAV-файл на SD-карті.

Дані з мікрофона зчитуються блоками, обробляються та записуються у файл. Запис триває до 15 секунд або до відпускання кнопки, після чого файл закривається, а процес завершується. Цей процес забезпечує можливість створення нових звукових файлів для подальшого використання.

Процес відтворення звуку починається з вибору WAV-файлу, який зберігається на SD-карті. Користувач може вибрати файл через інтерфейс або API-передаючи ID файлу. Система перевіряє, чи відтворення вже не виконується, і якщо ні, відкриває вибраний файл і починає його відтворення через динамік, підключений до ESP32 через I2S-інтерфейс. Дані зчитуються блоками, перевіряються на відповідність формату WAV і передаються на динамік. Відтворення можна зупинити в будь-який момент через інтерфейс системи. Цей процес дозволяє використовувати попередньо записані файли для створення звукових перешкод.

Окремо варто відзначити відображення статусної інформації на OLED-екрані. Під час зміни режимів або подій (наприклад, включення генерації шуму) екран динамічно оновлює дані. Це забезпечує прозорість процесів і спрощує контроль над пристроєм.

Процес генерації ультразвукових сигналів активується через API-запит /toggle-ultrasonic, де передається параметр isActive=true. Система генерує квадратну хвилю на частоті 20–22 кГц , яка передається через динамік. Ультразвуковий сигнал створює додаткові перешкоди для аудіозаписуючих пристроїв, які працюють у чутному діапазоні. Ультразвуковий режим можна вимкнути через API-запит із параметром isActive=false. Цей процес забезпечує додатковий рівень захисту приватності розмов.

3.5 Розробка принципової схеми пристрою

Принципова схема пристрою, яка зображена на рисунку 3.4, ґрунтується на модульному поєднанні апаратних блоків із центральним мікроконтролером ESP32, який забезпечує керування всіма функціональними елементами системи.

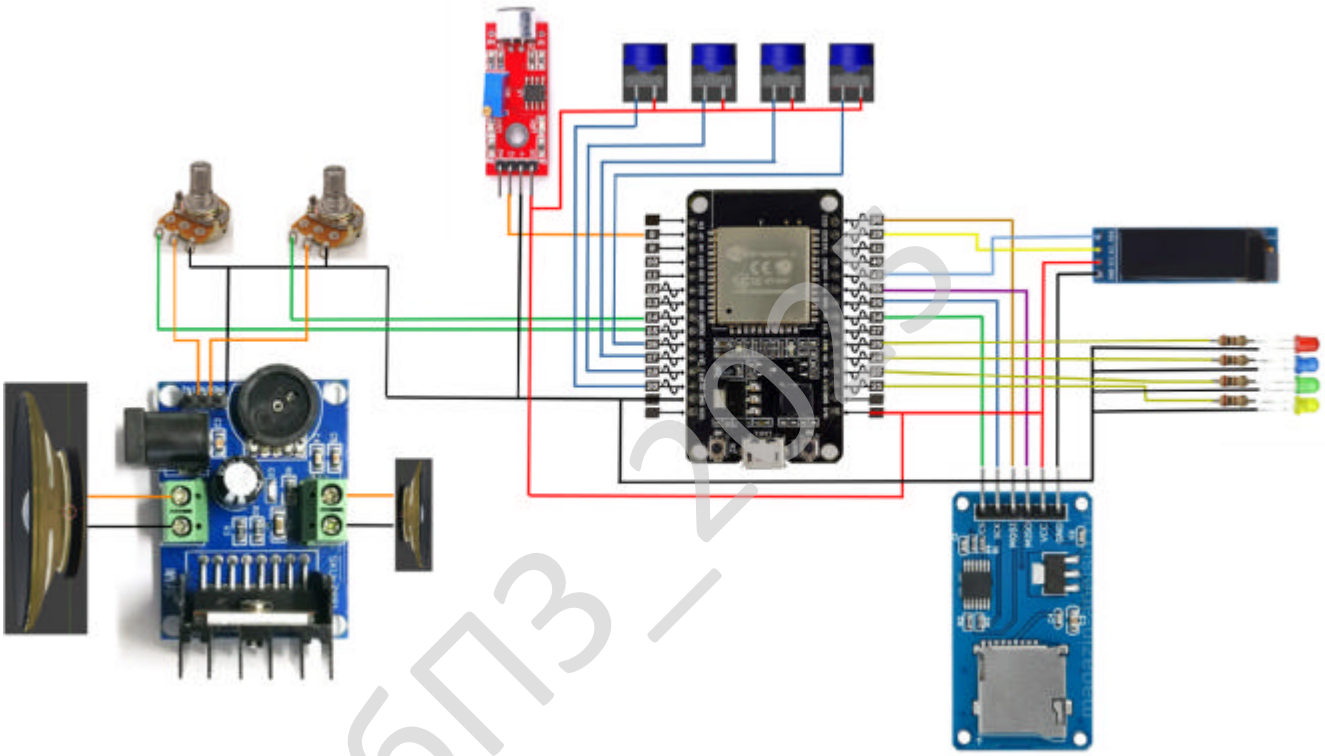

Рисунок 3.4 – Принципова схема системи кібербезпеки захисту приватності розмови шляхом генерування шуму

На вході пристрою розташовано аналоговий мікрофон, підключений до одного з каналів АЦП ESP32, що дозволяє реєструвати рівень мовного сигналу та за потреби ініціювати процес генерації шуму. Для формування маскуючих сигналів використовуються два вихідні аудіоканали: перший — для відтворення шуму у звуковому діапазоні через динаміки, другий — для ультразвукового сигналу на частоті 20–22 кГц. Ці канали реалізовані через інтерфейс I²S та підсилювач звуку,

що гарантує необхідний рівень звукового тиску для ефективного екранування розмови.

У схемі передбачено чотири тактові кнопки для управління режимами роботи: увімкнення/вимкнення генерації шуму, переключення між типами шуму, початок/зупинка запису тощо. Кожна кнопка оснащена світлодіодною індикацією, підключеною до цифрових виходів ESP32, що надає зворотний зв’язок про стан пристрою. Відображення детальної статусної інформації здійснюється на OLED-дисплеї, підключеному через інтерфейс I²C, що дозволяє в реальному часі інформувати користувача про активний режим, обраний аудіофайл або стан ультразвукового генератора.

Живлення пристрою організовано від стабілізованого джерела 5 В, далі розгалуженого на лінії 3,3 В для ESP32 та інтерфейсних модулів, а також 12 В для підсилювача звуку. Принципова схема містить захисні елементи — запобіжники та LC-фільтри — для зменшення перешкод у живленні та забезпечення стабільної роботи. Всі компоненти змонтовані на друкованій платі з урахуванням грамотного розташування ліній живлення, високочастотних інтерфейсів та роз’ємів, що дозволяє мінімізувати перехресні завади та забезпечити належну електромагнітну сумісність. Таке рішення забезпечує компактність, надійність та зручність сервісного обслуговування пристрою.

4 РЕАЛІЗАЦІЯ РОБОТИ

Для наочності реалізацію використаних функцій буде подано у вигляді блок-схем із детальним відображенням їх роботи.

4.1 Розробка блок-схем та опис алгоритмів функціонування системи

При запуску програмного забезпечення ініціюється перевірка наявності та цілісності файлової системи SD-картки, що забезпечує збереження та відтворення аудіофайлів під час роботи системи.

Далі відбувається налаштування інтерфейсів для передачі та прийому аудіосигналів — інтерфейс I^2S конфігурується для виведення сформованої шумової послідовності, а вхід ADC призначається для зчитування даних із підключеного мікрофона;

Одночасно встановлюється зв’язок з OLED-дисплеєм для відображення інформації про стан системи та налаштування параметрів, а також активується вбудований веб-сервер, який надає користувацький інтерфейс для віддаленого керування.

Після успішного проходження процедури самодіагностики, що підтверджує готовність усіх модулів, програмне забезпечення переходить у режим очікування подій користувача та зовнішніх запитів.

У системі реалізовано чотири базові операції: вибір файлів, генерація шуму, генерація ультразвуку та запис голосу, на основі якого формується кінцевий аудіофайл із шумовим наповненням. Кожен із цих модулів супроводжується відповідною блок-схемою, що ілюструє послідовність операцій та умовні переходи між станами системи.

Блок-схема загальної структури програмного забезпечення зображена на рисунку 4.1.

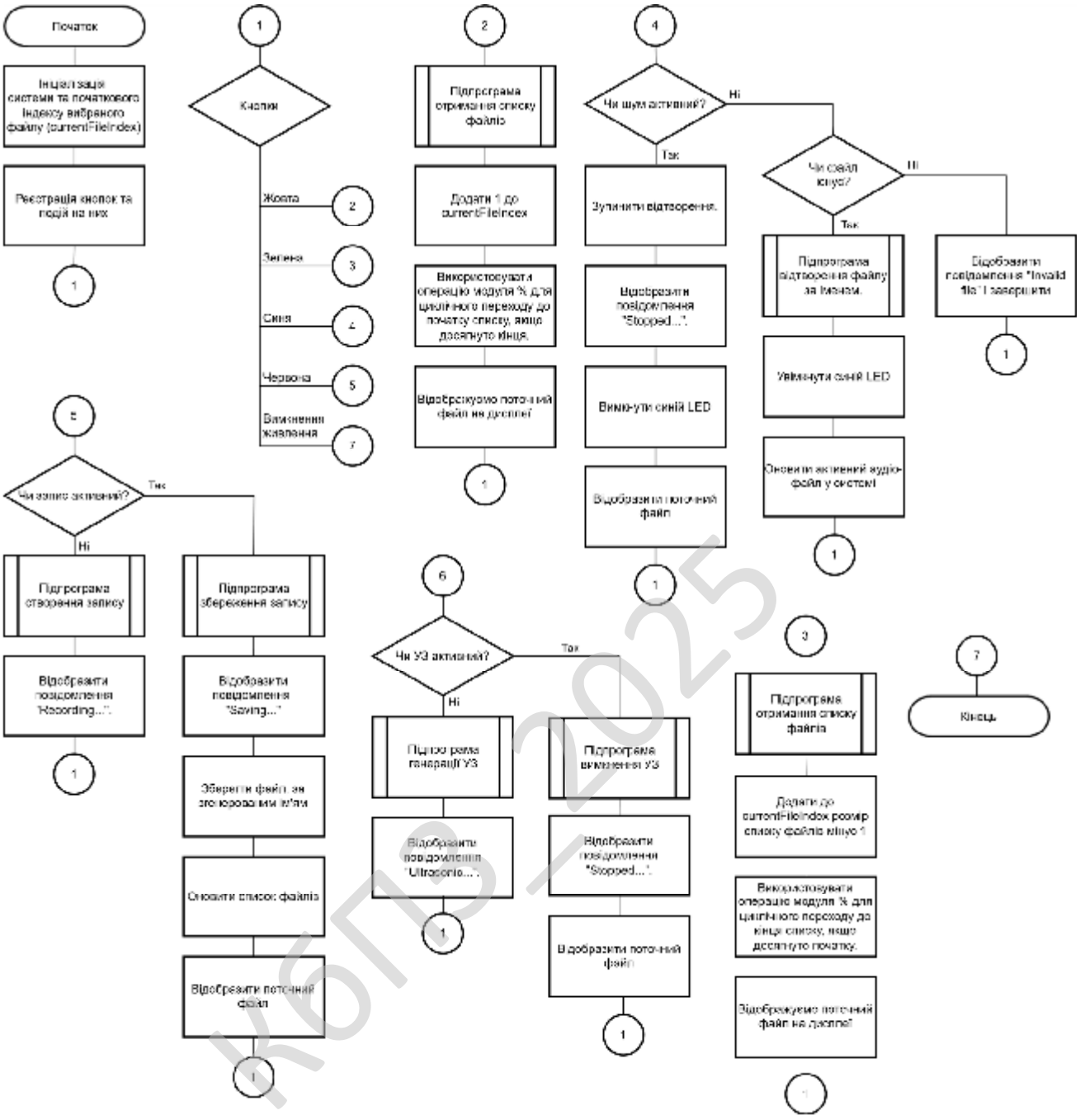

Рисунок 4.1 – Блок-схема основної програми

Першим кроком користувач обирає потрібний файл або створює новий. Алгоритм, для отримання всіх файлів, зображений на рисунку 4.2.

Рисунок 4.2 – Блок-схема підпрограми отримання файлів

Модуль вибору файлів перевіряє їх доступність на SD-карті, відображає список існуючих записів та приймає команду на програвання або створення нового файлу.

У разі вибору функції програвання фалу, система ініціює алгоритм, який зображений на рисунку 4.3.

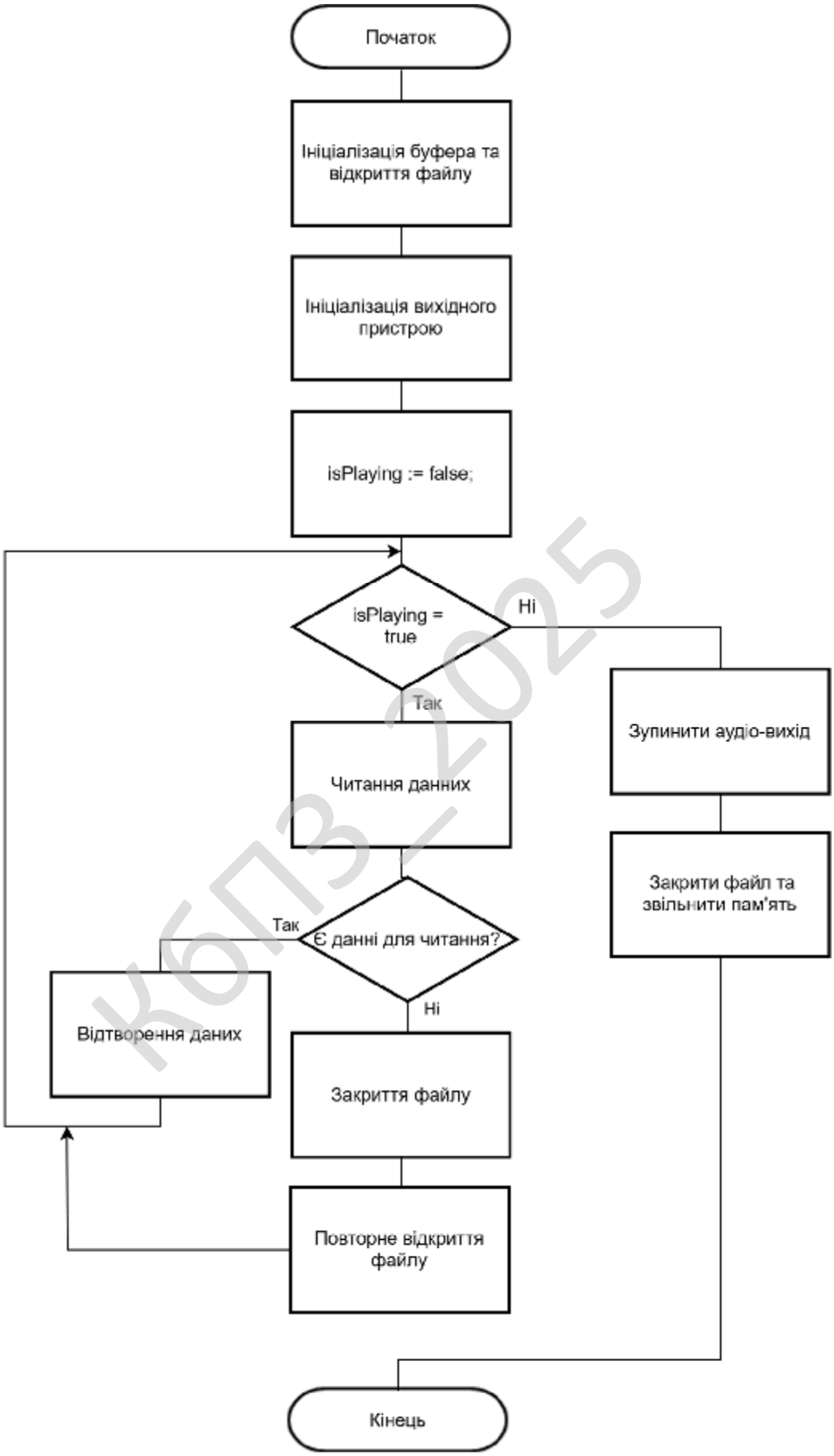

Рисунок 4.3 – Блок-схема підпрограми програвання аудіо

При увімкненні генерації ультразвуку, за допомогою веб-додатку, програмне забезпечення формує квадратну хвилю в діапазоні 20–22 кГц, що відтворюється через підсилювач і випромінюється відповідним випромінювачем. Цей режим використовується для дезорієнтації потенційного прослуховування, створюючи високочастотний захисний фон. Алгоритм ультразвукового модуля відображено на рисунку 4.4.

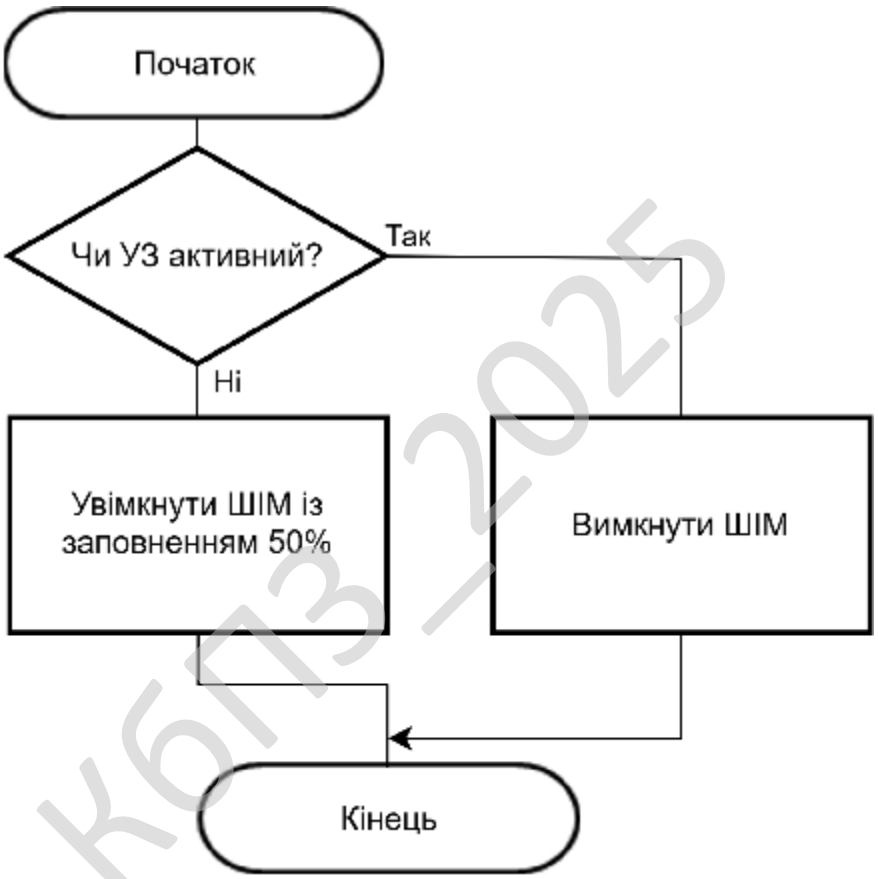

Рисунок 4.4 – Блок-схема підпрограми генерації ультразвуку

Далі йде модуль запис голосу. Він активується через апаратну кнопку або веб-команду, мікрофон зчитує сигнал за допомогою ADC, накопичує аудіодані в буфер та форматує їх у WAV-файл. Цей голосовий запис слугує основою для подальшого змішування з шумом. Після завершення запису система автоматично зливає голосовий та шумовий шари, зберігаючи фінальний файл у вибраному каталозі. Блок-схема процедури запису голосу представлена на рисунку 4.5.

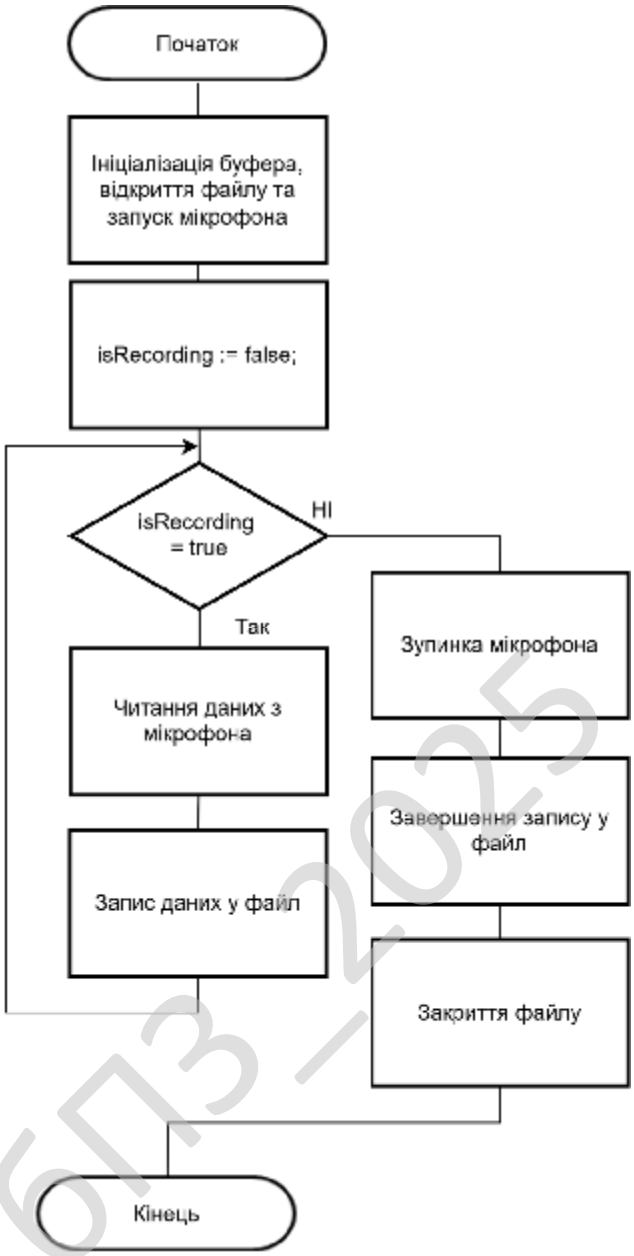

Рисунок 4.5 – Блок-схема підпрограми запису звуку

У підсумку, наведені блок-схеми ілюструють чітку послідовність роботи програмного забезпечення: від ініціалізації апаратних інтерфейсів до виконання чотирьох базових операцій — вибору файлів (рис. 4.2), відтворення WAV-файлів (рис. 4.3), генерації ультразвуку (рис. 4.4) та запису голосу з подальшим змішуванням із шумом (рис. 4.5). Така структурована архітектура забезпечує модульність, гнучкість налаштувань і надійне формування захисного шумового сигналу.

4.2 Захист розробленого програмного забезпечення

При розроблені системи кібербезпеки захисту приватності розмови особливу увагу було приділено заходам безпеки програмного забезпечення, що керує пристроєм. Її захист передбачає як технічні, так і програмні заходи, спрямовані на унеможливлення несанкціонованого доступу, модифікації або зловмисного використання функціоналу генерації шуму.

Перш за все, програмне забезпечення працює в обмеженому середовищі мікроконтролера ESP32, що вже саме по собі обмежує можливості зовнішнього впливу. Всі комунікації між веб-інтерфейсом та пристроєм захищені через локальну мережу, яка захищена паролем, що суттєво знижує ризик зовнішніх атак. Додатково, до коду реалізовані перевірки цілісності параметрів, що передаються через інтерфейс, для уникнення ін'єкційного або буферного впливу.

Усі ключові функції контролюються через натискання кнопок фізичного інтерфейсу та взаємодія через веб-додаток, який обмежуються лише базовим набором команд, що не дозволяє користувачу здійснювати критичні зміни без фізичного доступу до пристрою. Таким чином, загрози, пов'язані з віддаленим керуванням, суттєво знижуються.

Також в пристрої враховано ризики, пов'язані з компрометацією прошивки. Для цього передбачено обмеження на оновлення програмного забезпечення: прошивка пристрою може бути змінена лише через фізичне підключення до комп'ютера розробника. Зберігання бінарних файлів і вихідного коду здійснюється на захищених сховищах з резервним копіюванням, що мінімізує втрату або витік інформації.

Важливою складовою безпеки є забезпечення прозорості коду та його перевірка. Під час розробки застосовувались практики статичного аналізу та модульного тестування, що дозволяє знизити ймовірність помилок або прихованих вразливостей. Що в свою чергу підвищує надійність функціонування пристрою в реальних умовах.

5 ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Впровадження розробленої системи захисту приватності розмови шляхом генерації акустичного шуму в промислове середовище здійснюється поетапно та включає адаптацію апаратно-програмного комплексу до умов підприємства, налаштування середовища експлуатації та організаційне забезпечення роботи персоналу. Першим кроком є встановлення та конфігурація пристрою на обраних робочих місцях із підключенням до існуючої акустичної інфраструктури. Це передбачає монтаж динаміків, підсилювача та сенсорних елементів управління згідно з розробленою принциповою схемою, а також завантаження базового прошивального образу ESP32 разом із наборами конфігураційних файлів, які містять ключі шифрування та параметри генерації шуму.

На етапі пілотного запуску система розгортається на обмеженій кількості робочих станцій, де проводиться навчання операторів і тестування в реальних умовах. Працівників ознайомлюють із процедурою безпечного керування генерацією шуму та записом розмови, надаючи інструктаж щодо захисту апаратних інтерфейсів і збереження криптографічних токенів. Окрему увагу приділено роз’ясненню принципів роботи симетричного шифру «Калина» у режимі CBC, що застосовується для приховування аудіоданих під час передавання та зберігання.

Після навчання персоналу проводиться комплексне функціональне та навантажувальне тестування системи. Функціональні випробування охоплюють перевірку коректності генерації шуму в заданих частотних діапазонах та своєчасності реакції на команди користувача, тоді як модельні атаки та емуляція несанкціонованого доступу використовуються для оцінки стійкості захисних механізмів. Результати аналізу продуктивності під навантаженням дозволяють налаштувати оптимальні параметри інтервалів шифрування й оновлення ключів

для гарантування безперервності роботи та мінімізації затримок при обробці аудіосигналів.

У вихідний промисловий режим система переходить після успішного завершення всіх випробувань і затвердження звіту про приймання. Відтоді забезпечується постійний моніторинг технічного стану пристрою через вбудовані засоби логування та віддалений модуль телеметрії, що фіксує параметри роботи генератора шуму, стан шифрувальних ключів і частоту запуску процедур оновлення. Організація заходів технічної підтримки передбачає регулярні оновлення прошивки та конфігураційних файлів, аудит безпеки й аналіз журналів подій із метою оперативного виявлення відхилень від нормативних показників.

Завдяки такому структурованому підходу до впровадження забезпечується надійна інтеграція розробленої системи в промислові процеси підприємства, гарантується високий рівень кібербезпеки та стабільність роботи пристрою в умовах інтенсивного використання. Це створює передумови для масштабування рішення та його подальшого вдосконалення з урахуванням змін стандартів і вимог інформаційної безпеки.

ОСНОВНІ ВИСНОВКИ

У випускній кваліфікаційній роботі було проведено дослідження методів та засобів захисту акустичної інформації і розроблено програмне забезпечення системи кібербезпеки для захисту приватності розмови шляхом генерування шуму.

Метою роботи є розробка системи кібербезпеки та її програмного забезпечення для захисту приватності розмови шляхом генерування шуму. Для досягнення цієї мети було проведено вивчення нормативно-правової бази забезпечення захисту інформації з обмеженим доступом в Україні, здійснено аналіз каналів витоку акустичної інформації та вимог до технічних засобів захисту інформації через акустичні канали. Також було спроєктовано систему кібербезпеки технічного захисту мовної інформації шляхом генерування шуму, а на завершальному етапі розроблено та протестовано програмне забезпечення цієї системи.

Призначення такої системи захисту, це забезпечити надійний захист конфіденційних розмов, запобігаючи можливості їх прослуховування та запису. Система створює акустичні перешкоди, що маскують корисний мовний сигнал, роблячи його нерозбірливим для сторонніх осіб та пристроїв. Це особливо актуально в умовах зростання технічних можливостей засобів розвідки та прослуховування, роблячи захист приватності розмов критично важливим завданням.

Існують безліч інших методів та засобів технічного захисту акустичної інформації від витоків. До них відносяться як пасивні методи, що включають звукоізоляцію приміщень та екранування, так і активні методи, що передбачають генерацію шумових перешкод. Пасивні методи спрямовані на фізичне обмеження поширення звукових хвиль, тоді як активні методи створюють маскуючий шум, що ускладнює виділення корисного сигналу. Окрім того, використовуються методи обробки сигналів, такі як скремблювання та шифрування, для маскування змісту

мовленнєвих повідомлень. Важливим аспектом є також захист від витоку інформації через вібрації та електромагнітні випромінювання, для чого застосовуються спеціальні віброакустичні генератори та системи електромагнітного екранування. Різноманітність існуючих методів відображає складність задачі захисту акустичної інформації та необхідність комплексного підходу до її вирішення.

Наприклад, пристрої, що генерують "мовленнєподібний" шум, використовують фрагменти акустичного сигналу конфіденційної інформації, багаторазово накладаючи їх з різними рівнями для створення маскуючої перешкоди. Такий підхід реалізований, наприклад, в пристрої "PSP-2A AUTO", де фрагменти мови багаторазово накладаються один на одний, ускладнюючи розбірливість оригінального повідомлення. Інші генератори можуть формувати шум на основі фрагментів мови різних дикторів або музичних фрагментів, змішаних з випадковим шумом, забезпечуючи більш складне маскування. Для захисту від витоку інформації через вібрації, наприклад, використовуються вібровипромінювачі, що кріпляться на вікна або труби, створюючи вібраційні перешкоди, які заважають зчитуванню акустичної інформації за допомогою спеціальних пристроїв.

Існують також системи, що використовують електромагнітні завади для захисту від прослуховування за допомогою пристроїв, які можуть зчитувати інформацію з електромагнітних випромінювань.

Система працює таким чином, що генерує спеціально сформований шумовий сигнал, який додається до звукового поля в приміщенні. Цей сигнал маскує мову, роблячи її нерозбірливою для пристроїв запису та прослуховування. Розроблена система використовує сучасні технології обробки сигналів для створення ефективного шумового маскування, адаптованого до конкретних умов приміщення.

Перед впровадженням даної системи було проведено низку підготовчих дій, а саме: аналіз існуючих методів та засобів захисту акустичної інформації, вивчення

нормативно-правової бази, визначення вимог до системи захисту, розробка структурної та функціональної схеми системи, вибір апаратних та програмних засобів для реалізації системи.

Реалізації такого захисту сприяє використання сучасних технологій та обладнання. Розроблена система базується на мікроконтролері ESP32, який забезпечує високу продуктивність та функціональність. Для генерації шумових сигналів використовуються спеціальні алгоритми та методи цифрової обробки сигналів. Програмне забезпечення системи розроблено з використанням мови програмування C++, що забезпечує ефективність та надійність роботи системи.

В ході виконання роботи було досягнуто поставленої мети та вирішено ряд важливих завдань. Розроблена система кібербезпеки забезпечує ефективний захист приватності розмови шляхом генерування шуму. Система може бути використана в різних сферах діяльності, де потрібен захист конфіденційності усного спілкування.

СПИСОК ДЖЕРЕЛ

1. Про захист інформації в інформаційно-телекомунікаційних системах: Закон України від 05.07.94 № 81/94- ВР. Дата оновлення 04.07.2017. URL:https://zakon.rada.gov.ua/laws/show/80/94-%D0%B2%D1%80#Text (дата звернення: 18.05.2025).

2. Козубцова Л. М., Хлапонин Ю. І., Козубцов І. М. Методика оцінювання ефективності виконання заходів забезпечення кібербезпеки об'єктів критичної інформаційної інфраструктури організацій. Сучасні інформаційні технології у сфері безпеки та оборони. 2021. № 2 (41). С. 17 – 22.

3. Юдін О. К. Правові аспекти формування системи державних інформаційних ресурсів. Безпека інформації 2014. Том 20 (1). Технічні науки. С. 76-82.

4. ESP32. URL:https://oxorona.com/esp32/ (дата звернення: 18.05.2025).

5. Лизанчук В. В. Інформаційна безпека України: теорія і практика : підручник / Львів. нац. ун-т ім. Івана Франка, Львів. шк. журналістики. Львів: ЛНУ ім. Івана Франка, 2017. 725 с.

6. Плата розробника ESP32-WROOM-32D Wi-Fi+BT+BLE від Keyestudio. URL: https://surl.li/crplgi (дата звернення: 18.05.2025).

7. ESP-WROOM-32 ESP32 ESP-32S Development Board 2.4GHz Dual-Mode WiFi + Bluetooth Dual Cores Microcontroller Processor Integrated with Antenna RF AMP Filter AP STA Compatible with Arduino IDE (1 PCS). URL:https://surli.cc/dvmsqe (дата звернення: 18.05.2025).

8. Принцип роботи генераторів шуму та сфера їх застосування. URL:https://surl.li/hmreoa (дата звернення: 18.05.2025).

9. 7 Watt Class-A Audio Amplifier Circuit. URL:https://www.circuits-diy.com/7-watt-class-a-audio-amplifier-circuit/ (дата звернення: 18.05.2025).

10. Моделювання шуму – білий, рожевий та коричневий шум, попси та тріски. URL:https://surl.li/hkyziz (дата звернення: 18.05.2025).

11. Білий шум / Вікіпедія. Вільна енциклопедія. URL:https://surli.cc/nzvbjm (дата звернення: 18.05.2025).

12. Рожевий шум / Вікіпедія. Вільна енциклопедія. URL:https://surl.li/ecdtfj (дата звернення: 18.05.2025).

13. Червоний шум / Вікіпедія. Вільна енциклопедія [Електронний ресурс]. URL:https://surl.lu/gaxbds (дата звернення: 18.05.2025).

14. Електронні системи. Білий шум. URL:https://web.posibnyky.vntu.edu.ua/firen/6bilynskyj_elektronni_systemy/34.htm (дата звернення: 18.05.2025).

15. Канали витоку інформації / Вікіпедія. Вільна енциклопедія. URL:https://surl.li/mvqdrn (дата звернення: 18.05.2025).

16. Захист інформації від витоку технічними каналами. URL: https://tzi.com.ua/zaxist-nformacz-vd-vitoku-texnchnimi-kanalami.html (дата звернення: 18.05.2025).

17. Акустика слуху: конспект лекцій [Електронний ресурс]: навч. посіб. для студ. спеціальності 171 «Електроніка», освітньої програми «Акустичні електронні системи та технології обробки акустичної інформації» / КПІ ім. Ігоря Сікорського; уклад.: К. С. Дрозденко, О. І. Дрозденко. – Електронні текстові данні (1 файл: 7,67 Мбайт). – Київ: КПІ ім. Ігоря Сікорського, 2020. – 99 с. URL:https://surl.li/tbxtwr (дата звернення: 18.05.2025).

18. Сінюгін В. В., Катаєв В.С., Грицак А. В. Модульний генератор шуму для блокування витоку акустичної інформації. Вісник Вінницького політехнічного інституту. 2021. № 6. С. 158-164. URL:https://ir.lib.vntu.edu.ua/handle/123456789/35321 (дата звернення: 18.05.2025).

19. Іванченко С.О., Гавриленко О.В., Липський О.А., Шевцов А.С.Технічні канали витоку інформації. Порядок створення комплексів

технічного захисту інформації: навчальний посібник. Київ: ІСЗЗІ НТУУ «КПІ», 2016. 104 с.

20. Закон України № 2657-XII від 02.10.1992 Про інформацію. URL:https://zakononline.com.ua/documents/show/151399___591480 (дата звернення: 18.05.2025).

21. Створення комплексних систем захисту інформації. URL:https://tzi.com.ua/stvorennya-kompleksnix-sistem-zaxistu-nformacz.html (дата звернення: 18.05.2025).

22. Про доступ до публічної інформації: Закон України. Відомості Верховної Ради України. 2011. № 32, ст. 314. URL:https://zakon.rada.gov.ua/laws/main/2939-17#Text (дата звернення: 18.05.2025).

23. Положення про технічний захист інформації в Україні: Указ Президента України від 27 вересня 1999 року № 1229/99. URL:https://zakon.rada.gov.ua/laws/show/1229/99#Text (дата звернення: 18.05.2025).

24. Моргулець О. Б. Менеджмент у сфері послуг. Навч. посіб. Київ: Центр учбової літератури, 2012. 384 с. URL:https://surl.li/lprett (дата звернення: 18.05.2025).

25. Гриценко Т.Б. Етика ділового спілкування: навч. посіб. Київ: Центр учбової літератури. 2007. 344 с.

26. Про затвердження Концепції технічного захисту інформації в Україні: Постанова Кабінету Міністрів України від 08 жовтня 1997 року №1126 URL:https://zakon.rada.gov.ua/laws/show/1126-97-%D0%BF#Text (дата звернення: 18.05.2025).

27. ДСТУ 3396.2-97. Захист інформації. Технічний захист інформації. Терміни та визначення. Київ: Держстандарт України, 1997.

28. ДСТУ 3396.0-96. Захист інформації. Технічний захист інформації. Основні положення. Київ: Держстандарт України, 1996.

29. Конституція України. Відомості Верховної Ради України (ВВР), 1996, №30, ст.141 URL:https://zakon.rada.gov.ua/laws/show/254%D0%BA/96-%D0%B2%D1%80#Text (дата звернення: 18.05.2025).

30. Про захист інформації в інформаційно-комунікаційних системах. Відомості Верховної Ради України (ВВР), 1994, № 31, ст.286. URL: https://zakon.rada.gov.ua/laws/show/80/94-%D0%B2%D1%80#Text (дата звернення: 18.05.2025).

31. Про державну таємницю. Відомості Верховної Ради України (ВВР), 1994, № 16, ст.93. URL: https://zakon.rada.gov.ua/laws/show/3855-12#Text

32. Про науково-технічну інформацію. Відомості Верховної Ради України (ВВР), 1993, № 33, ст.345. URL:https://zakon.rada.gov.ua/laws/show/3322-12#Text (дата звернення: 18.05.2025).

33. Перелік документів нормативно-правової бази, що забезпечує надання відповідних видів послуг у галузі технічного захисту інформації. URL:https://cip.gov.ua/ua/news/perelik-dokumentiv-normativno-pravovoyi-bazi-sho-zabezpechuye-nadannya-vidpovidnikh-vidiv-poslug-u-galuzi-tekhnichnogo-zakhistu-informaciyi (дата звернення: 18.05.2025).

34. ДСТУ ISO/IEC 27001:2023 Інформаційна безпека, кібербезпека та захист конфіденційності. Системи керування інформаційною безпекою. Вимоги (ISO/IEC 27001:2022, IDT): Наказ від 17.08.2023 № 210 Про прийняття національних стандартів, зміни до національного стандарту та скасування національних стандартів. URL:https://online.budstandart.com/ua/catalog/doc-page.html?id_doc=104398 (дата звернення: 18.05.2025).

35. Павленко Є.В. Оцінка можливості перехоплення мовної інформації із виділеного приміщення та способи його захисту від витоків інформації технічними каналами: дис…канд..н. Київ: 2024. 150 с. URL: https://surl.lu/gohpza (дата звернення: 18.05.2025).

36. Аль-Амморі А. Н., Дехтяр М. М., Іщенко Р. М., Клочан А. Є. Методи та засоби захисту інформації. Системи управління навігації та зв'язку: Зб. наук.

праць. 2024. 1(75). С.38-44. URL: https://surl.li/rfhahx (дата звернення: 18.05.2025).

37. Лужецький В.А., Кожухівський А.Д., Войтович О.П. Основи інформаційної безпеки: Навч. посіб. Вінниця: ВНТУ, 2009. 268 с. URL: https://studfile.net/preview/6012701/ (дата звернення: 18.05.2025).

38. ДСТУ 3396.1-96. Захист інформації. Технічний захист інформації. Порядок проведення робіт. Затверджено наказом Держстандарту України від 19.12.96 р. № 51, введено в дію 01.01.97 р. URL: https://online.budstandart.com/ua/catalog/doc-page.html?id_doc=69173 (дата звернення: 18.05.2025).

39. Нормативний документ системи технічного захисту інформації Р-001-2000. Засоби активного захисту мовної інформації з акустичними та віброакустичними джерелами випромінювання. Класифікація та загальні технічні вимоги. Рекомендації. ДСТСЗІ СБ України, Київ, 2000. Чинний від 01.10.2000. 8 с. URL:https://usts.kiev.ua/wp-content/uploads/2020/07/nd-tzi-r-001-2000.pdf (дата звернення: 18.05.2025).

40. Нормативний документ системи технічного захисту інформації 1.1-003-99. Термінологія в галузі захисту інформації в комп'ютерних системах від несанкціонованого доступу. ДСТСЗІ СБ України, Київ, 1999. Чинний від 01.07.1999. 22 с. URL:https://tzi.ua/assets/files/1.1_003_99.pdf (дата звернення: 18.05.2025).

41. Технічні засоби захисту інформації: опорний конспект лекцій з дисципліни для студентів освітньо-професійної програми підготовки бакалавра галузі знань 12 Інформаційні спеціальності 125 «Кібербезпека» / Укл.: Яцків В.В., Кулина С.В. Тернопіль, 2023. 88 с.

42. Інструкція з технічного забезпечення засобами технічного захисту інформації і комплексами технічного контролю Національної гвардії України: затв. Наказ МВСУ від 27.07.2015 №906. URL:https://zakon.rada.gov.ua/laws/show/z0999-15?utm_source=chatgpt.com#Text (дата звернення: 18.05.2025).

43. Вібровипромінювач ВІ-3 (для вікна).

URL:https://tzi.ua/ua/vbrovipromnyuvach_v-3_dlya_vkna.html (дата звернення: 18.05.2025).

44. Вібровипромінювач ВИ4 (для стіни, стелі, підлоги та системи опалення). URL:https://tzi.com.ua/vi4.html (дата звернення: 18.05.2025).

45. Колонка акустична захищена МАРС-АКЗ. URL:https://tzi.com.ua/mars-ak3.html (дата звернення: 18.05.2025).

46. Генератор шумових сигналів МАРС-ТЗО-4-2. URL:https://tzi.com.ua/mars-tzo42.html (дата звернення: 18.05.2025).

47. Випромінювач акустичний «Базальт-4ДА». URL: https://usts.kiev.ua/vyprominiuvach-akustychnyj-bazalt-4da/ (дата звернення: 18.05.2025).

48. ANG-2200 Acoustic Noise Generator. URL:https://reiusa.net/audio-security/ang-2200-acoustic-noise-generator/ (дата звернення: 18.05.2025).

49. Методичні вказівки до виконання лабораторних робіт з дисциплін «Фізико-технічні методи захисту інформації» для студентів спеціальностей 6.170101 і 6.170102 всіх форм навчання / укладач Клочко В.В., Слєпцов В.І. Запоріжжя: ЗНТУ, 2008. 32 с. URL: https://surl.lu/uffwof (дата звернення: 18.05.2025).

50. PSP-2A AUTO - Pocket Speech Protector Acoustic (2). URL: http://intevro.com.ua/means_of_information_protection/psp-2a_auto.html?utm_source=chatgpt.com (дата звернення: 18.05.2025).

51. Генератор акустического шума стационарный "РИАС-2ГС" / Відеокамери.com.ua. URL:https://xn--80adgebslrpy8u.com.ua/ru/henerator-akustychnoho-shumu-statsionarnyi-rias2hs/ (дата звернення: 18.05.2025).

Додаток А

Технічне завдання

ЗМІСТ

					ВКРБ-125.25.0004.00.00.ТЗ	
Вим.	Арк	№ докум.	Підпис	Дата		
Розроб.		Задорожний К.О.			*Програмне забезпечення системи кібербезпеки захисту приватності розмови шляхом генерування шуму*	Літ. К / Аркуш 1 / Аркушів 6
Перевір.		Дрєєв О.М.				
Н. контр.		*Коваленко А. С.*				*ЦНТУ КБ-21*
Затв.		*Смірнов О. А.*				

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку системи кібербезпеки захисту приватності розмови шляхом генерування шуму.

2 Підстава для розробки

Підставою для розробки служить завдання на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, видане на кафедрі кібербезпеки та програмного забезпечення (нак. № 57-02 від 17.01.2025 року).

3 Мета та призначення розробки

Метою випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти є розробка програмне забезпечення системи кібербезпеки захисту приватності розмови шляхом генерування шуму.

4 Джерела розробки

Джерелом цієї випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти є стосовна до теми літератури та існуючих аналогів.

5 Технічні вимоги

5.1 Вміст проекту

Складовими розробки є:

– огляд та аналіз аналогічних існуючих систем на предмет їхньої відповідності сучасним вимогам.

– вибір і обґрунтування вибору засобів для побудови системи та мови програмування;

– опис і розробка функціональних, структурних схем та діаграм процесів.

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– системи кібербезпеки захисту приватності розмови шляхом генерування шуму;

– цілісність даних у процесі роботи та при зберіганні;

– простий, інтуїтивно зрозумілий інтерфейс.

5.3 Вимоги до функціональних характеристик

Система повинна в реальному часі генерувати мовоподібний шум із можливістю динамічного перемикання режимів, підтримувати одночасну роботу до чотирьох незалежних вихідних каналів із запису голосу користувача.

5.4 Вимоги до архітектури

Архітектура базується на клієнт-серверній моделі: вбудований контролер ESP32 створює веб сервер, який надає веб-інтерфейс на React. Системні модулі (генератор шуму, комунікаційний блок) мають чітко визначені API, що забезпечує масштабованість та можливість інтеграції нових типів випромінювачів без зміни основної структури.

5.5 Вимоги до надійності

Генерація шуму повинна продовжуватися щонайменше 30 хвилин у разі втрати зв’язку; некритичні збої контролера автоматично усуваються перезапуском; дані конфігурації захищені CRC32; юніт-тести покривають не менше 85 % коду, а інтеграційні випробування перевіряють ключові режими роботи.

5.6 Умови експлуатації

Пристрій призначений для роботи при температурі 0–40 °C, вологості до 80 % без конденсації та атмосферному тиску 84–106 кПа. Живлення здійснюється від 12 В постійного струму (±10 %) або через блок 100–240 В змінного струму.

5.7 Вимоги до складу та параметрів технічних засобів

Вбудований контролер має бути оснащений мінімум 8 МБ флеш-пам’яті та 512 КБ ОЗП, має ЦАП 12 бит/48 кГц, підсилювач потужністю до 5 Вт на канал, Wi-Fi 802.11 b/g/n і UART для налагодження. Динаміки або вібровипромінювачі повинні забезпечувати рівень шуму не менше 75 дБ на відстані 1 м.

5.8 Вимоги до інформаційної і програмної сумісності

Система має бути реалізована на базі мікроконтролера ESP32 із наступними апаратними інтерфейсами: I²S для виведення шумових сигналів, ADC для зчитування мікрофонного входу, I²C для керування OLED-індикатором та Digital I/O для кнопкового інтерфейсу й LED-індикації; налагодження та завантаження прошивки здійснюється через UART, віддалене керування — через вбудований веб-сервер по Wi-Fi.

Прошивка написана на C/C++ (ESP-IDF), серверна частина веб-інтерфейсу — на C++ клієнтська частина — на JavaScript (React). Обмін даними між контролером і сервером організовано у форматі JSON через HTTPS; оновлення прошивки можливе лише через захищений UART-інтерфейс із цифровим підписом образу

5.8.1 Обладнання

Мікроконтролер ESP32, оснащений вбудованим ЦАП/I²S-інтерфейсом для виведення шуму, ADC для зчитування сигналу з мікрофона та WIFI модулем, чотирма тактовими кнопками з LED-індикацією і OLED-дисплеєм, модулем SD-card для Arduino (SPI), підсилювачем звуку TDA7297 (12 В) і стабілізованим живленням 3,3 В/5 В; для розробки і налагодження використовуються UART-інтерфейс і персональний комп’ютер із USB-UART переходником .

5.8.2 Мова програмування

Середовище PlatformIO.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи. Схема запропонованого пристрою

5.8.4 Вихідні дані

Робоча програма та пристрій.

6 Вимоги до програмної документації

Програмна продукція повинна бути представлена у вигляді опису структури даних, схем, алгоритмів та діаграм, а також текстів вихідних модулів програмного забезпечення відповідно до ЄСПД.

7 Перелік документів, що розробляються

– Структурна схема системи – 1 аркуш.
– Функціональна схема системи – 1 аркуш.
– Принципова схема пристрою – 1 аркуш.
– Діаграма процесів – 1 аркуш.
– Блок-схема алгоритму роботи програми – 5 аркушів.
– Пояснювальна записка – 75 аркушів.

8 Етапи розробки

8.1 Збір і обробка інформації за темою випускної кваліфікаційної роботи першого (бакалаврського) рівня вищої освіти. Формулювання задачі для виконання випускної кваліфікаційної роботи першого (бакалаврського) рівня вищої освіти (розробка технічного завдання).

8.2 Проведення досліджень або експериментальних робіт для уточнення основних положень випускної кваліфікаційної роботи першого (бакалаврського) рівня вищої освіти.

8.3 Розробка функціональних схем, блок-схем та алгоритмів роботи програмного забезпечення.

8.4 Побудова схем взаємодії даних.

8.5 Створення прототипу програмного забезпечення.

8.6 Тестування програмного забезпечення, аналіз отриманих результатів.

8.7 Оформлення пояснювальної записки та виконання робіт, пов'язаних з графічною частиною.

9 Порядок контролю та приймання

9.1 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на попередній захист 24.05.2025 р.

9.2 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на захист 02.06.2025 р.

ДОДАТОК Б

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти

_____ Дрєєв О.М.

Програмне забезпечення системи кібербезпеки захисту приватності розмови шляхом генерування шуму

Лістинг програми

Код документу 12

Носій: USB-флеш-накопичувач

Загальна кількість аркушів: 51

Літера: РП

Кропивницький - 2025 року

Основна програма

Файл components.json

```
{
  "$schema": "https://ui.shadcn.com/schema.json",
  "style": "new-york",
  "rsc": false,
  "tsx": true,
  "tailwind": {
    "config": "",
    "css": "src/styles.css",
    "baseColor": "zinc",
    "cssVariables": true,
    "prefix": ""
  },
  "aliases": {
    "components": "@/components",
    "utils": "@/lib/utils",
    "ui": "@/components/ui",
    "lib": "@/lib",
    "hooks": "@/hooks"
  },
  "iconLibrary": "lucide"
}
```

Файл index.html

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link rel="icon" href="/favicon.ico" />
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Noise generator controller"
    />
    <link rel="apple-touch-icon" href="/logo192.png" />
    <link rel="manifest" href="/manifest.json" />
    <title>Noise generator controller</title>
  </head>
  <body>
    <div id="app"></div>
    <script type="module" src="/src/main.tsx"></script>
  </body>
</html>
```

Файл package.json

```
{
  "name": "noise-generator-web",
  "private": true,
  "type": "module",
  "scripts": {
    "dev": "vite --port 3000",
    "start": "vite --port 3000",
    "build": "vite build && tsc",
    "serve": "vite preview",
    "test": "vitest run",
    "lint": "eslint",
    "format": "prettier",
    "check": "prettier --write . && eslint --fix"
  },
  "dependencies": {
    "@radix-ui/react-context-menu": "^2.2.12",
    "@radix-ui/react-slot": "^1.2.0",
    "@radix-ui/react-switch": "^1.2.2",
    "@radix-ui/react-tooltip": "^1.2.4",
    "@tailwindcss/vite": "^4.0.6",
    "@tanstack/react-query": "^5.66.5",
    "@tanstack/react-query-devtools": "^5.66.5",
    "@tanstack/react-router": "^1.114.3",
    "@tanstack/react-router-devtools": "^1.114.3",
    "class-variance-authority": "^0.7.1",
    "clsx": "^2.1.1",
    "lucide-react": "^0.476.0",
    "react": "^19.0.0",
    "react-dom": "^19.0.0",
    "tailwind-merge": "^3.0.2",
    "tailwindcss": "^4.0.6",
    "tailwindcss-animate": "^1.0.7",
    "zustand": "^5.0.4"
  },
  "devDependencies": {
    "@tanstack/eslint-config": "^0.1.0",
    "@testing-library/dom": "^10.4.0",
    "@testing-library/react": "^16.2.0",
    "@types/react": "^19.0.8",
    "@types/react-dom": "^19.0.3",
    "@vitejs/plugin-react": "^4.3.4",
    "jsdom": "^26.0.0",
    "prettier": "^3.5.3",
    "typescript": "^5.7.2",
    "vite": "^6.1.0",
    "vitest": "^3.0.5",
    "web-vitals": "^4.2.4"
  }
}
```

Файл prettier.config.js

```
//  @ts-check

/** @type {import('prettier').Config} */
const config = {
  semi: false,
  singleQuote: true,
  trailingComma: 'all',
}
export default config
```

Файл tsconfig.json

```
{
  "include": [
    "**/*.ts",
    "**/*.tsx",
    "eslint.config.js",
    "prettier.config.js",
    "vite.config.js"
  ],
  "compilerOptions": {
    "target": "ES2022",
    "jsx": "react-jsx",
    "module": "ESNext",
    "lib": ["ES2022", "DOM", "DOM.Iterable"],
    "types": ["vite/client"],

    /* Bundler mode */
    "moduleResolution": "bundler",
    "allowImportingTsExtensions": true,
    "verbatimModuleSyntax": true,
    "noEmit": true,

    /* Linting */
    "skipLibCheck": true,
    "strict": true,
    "noUnusedLocals": true,
    "noUnusedParameters": true,
    "noFallthroughCasesInSwitch": true,
    "noUncheckedSideEffectImports": true,
    "baseUrl": ".",
    "paths": {
      "@/*": ["./src/*"]
    }
  }
}
```

Файл vite.config.js

```
import { resolve } from 'node:path'
import { defineConfig } from 'vite'
import viteReact from '@vitejs/plugin-react'
import tailwindcss from '@tailwindcss/vite'


// https://vitejs.dev/config/
export default defineConfig({
  plugins: [viteReact(), tailwindcss()],
  test: {
    globals: true,
    environment: 'jsdom',
  },
  resolve: {
    alias: {
      '@': resolve(__dirname, './src'),
    },
  },
})
```

Файл src\App.test.tsx

```
import { describe, expect, test } from 'vitest'
import { render, screen } from '@testing-library/react'
import App from './App.tsx'

describe('App', () => {
  test('renders', () => {
    render(<App />)
    expect(screen.getByText('Learn React')).toBeDefined()
  })
})
```

Файл src\App.tsx

```
// src/App.tsx
import { AudioList } from './components/features/audio/AudioList';
import Recorder from './components/features/recorder/Recorder';
import UltrasonicControl from
'./components/features/ultrasonic/UltrasonicControl';

function App() {
  return (
    <div className="min-h-screen flex flex-col items-center bg-gray-50 p-4">
      <div className="w-full max-w-md bg-white rounded-lg shadow-md p-4
space-y-4">
        <Recorder />
        <UltrasonicControl />
        <AudioList />
      </div>
    </div>
  );
}

export default App;
```


Файл src\main.tsx

```
import { StrictMode } from 'react'
import ReactDOM from 'react-dom/client'
import {
  Outlet,
  RouterProvider,
  createRootRoute,
  createRoute,
  createRouter,
} from '@tanstack/react-router'
import { TanStackRouterDevtools } from '@tanstack/react-router-devtools'
import DemoTanstackQuery from './routes/demo.tanstack-query'

import TanstackQueryLayout from './integrations/tanstack-query/layout'

import * as TanstackQuery from './integrations/tanstack-query/root-provider'

import './styles.css'
import reportWebVitals from './reportWebVitals.ts'

import App from './App.tsx'
import NoiseGeneratorProvider from './providers/NoiseGeneratorProvider.tsx'

const rootRoute = createRootRoute({
  component: () => (
```

```
    <>
      <Outlet />
      <TanStackRouterDevtools />
      <TanstackQueryLayout />
    </>
  ),
})

const indexRoute = createRoute({
  getParentRoute: () => rootRoute,
  path: '/',
  component: App,
})

const routeTree = rootRoute.addChildren([
  indexRoute,
  DemoTanstackQuery(rootRoute),
])

const router = createRouter({
  routeTree,
  context: {
    ...TanstackQuery.getContext(),
  },
  defaultPreload: 'intent',
  scrollRestoration: true,
  defaultStructuralSharing: true,
  defaultPreloadStaleTime: 0,
})

declare module '@tanstack/react-router' {
  interface Register {
    router: typeof router
  }
}

const rootElement = document.getElementById('app')
if (rootElement && !rootElement.innerHTML) {
  const root = ReactDOM.createRoot(rootElement)
  root.render(
    <StrictMode>
      <TanstackQuery.Provider>
      <NoiseGeneratorProvider>
        <RouterProvider router={router} />
      </NoiseGeneratorProvider>
      </TanstackQuery.Provider>
    </StrictMode>,
  )
}

// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
reportWebVitals()
```

Файл src\reportWebVitals.ts

```
const reportWebVitals = (onPerfEntry?: () => void) => {
  if (onPerfEntry && onPerfEntry instanceof Function) {
    import('web-vitals').then(({ onCLS, onINP, onFCP, onLCP, onTTFB }) => {
      onCLS(onPerfEntry)
      onINP(onPerfEntry)
      onFCP(onPerfEntry)
```

```
      onLCP(onPerfEntry)
      onTTFB(onPerfEntry)
    })
  }
}

export default reportWebVitals
```

Файл src\styles.css

```
@import 'tailwindcss';

@plugin "tailwindcss-animate";

@custom-variant dark (&:is(.dark *));

body {
  @apply m-0;
  font-family:
    -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto', 'Oxygen',
'Ubuntu',
    'Cantarell', 'Fira Sans', 'Droid Sans', 'Helvetica Neue', sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
}

code {
  font-family:
    source-code-pro, Menlo, Monaco, Consolas, 'Courier New', monospace;
}

:root {
  --background: oklch(1 0 0);
  --foreground: oklch(0.141 0.005 285.823);
  --card: oklch(1 0 0);
  --card-foreground: oklch(0.141 0.005 285.823);
  --popover: oklch(1 0 0);
  --popover-foreground: oklch(0.141 0.005 285.823);
  --primary: oklch(0.21 0.006 285.885);
  --primary-foreground: oklch(0.985 0 0);
  --secondary: oklch(0.967 0.001 286.375);
  --secondary-foreground: oklch(0.21 0.006 285.885);
  --muted: oklch(0.967 0.001 286.375);
  --muted-foreground: oklch(0.552 0.016 285.938);
  --accent: oklch(0.967 0.001 286.375);
  --accent-foreground: oklch(0.21 0.006 285.885);
  --destructive: oklch(0.577 0.245 27.325);
  --destructive-foreground: oklch(0.577 0.245 27.325);
  --border: oklch(0.92 0.004 286.32);
  --input: oklch(0.92 0.004 286.32);
  --ring: oklch(0.871 0.006 286.286);
  --chart-1: oklch(0.646 0.222 41.116);
  --chart-2: oklch(0.6 0.118 184.704);
  --chart-3: oklch(0.398 0.07 227.392);
  --chart-4: oklch(0.828 0.189 84.429);
  --chart-5: oklch(0.769 0.188 70.08);
  --radius: 0.625rem;
  --sidebar: oklch(0.985 0 0);
  --sidebar-foreground: oklch(0.141 0.005 285.823);
  --sidebar-primary: oklch(0.21 0.006 285.885);
  --sidebar-primary-foreground: oklch(0.985 0 0);
  --sidebar-accent: oklch(0.967 0.001 286.375);
  --sidebar-accent-foreground: oklch(0.21 0.006 285.885);
```

```
  --sidebar-border: oklch(0.92 0.004 286.32);
  --sidebar-ring: oklch(0.871 0.006 286.286);
}

.dark {
  --background: oklch(0.141 0.005 285.823);
  --foreground: oklch(0.985 0 0);
  --card: oklch(0.141 0.005 285.823);
  --card-foreground: oklch(0.985 0 0);
  --popover: oklch(0.141 0.005 285.823);
  --popover-foreground: oklch(0.985 0 0);
  --primary: oklch(0.985 0 0);
  --primary-foreground: oklch(0.21 0.006 285.885);
  --secondary: oklch(0.274 0.006 286.033);
  --secondary-foreground: oklch(0.985 0 0);
  --muted: oklch(0.274 0.006 286.033);
  --muted-foreground: oklch(0.705 0.015 286.067);
  --accent: oklch(0.274 0.006 286.033);
  --accent-foreground: oklch(0.985 0 0);
  --destructive: oklch(0.396 0.141 25.723);
  --destructive-foreground: oklch(0.637 0.237 25.331);
  --border: oklch(0.274 0.006 286.033);
  --input: oklch(0.274 0.006 286.033);
  --ring: oklch(0.442 0.017 285.786);
  --chart-1: oklch(0.488 0.243 264.376);
  --chart-2: oklch(0.696 0.17 162.48);
  --chart-3: oklch(0.769 0.188 70.08);
  --chart-4: oklch(0.627 0.265 303.9);
  --chart-5: oklch(0.645 0.246 16.439);
  --sidebar: oklch(0.21 0.006 285.885);
  --sidebar-foreground: oklch(0.985 0 0);
  --sidebar-primary: oklch(0.488 0.243 264.376);
  --sidebar-primary-foreground: oklch(0.985 0 0);
  --sidebar-accent: oklch(0.274 0.006 286.033);
  --sidebar-accent-foreground: oklch(0.985 0 0);
  --sidebar-border: oklch(0.274 0.006 286.033);
  --sidebar-ring: oklch(0.442 0.017 285.786);
}

@theme inline {
  --color-background: var(--background);
  --color-foreground: var(--foreground);
  --color-card: var(--card);
  --color-card-foreground: var(--card-foreground);
  --color-popover: var(--popover);
  --color-popover-foreground: var(--popover-foreground);
  --color-primary: var(--primary);
  --color-primary-foreground: var(--primary-foreground);
  --color-secondary: var(--secondary);
  --color-secondary-foreground: var(--secondary-foreground);
  --color-muted: var(--muted);
  --color-muted-foreground: var(--muted-foreground);
  --color-accent: var(--accent);
  --color-accent-foreground: var(--accent-foreground);
  --color-destructive: var(--destructive);
  --color-destructive-foreground: var(--destructive-foreground);
  --color-border: var(--border);
  --color-input: var(--input);
  --color-ring: var(--ring);
  --color-chart-1: var(--chart-1);
  --color-chart-2: var(--chart-2);
  --color-chart-3: var(--chart-3);
  --color-chart-4: var(--chart-4);
  --color-chart-5: var(--chart-5);
```

```
  --radius-sm: calc(var(--radius) - 4px);
  --radius-md: calc(var(--radius) - 2px);
  --radius-lg: var(--radius);
  --radius-xl: calc(var(--radius) + 4px);
  --color-sidebar: var(--sidebar);
  --color-sidebar-foreground: var(--sidebar-foreground);
  --color-sidebar-primary: var(--sidebar-primary);
  --color-sidebar-primary-foreground: var(--sidebar-primary-foreground);
  --color-sidebar-accent: var(--sidebar-accent);
  --color-sidebar-accent-foreground: var(--sidebar-accent-foreground);
  --color-sidebar-border: var(--sidebar-border);
  --color-sidebar-ring: var(--sidebar-ring);
}

@layer base {
  * {
    @apply border-border outline-ring/50;
  }
  body {
    @apply bg-background text-foreground;
  }
}
```

Файл src\hooks\useAudioList.ts

```
import { fetcher } from '@/lib/utils'
import { useNoiseGeneratorStore } from '@/stores/useNoiseGeneratorStore'
import { useMutation } from '@tanstack/react-query'

export type AudioFile = {
  id: number
  name: string
}

const togglePlaybackAudio = async (id: number) => {
  return fetcher<void>('/toggle-playback', {
    method: 'POST',
    body: new URLSearchParams({
      id: id.toString(),
    }),
  })
}

export const useAudioList = () => {
  const { audioFiles, setAudioFiles, activeAudioId, setActiveAudioId } =
    useNoiseGeneratorStore()

  const playMutation = useMutation({
    mutationFn: togglePlaybackAudio,
  })

  const deleteMutation = useMutation({
    mutationFn: async (id: number) => {
      return fetcher<void>(`/delete-audio/${id}`, {
        method: 'DELETE',
      })
    },
    onSuccess: (_, id) => {
      if(!audioFiles) return;
      setAudioFiles(audioFiles.filter((file) => file.id !== id))
    },
  })
```

```
  const handleStop = async (id: AudioFile['id']) => {
    setActiveAudioId(null)
    await playMutation.mutateAsync(id)
  }

  const handlePlay = async (id: AudioFile['id']) => {
    if (id === activeAudioId) return handleStop(id)
    setActiveAudioId(id)
    await playMutation.mutateAsync(id)
  }

  const handleDelete = async (id: AudioFile['id']) => {
    await deleteMutation.mutateAsync(id)
  }

  return {
    data: audioFiles ?? [],
    isLoading: !audioFiles,
    isError: false,
    activeAudioId,
    play: handlePlay,
    stop: handleStop,
    deleteAudio: handleDelete,
    playState: playMutation,
    deleteState: deleteMutation,
  }
}
```

Файл src\hooks\useRecording.ts

```
import { useMutation } from '@tanstack/react-query';
import { fetcher } from '@/lib/utils';
import { useNoiseGeneratorStore } from '@/stores/useNoiseGeneratorStore';

const toggleRecoding = async (filename: string) => {
  if (!filename || typeof filename !== 'string') {
    throw new Error('Filename must be a valid string');
  }

  return fetcher<void>('/start-recording', {
    method: 'POST',
    body: new URLSearchParams({
      filename: filename,
    }),
  });
};

const useRecording = () => {
  const { isRecording, setIsRecording } = useNoiseGeneratorStore()
  const toggleRecordingMutation = useMutation({
    mutationFn: toggleRecoding,
  });

  const toggleRecordingHandler = async (filename: string) => {
    try {
      await toggleRecordingMutation.mutateAsync(filename);
      setIsRecording(true);
    } catch (error) {
      console.error('Error starting recording:', error);
    }
  };

  return {
```

```
    isRecording: isRecording,
    toggleRecordingHandler,
    isToggle: toggleRecordingMutation.isPending,
    toggleError: toggleRecordingMutation.isError,
  };
};

export default useRecording;
```

Файл src\hooks\useUltrasonicControl.ts

```
import { useMutation } from '@tanstack/react-query'
import { fetcher } from '@/lib/utils'
import { useNoiseGeneratorStore } from '@/stores/useNoiseGeneratorStore'

const toggleUltrasonic = async (isActive: boolean) => {
  return fetcher<void>('/toggle-ultrasonic', {
    method: 'POST',
    body: new URLSearchParams({
      isActive: isActive.toString(),
    }),
  })
}

export const useUltrasonicControl = () => {
  const { isUltrasonic, setIsUltrasonic } = useNoiseGeneratorStore()

  const {
    mutateAsync: toggle,
    isPending,
    isError,
  } = useMutation({
    mutationFn: toggleUltrasonic,
  })
  const handleToggle = async () => {
    await toggle(!isUltrasonic)
    setIsUltrasonic(!isUltrasonic)
  }

  return {
    isUltrasonic,
    toggle: handleToggle,
    isLoading: isPending,
    isError: isError,
  }
}
```

Файл src\lib\utils.ts

```
import {  clsx } from 'clsx'
import { twMerge } from 'tailwind-merge'
import type {ClassValue} from 'clsx';

export function cn(...inputs: Array<ClassValue>) {
  return twMerge(clsx(inputs))
}

export function createFetchInstance(baseURL: string, defaultOptions:
RequestInit = {}) {
  return async <T>(endpoint: string, options: RequestInit = {}): Promise<T>
=> {
    const url = `${baseURL}${endpoint}`;
    const mergedOptions = { ...defaultOptions, ...options };
```

```
    const response = await fetch(url, mergedOptions);

    if (!response.ok) {
      throw new Error(`Fetch error: ${response.status}
${response.statusText}`);
    }

    const contentType = response.headers.get("Content-Type");
    if (contentType && contentType.includes("application/json")) {
      return response.json() as T;
    } else {
      return response.text() as unknown as T; // Якщо це текст, повертаємо
його як T
    }
  };
}

export const BASE_URL = "http://192.168.4.1"
export const fetcher = createFetchInstance(BASE_URL);
```

Файл src\providers\NoiseGeneratorProvider.tsx

```
import React, { useEffect, type FC } from 'react'
import {
  useNoiseGeneratorStore,
  type ApiNoiseGeneratorState,
} from '@/stores/useNoiseGeneratorStore'
import { BASE_URL, fetcher } from '@/lib/utils'
import { useQuery } from '@tanstack/react-query'

export type NoiseGeneratorProviderProps = { children: React.ReactNode }

const fetchAudioFiles = async () => {
  return fetcher<ApiNoiseGeneratorState>('/state')
}

const NoiseGeneratorProvider: FC<NoiseGeneratorProviderProps> = ({
  children,
}) => {
  const { setIsRecording, setActiveAudioId, setAudioFiles, setIsUltrasonic }
=
    useNoiseGeneratorStore()

  // Використовуємо useQuery для початкового фетчингу даних
  const { data, isSuccess } = useQuery({
    queryKey: ['noise-generator-state'],
    queryFn: fetchAudioFiles,
  })

  useEffect(() => {
    if (isSuccess) {
      setIsRecording(data.isRecording)
      setActiveAudioId(data.activeAudioId)
      setAudioFiles(data.audioFiles)
      setIsUltrasonic(data.isUltrasonic)
    }
  }, [isSuccess])

  // Підписка на SSE
  useEffect(() => {
    const eventSource = new EventSource(`${BASE_URL}/events`)

    eventSource.addEventListener('active-audio-id', (event) => {
```

```
      const id = Number(event.data)
      if (id === -1) {
        setActiveAudioId(null)
        return
      }
      setActiveAudioId(id)
    })

    eventSource.addEventListener('audio-files', (event) => {
      setAudioFiles(JSON.parse(event.data))
    })

    eventSource.addEventListener('is-recording', (event) => {
      console.log('event record', event);
      setIsRecording(event.data === "true")
    })

    eventSource.addEventListener('is-ultrasonic', (event) => {
      setIsUltrasonic(event.data === "true")
    })

    eventSource.onmessage = (event) => {
      console.log('event', event)

      try {
        const parsedData = JSON.parse(event.data)

        switch (event.type) {
          case 'is-recording':
            setIsRecording(parsedData === 'true')
            break
          case 'active-audio-id':
            const id = parseInt(parsedData)
            setActiveAudioId(id !== -1 ? id : null)
            break
          case 'is-ultrasonic':
            setIsUltrasonic(parsedData === 'true')
            break
          case 'audio-files':
            setAudioFiles(parsedData)
            break
          default:
            console.warn(`Unknown SSE event type: ${event.type}`)
        }
      } catch (error) {
        console.error('Failed to process SSE message:', error)
      }
    }

    eventSource.onerror = () => {
      console.error('SSE connection failed.')
      eventSource.close()
    }

    return () => {
      eventSource.close()
    }
  }, [setIsRecording, setActiveAudioId, setAudioFiles, setIsUltrasonic])

  return <>{children}</>
}

export default NoiseGeneratorProvider
```

Файл src\routes\demo.tanstack-query.tsx

```
import { createRoute } from '@tanstack/react-router'
import { useQuery } from '@tanstack/react-query'

import type { RootRoute } from '@tanstack/react-router'

function TanStackQueryDemo() {
  const { data } = useQuery({
    queryKey: ['people'],
    queryFn: () =>
      Promise.resolve([{ name: 'John Doe' }, { name: 'Jane Doe' }]),
    initialData: [],
  })

  return (
    <div className="p-4">
      <h1 className="text-2xl mb-4">People list from Swapi</h1>
      <ul>
        {data.map((person) => (
          <li key={person.name}>{person.name}</li>
        ))}
      </ul>
    </div>
  )
}

export default (parentRoute: RootRoute) =>
  createRoute({
    path: '/demo/tanstack-query',
    component: TanStackQueryDemo,
    getParentRoute: () => parentRoute,
  })
```

Файл src\stores\useNoiseGeneratorStore.ts

```
import { type Nullable } from './../types/global.d'
import type { AudioFile } from '@/hooks/useAudioList'
import type { BooleanFn, VoidFn } from '@/types/global'
import { create } from 'zustand'

export interface ApiNoiseGeneratorState {
  activeAudioId: Nullable<AudioFile['id']>
  audioFiles:  AudioFile[]
  isRecording: boolean
  isUltrasonic: boolean
}

export interface NoiseGeneratorState extends Omit<ApiNoiseGeneratorState,
'audioFiles'> {
  audioFiles:  Nullable<AudioFile[]>
  setAudioFiles: VoidFn<AudioFile[]>
  setActiveAudioId: VoidFn<Nullable<AudioFile['id']>>
  setIsRecording: BooleanFn
  setIsUltrasonic: BooleanFn
}

export const useNoiseGeneratorStore = create<NoiseGeneratorState>((set) => ({
  activeAudioId: null,
  audioFiles: [],
  isRecording: false,
  isUltrasonic: false,
  setActiveAudioId: (id: Nullable<AudioFile['id']>) => set({ activeAudioId:
```

```
id }),
  setAudioFiles: (files: AudioFile[]) => set({ audioFiles: files }),
  setIsRecording: (val: boolean) => set({ isRecording: val }),
  setIsUltrasonic: (val: boolean) => set({ isUltrasonic: val }),
}))
```

Файл src\types\global.d.ts

```
export type Nullable<T> = T | null;

export type VoidFn<T> = (value: T) => void;

export type BooleanFn = VoidFn<boolean>;
```

Файл src\components\ui\alert.tsx

```
import * as React from "react"
import { cva, type VariantProps } from "class-variance-authority"

import { cn } from "@/lib/utils"

const alertVariants = cva(
  "relative w-full rounded-lg border px-4 py-3 text-sm grid has-[>svg]:grid-
cols-[calc(var(--spacing)*4)_1fr] grid-cols-[0_1fr] has-[>svg]:gap-x-3 gap-y-
0.5 items-start [&>svg]:size-4 [&>svg]:translate-y-0.5 [&>svg]:text-current",
  {
    variants: {
      variant: {
        default: "bg-card text-card-foreground",
        destructive:
          "text-destructive bg-card [&>svg]:text-current *:data-[slot=alert-
description]:text-destructive/90",
      },
    },
    defaultVariants: {
      variant: "default",
    },
  }
)

function Alert({
  className,
  variant,
  ...props
}: React.ComponentProps<"div"> & VariantProps<typeof alertVariants>) {
  return (
    <div
      data-slot="alert"
      role="alert"
      className={cn(alertVariants({ variant }), className)}
      {...props}
    />
  )
}

function AlertTitle({ className, ...props }: React.ComponentProps<"div">) {
  return (
    <div
      data-slot="alert-title"
      className={cn(
        "col-start-2 line-clamp-1 min-h-4 font-medium tracking-tight",
        className
```

```
      )}
      {...props}
    />
  )
}

function AlertDescription({
  className,
  ...props
}: React.ComponentProps<"div">) {
  return (
    <div
      data-slot="alert-description"
      className={cn(
        "text-muted-foreground col-start-2 grid justify-items-start gap-1
text-sm [&_p]:leading-relaxed",
        className
      )}
      {...props}
    />
  )
}

export { Alert, AlertTitle, AlertDescription }
```

Файл src\components\ui\button.tsx

```
import * as React from "react"
import { Slot } from "@radix-ui/react-slot"
import { cva, type VariantProps } from "class-variance-authority"

import { cn } from "@/lib/utils"

const buttonVariants = cva(
  "inline-flex items-center justify-center gap-2 whitespace-nowrap rounded-md
text-sm font-medium transition-all cursor-pointer disabled:pointer-events-
none disabled:opacity-50 [&_svg]:pointer-events-none
[&_svg:not([class*='size-'])]:size-4 shrink-0 [&_svg]:shrink-0 outline-none
focus-visible:border-ring focus-visible:ring-ring/50 focus-visible:ring-[3px]
aria-invalid:ring-destructive/20 dark:aria-invalid:ring-destructive/40 aria-
invalid:border-destructive",
  {
    variants: {
      variant: {
        default:
          "bg-primary text-primary-foreground shadow-xs hover:bg-primary/90",
        destructive:
          "bg-destructive text-white shadow-xs hover:bg-destructive/80 focus-
visible:ring-destructive/20 dark:focus-visible:ring-destructive/40 dark:bg-
destructive/60",
        outline:
          "border bg-background shadow-xs hover:bg-accent hover:text-accent-
foreground dark:bg-input/30 dark:border-input dark:hover:bg-input/50",
        secondary:
          "bg-secondary text-secondary-foreground shadow-xs hover:bg-
secondary/80",
        ghost:
          "hover:bg-accent hover:text-accent-foreground dark:hover:bg-
accent/50",
        link: "text-primary underline-offset-4 hover:underline",
      },
      size: {
        default: "h-9 px-4 py-2 has-[>svg]:px-3",
```

```
        sm: "h-8 rounded-md gap-1.5 px-3 has-[>svg]:px-2.5",
        lg: "h-10 rounded-md px-6 has-[>svg]:px-4",
        icon: "size-9",
      },
    },
    defaultVariants: {
      variant: "default",
      size: "default",
    },
  }
)

function Button({
  className,
  variant,
  size,
  asChild = false,
  ...props
}: React.ComponentProps<"button"> &
  VariantProps<typeof buttonVariants> & {
    asChild?: boolean
  }) {
  const Comp = asChild ? Slot : "button"

  return (
    <Comp
      data-slot="button"
      className={cn(buttonVariants({ variant, size, className }))}
      {...props}
    />
  )
}

export { Button, buttonVariants }
```

Файл src\components\ui\card.tsx

```
import * as React from "react"

import { cn } from "@/lib/utils"

function Card({ className, ...props }: React.ComponentProps<"div">) {
  return (
    <div
      data-slot="card"
      className={cn(
        "bg-card text-card-foreground flex flex-col gap-6 rounded-xl border
py-6 shadow-sm",
        className
      )}
      {...props}
    />
  )
}

function CardHeader({ className, ...props }: React.ComponentProps<"div">) {
  return (
    <div
      data-slot="card-header"
      className={cn(
        "@container/card-header grid auto-rows-min grid-rows-[auto_auto]
items-start gap-1.5 px-6 has-data-[slot=card-action]:grid-cols-[1fr_auto]
[.border-b]:pb-6",
```

```
        className
      )}
      {...props}
    />
  )
}

function CardTitle({ className, ...props }: React.ComponentProps<"div">) {
  return (
    <div
      data-slot="card-title"
      className={cn("leading-none font-semibold", className)}
      {...props}
    />
  )
}

function CardDescription({ className, ...props }:
React.ComponentProps<"div">) {
  return (
    <div
      data-slot="card-description"
      className={cn("text-muted-foreground text-sm", className)}
      {...props}
    />
  )
}

function CardAction({ className, ...props }: React.ComponentProps<"div">) {
  return (
    <div
      data-slot="card-action"
      className={cn(
        "col-start-2 row-span-2 row-start-1 self-start justify-self-end",
        className
      )}
      {...props}
    />
  )
}

function CardContent({ className, ...props }: React.ComponentProps<"div">) {
  return (
    <div
      data-slot="card-content"
      className={cn("px-6", className)}
      {...props}
    />
  )
}

function CardFooter({ className, ...props }: React.ComponentProps<"div">) {
  return (
    <div
      data-slot="card-footer"
      className={cn("flex items-center px-6 [.border-t]:pt-6", className)}
      {...props}
    />
  )
}

export {
  Card,
  CardHeader,
```

```
  CardFooter,
  CardTitle,
  CardAction,
  CardDescription,
  CardContent,
}
```

Файл src\components\ui\context-menu.tsx

```
import * as React from "react"
import * as ContextMenuPrimitive from "@radix-ui/react-context-menu"
import { CheckIcon, ChevronRightIcon, CircleIcon } from "lucide-react"

import { cn } from "@/lib/utils"

function ContextMenu({
  ...props
}: React.ComponentProps<typeof ContextMenuPrimitive.Root>) {
  return <ContextMenuPrimitive.Root data-slot="context-menu" {...props} />
}

function ContextMenuTrigger({
  ...props
}: React.ComponentProps<typeof ContextMenuPrimitive.Trigger>) {
  return (
    <ContextMenuPrimitive.Trigger data-slot="context-menu-trigger" {...props}
/>
  )
}

function ContextMenuGroup({
  ...props
}: React.ComponentProps<typeof ContextMenuPrimitive.Group>) {
  return (
    <ContextMenuPrimitive.Group data-slot="context-menu-group" {...props} />
  )
}

function ContextMenuPortal({
  ...props
}: React.ComponentProps<typeof ContextMenuPrimitive.Portal>) {
  return (
    <ContextMenuPrimitive.Portal data-slot="context-menu-portal" {...props}
/>
  )
}

function ContextMenuSub({
  ...props
}: React.ComponentProps<typeof ContextMenuPrimitive.Sub>) {
  return <ContextMenuPrimitive.Sub data-slot="context-menu-sub" {...props} />
}

function ContextMenuRadioGroup({
  ...props
}: React.ComponentProps<typeof ContextMenuPrimitive.RadioGroup>) {
  return (
    <ContextMenuPrimitive.RadioGroup
      data-slot="context-menu-radio-group"
      {...props}
    />
  )
}
```

```
function ContextMenuSubTrigger({
  className,
  inset,
  children,
  ...props
}: React.ComponentProps<typeof ContextMenuPrimitive.SubTrigger> & {
  inset?: boolean
}) {
  return (
    <ContextMenuPrimitive.SubTrigger
      data-slot="context-menu-sub-trigger"
      data-inset={inset}
      className={cn(
        "focus:bg-accent focus:text-accent-foreground data-[state=open]:bg-
accent data-[state=open]:text-accent-foreground flex cursor-default items-
center rounded-sm px-2 py-1.5 text-sm outline-hidden select-none data-
[inset]:pl-8 [&_svg]:pointer-events-none [&_svg]:shrink-0
[&_svg:not([class*='size-'])]:size-4",
        className
      )}
      {...props}
    >
      {children}
      <ChevronRightIcon className="ml-auto" />
    </ContextMenuPrimitive.SubTrigger>
  )
}

function ContextMenuSubContent({
  className,
  ...props
}: React.ComponentProps<typeof ContextMenuPrimitive.SubContent>) {
  return (
    <ContextMenuPrimitive.SubContent
      data-slot="context-menu-sub-content"
      className={cn(
        "bg-popover text-popover-foreground data-[state=open]:animate-in
data-[state=closed]:animate-out data-[state=closed]:fade-out-0 data-
[state=open]:fade-in-0 data-[state=closed]:zoom-out-95 data-
[state=open]:zoom-in-95 data-[side=bottom]:slide-in-from-top-2 data-
[side=left]:slide-in-from-right-2 data-[side=right]:slide-in-from-left-2
data-[side=top]:slide-in-from-bottom-2 z-50 min-w-[8rem] origin-(--radix-
context-menu-content-transform-origin) overflow-hidden rounded-md border p-1
shadow-lg",
        className
      )}
      {...props}
    />
  )
}

function ContextMenuContent({
  className,
  ...props
}: React.ComponentProps<typeof ContextMenuPrimitive.Content>) {
  return (
    <ContextMenuPrimitive.Portal>
      <ContextMenuPrimitive.Content
        data-slot="context-menu-content"
        className={cn(
          "bg-popover text-popover-foreground data-[state=open]:animate-in
data-[state=closed]:animate-out data-[state=closed]:fade-out-0 data-
[state=open]:fade-in-0 data-[state=closed]:zoom-out-95 data-
```

```
[state=open]:zoom-in-95 data-[side=bottom]:slide-in-from-top-2 data-
[side=left]:slide-in-from-right-2 data-[side=right]:slide-in-from-left-2
data-[side=top]:slide-in-from-bottom-2 z-50 max-h-(--radix-context-menu-
content-available-height) min-w-[8rem] origin-(--radix-context-menu-content-
transform-origin) overflow-x-hidden overflow-y-auto rounded-md border p-1
shadow-md",
          className
        )}
        {...props}
      />
    </ContextMenuPrimitive.Portal>
  )
}

function ContextMenuItem({
  className,
  inset,
  variant = "default",
  ...props
}: React.ComponentProps<typeof ContextMenuPrimitive.Item> & {
  inset?: boolean
  variant?: "default" | "destructive"
}) {
  return (
    <ContextMenuPrimitive.Item
      data-slot="context-menu-item"
      data-inset={inset}
      data-variant={variant}
      className={cn(
        "focus:bg-accent focus:text-accent-foreground data-
[variant=destructive]:text-destructive data-[variant=destructive]:focus:bg-
destructive/10 dark:data-[variant=destructive]:focus:bg-destructive/20 data-
[variant=destructive]:focus:text-destructive data-
[variant=destructive]:*:[svg]:!text-destructive [&_svg:not([class*='text-
'])]:text-muted-foreground relative flex cursor-default items-center gap-2
rounded-sm px-2 py-1.5 text-sm outline-hidden select-none data-
[disabled]:pointer-events-none data-[disabled]:opacity-50 data-[inset]:pl-8
[&_svg]:pointer-events-none [&_svg]:shrink-0 [&_svg:not([class*='size-
'])]:size-4",
        className
      )}
      {...props}
    />
  )
}

function ContextMenuCheckboxItem({
  className,
  children,
  checked,
  ...props
}: React.ComponentProps<typeof ContextMenuPrimitive.CheckboxItem>) {
  return (
    <ContextMenuPrimitive.CheckboxItem
      data-slot="context-menu-checkbox-item"
      className={cn(
        "focus:bg-accent focus:text-accent-foreground relative flex cursor-
default items-center gap-2 rounded-sm py-1.5 pr-2 pl-8 text-sm outline-hidden
select-none data-[disabled]:pointer-events-none data-[disabled]:opacity-50
[&_svg]:pointer-events-none [&_svg]:shrink-0 [&_svg:not([class*='size-
'])]:size-4",
        className
      )}
      checked={checked}
```

```
      {...props}
    >
      <span className="pointer-events-none absolute left-2 flex size-3.5
items-center justify-center">
        <ContextMenuPrimitive.ItemIndicator>
          <CheckIcon className="size-4" />
        </ContextMenuPrimitive.ItemIndicator>
      </span>
      {children}
    </ContextMenuPrimitive.CheckboxItem>
  )
}

function ContextMenuRadioItem({
  className,
  children,
  ...props
}: React.ComponentProps<typeof ContextMenuPrimitive.RadioItem>) {
  return (
    <ContextMenuPrimitive.RadioItem
      data-slot="context-menu-radio-item"
      className={cn(
        "focus:bg-accent focus:text-accent-foreground relative flex cursor-
default items-center gap-2 rounded-sm py-1.5 pr-2 pl-8 text-sm outline-hidden
select-none data-[disabled]:pointer-events-none data-[disabled]:opacity-50
[&_svg]:pointer-events-none [&_svg]:shrink-0 [&_svg:not([class*='size-
'])]:size-4",
        className
      )}
      {...props}
    >
      <span className="pointer-events-none absolute left-2 flex size-3.5
items-center justify-center">
        <ContextMenuPrimitive.ItemIndicator>
          <CircleIcon className="size-2 fill-current" />
        </ContextMenuPrimitive.ItemIndicator>
      </span>
      {children}
    </ContextMenuPrimitive.RadioItem>
  )
}

function ContextMenuLabel({
  className,
  inset,
  ...props
}: React.ComponentProps<typeof ContextMenuPrimitive.Label> & {
  inset?: boolean
}) {
  return (
    <ContextMenuPrimitive.Label
      data-slot="context-menu-label"
      data-inset={inset}
      className={cn(
        "text-foreground px-2 py-1.5 text-sm font-medium data-[inset]:pl-8",
        className
      )}
      {...props}
    />
  )
}

function ContextMenuSeparator({
  className,
```

```
  ...props
}: React.ComponentProps<typeof ContextMenuPrimitive.Separator>) {
  return (
    <ContextMenuPrimitive.Separator
      data-slot="context-menu-separator"
      className={cn("bg-border -mx-1 my-1 h-px", className)}
      {...props}
    />
  )
}

function ContextMenuShortcut({
  className,
  ...props
}: React.ComponentProps<"span">) {
  return (
    <span
      data-slot="context-menu-shortcut"
      className={cn(
        "text-muted-foreground ml-auto text-xs tracking-widest",
        className
      )}
      {...props}
    />
  )
}

export {
  ContextMenu,
  ContextMenuTrigger,
  ContextMenuContent,
  ContextMenuItem,
  ContextMenuCheckboxItem,
  ContextMenuRadioItem,
  ContextMenuLabel,
  ContextMenuSeparator,
  ContextMenuShortcut,
  ContextMenuGroup,
  ContextMenuPortal,
  ContextMenuSub,
  ContextMenuSubContent,
  ContextMenuSubTrigger,
  ContextMenuRadioGroup,
}
```


Файл src\components\ui\errorMessage.tsx

```
import { AlertCircle } from 'lucide-react'
import { Alert, AlertDescription, AlertTitle } from './alert'

export type ErrorMessageProps = {
  title?: string
  description: string
}

export const ErrorMessage = ({
  title = 'Error',
  description,
}: ErrorMessageProps) => (
  <Alert variant="destructive">
    <AlertCircle className="h-4 w-4" />
    <AlertTitle>{title}</AlertTitle>
    <AlertDescription>{description}</AlertDescription>
```

```
  </Alert>
)
```

Файл src\components\ui\skeleton.tsx

```
import { cn } from "@/lib/utils"

function Skeleton({ className, ...props }: React.ComponentProps<"div">) {
  return (
    <div
      data-slot="skeleton"
      className={cn("bg-accent animate-pulse rounded-md", className)}
      {...props}
    />
  )
}

export { Skeleton }
```

Файл src\components\ui\spinner.tsx

```
import React from 'react';
import { cn } from '@/lib/utils';
import { cva, type VariantProps } from "class-variance-authority"
import { Loader2 } from 'lucide-react';

const spinnerVariants = cva('flex-col items-center justify-center', {
  variants: {
    show: {
      true: 'flex',
      false: 'hidden',
    },
  },
  defaultVariants: {
    show: true,
  },
});

const loaderVariants = cva('animate-spin text-primary', {
  variants: {
    size: {
      small: 'size-5',
      medium: 'size-8',
      large: 'size-12',
    },
  },
  defaultVariants: {
    size: 'medium',
  },
});

interface SpinnerContentProps
  extends VariantProps<typeof spinnerVariants>,
    VariantProps<typeof loaderVariants> {
  className?: string;
  children?: React.ReactNode;
}

export function Spinner({ size, show, children, className }:
SpinnerContentProps) {
  return (
    <span className={spinnerVariants({ show })}>
      <Loader2 className={cn(loaderVariants({ size }), className)} />
```

```
      {children}
    </span>
  );
}
```

Файл src\components\ui\switch.tsx

```
import * as React from "react"
import * as SwitchPrimitive from "@radix-ui/react-switch"

import { cn } from "@/lib/utils"

function Switch({
  className,
  ...props
}: React.ComponentProps<typeof SwitchPrimitive.Root>) {
  return (
    <SwitchPrimitive.Root
      data-slot="switch"
      className={cn(
        "cursor-pointer peer data-[state=checked]:bg-primary data-
[state=unchecked]:bg-input focus-visible:border-ring focus-visible:ring-
ring/50 dark:data-[state=unchecked]:bg-input/80 inline-flex h-[1.15rem] w-8
shrink-0 items-center rounded-full border border-transparent shadow-xs
transition-all outline-none focus-visible:ring-[3px] disabled:cursor-not-
allowed disabled:opacity-50 hover:opacity-80", // Додано hover:opacity-80
        className
      )}
      {...props}
    >
      <SwitchPrimitive.Thumb
        data-slot="switch-thumb"
        className={cn(
          "bg-background dark:data-[state=unchecked]:bg-foreground dark:data-
[state=checked]:bg-primary-foreground pointer-events-none block size-4
rounded-full ring-0 transition-transform data-[state=checked]:translate-x-
[calc(100%-2px)] data-[state=unchecked]:translate-x-0"
        )}
      />
    </SwitchPrimitive.Root>
  )
}

export { Switch }
```

Файл src\integrations\tanstack-query\layout.tsx

```
import { ReactQueryDevtools } from '@tanstack/react-query-devtools'

export default function LayoutAddition() {
  return <ReactQueryDevtools buttonPosition="bottom-right" />
}
```

Файл src\integrations\tanstack-query\root-provider.tsx

```
import { QueryClient, QueryClientProvider } from '@tanstack/react-query'

const queryClient = new QueryClient({
  defaultOptions: {
    queries: {
      retry: 1,
      retryDelay: 1000,
```

```
    },
    mutations: {
      retry: 1,
    },
  },
})

export function getContext() {
  return {
    queryClient,
  }
}

export function Provider({ children }: { children: React.ReactNode }) {
  return (
    <QueryClientProvider
client={queryClient}>{children}</QueryClientProvider>
  )
}
```

Файл src\components\features\audio\Audio.tsx

```
import { Button } from '@/components/ui/button'
import { Card, CardTitle } from '@/components/ui/card'
import { Spinner } from '@/components/ui/spinner'
import {
  ContextMenu,
  ContextMenuTrigger,
  ContextMenuContent,
  ContextMenuItem,
} from '@/components/ui/context-menu' // Підключіть ContextMenu із shadcn
import type { AudioFile } from '@/hooks/useAudioList'
import { cn } from '@/lib/utils'
import { PlayIcon, PauseIcon, Trash2Icon } from 'lucide-react'

interface AudioProps {
  file: AudioFile
  onClick?: (id: AudioFile['id']) => void
  onDelete?: (id: AudioFile['id']) => void
  isActive: boolean
  isLoading: boolean
}

const Audio = ({
  file,
  onClick,
  onDelete,
  isActive = false,
  isLoading = false,
}: AudioProps) => {
  return (
    <ContextMenu>
      <ContextMenuTrigger>
        <Card
          key={file.id}
          className={cn(
            'bg-white cursor-pointer flex-row justify-between items-center
gap-4 px-4 py-2 shadow-md rounded-lg transition-all duration-300
hover:shadow-lg hover:bg-blue-50',
            {
              'bg-blue-100 ring-2 ring-blue-500': isActive,
            },
          )}
```

```
          onClick={() => onClick?.(file.id)}
        >
          <CardTitle className="text-base font-
semibold">{file.name}</CardTitle>
          <Button size="icon" variant="outline" className="rounded-full">
            {isLoading ? (
              <Spinner size={'small'} className="text-blue-500" />
            ) : isActive ? (
              <PauseIcon className="h-5 w-5 text-blue-500" />
            ) : (
              <PlayIcon className="h-5 w-5" />
            )}
          </Button>
        </Card>
      </ContextMenuTrigger>
      <ContextMenuContent>
        <ContextMenuItem
          onClick={() => onDelete?.(file.id)}
          className="flex items-center gap-2"
        >
          <Trash2Icon className="h-4 w-4 text-red-500" />
          Delete
        </ContextMenuItem>
      </ContextMenuContent>
    </ContextMenu>
  )
}

export default Audio
```

Файл src\components\features\audio\AudioList.tsx

```
import { useAudioList } from '@/hooks/useAudioList'
import Audio from './Audio'
import AudioLoader from './AudioLoader'

export function AudioList() {
  const {
    data,
    activeAudioId,
    isLoading,
    isError,
    play,
    deleteAudio,
    playState,
  } = useAudioList()

  return (
    <AudioLoader isLoading={isLoading} isError={isError}>
      <div className="max-w-md mx-auto flex flex-col gap-2 min-w-[250px]">
        {data?.map((file) => (
          <Audio
            key={file.id}
            isActive={file.id === activeAudioId}
            isLoading={file.id === activeAudioId && playState.isPending}
            file={file}
            onClick={play}
            onDelete={deleteAudio}
          />
        ))}
      </div>
    </AudioLoader>
  )
```

```
}
```

Файл src\components\features\audio\AudioLoader.tsx

```
import { ErrorMessage } from '@/components/ui/errorMessage'
import { Skeleton } from '@/components/ui/skeleton'
import type { ReactNode } from 'react'

interface AudioLoaderProps {
  isLoading?: boolean
  isError?: boolean
  children: ReactNode
}

const AudioLoader = ({ isLoading, isError, children }: AudioLoaderProps) => {
  if (isError) {
    return (
      <ErrorMessage description="Failed to load audio files. Please try again
later." />
    )
  }

  if (isLoading) {
    return (
      <div className="max-w-md mx-auto space-y-4 min-w-[250px]">
        {[...Array(5)].map((_, index) => (
          <Skeleton key={index} className="h-12 w-full rounded-lg" />
        ))}
      </div>
    )
  }

  return <>{children}</>
}

export default AudioLoader
```

Файл src\components\features\recorder\Recorder.tsx

```
import { Card, CardHeader, CardContent } from '@/components/ui/card'
import useRecording from '@/hooks/useRecording'
import RecorderHeader from './RecorderHeader'
import RecorderButtons from './RecorderButtons'
import RecorderError from './RecorderError'

const Recorder = () => {
  const { isRecording, isToggle, toggleError, toggleRecordingHandler } =
    useRecording()

  const handleToggle = async () => {
    const date = new Date()
    const options: Intl.DateTimeFormatOptions = {
      weekday: 'long',
      hour: '2-digit',
      minute: '2-digit',
      hour12: false,
    }
    const formattedDate = date.toLocaleString('en-US', options)
    const fileName = `${formattedDate}, ${date.getTime()}`
    await toggleRecordingHandler(fileName)
  }

  return (
```

```
    <Card className="gap-0">
      <CardHeader className="gap-4">
        <div className="text-lg font-semibold text-center">
          <RecorderHeader isRecording={isRecording} isStarting={isToggle} />
        </div>
      </CardHeader>
      <CardContent className="space-y-2">
        <div className="flex justify-center">
          <RecorderButtons
            isRecording={isRecording}
            isToggle={isToggle}
            onToggle={handleToggle}
          />
        </div>
        <RecorderError startError={toggleError} stopError={toggleError} />
      </CardContent>
    </Card>
  )
}

export default Recorder
```

Файл src\components\features\recorder\RecorderButtons.tsx

```
import { Button } from '@/components/ui/button'
import { MicIcon, StopCircle, Loader2 } from 'lucide-react'

interface RecorderButtonsProps {
  isRecording: boolean
  isToggle: boolean
  onToggle: () => Promise<void>
}

const RecorderButtons = ({
  isRecording,
  isToggle,
  onToggle,
}: RecorderButtonsProps) => {
  if (!isRecording) {
    return (
      <Button
        size="lg"
        variant="destructive"
        onClick={onToggle}
        disabled={isToggle}
        className="flex items-center gap-2"
      >
        {isToggle ? (
          <>
            <Loader2 className="h-5 w-5 animate-spin" />
            Starting...
          </>
        ) : (
          <>
            <MicIcon className="h-6 w-6" />
            Start
          </>
        )}
      </Button>
    )
  }
  return (
    <Button
```

```
      size="lg"
      variant="destructive"
      onClick={onToggle}
      disabled={isToggle}
      className="flex items-center gap-2"
    >
      {isToggle ? (
        <>
          <Loader2 className="h-5 w-5 animate-spin" />
          Stopping...
        </>
      ) : (
        <>
          <StopCircle className="h-6 w-6" />
          Stop
        </>
      )}
    </Button>
  )
}

export default RecorderButtons
```

Файл src\components\features\recorder\RecorderError.tsx

```
import { ErrorMessage } from "@/components/ui/errorMessage"


interface RecorderErrorProps {
  startError: boolean
  stopError: boolean
}

const RecorderError = ({ startError, stopError }: RecorderErrorProps) => {
  if (!startError && !stopError) return null

  return (
    <div className="space-y-2">
      {startError && (
        <ErrorMessage description="Failed to start recording. Please try
again." />
      )}
      {stopError && (
        <ErrorMessage description="Failed to stop recording. Please try
again." />
      )}
    </div>
  )
}

export default RecorderError
```

Файл src\components\features\recorder\RecorderHeader.tsx

```
interface RecorderHeaderProps {
  isRecording: boolean
  isStarting: boolean
}

const RecorderHeader = ({ isRecording, isStarting }: RecorderHeaderProps) =>
{
  if (isRecording) {
    return <span className="text-red-500">Recording...</span>
```

```
  }

  if (isStarting) {
    return <span className="text-blue-500">Starting Recording...</span>
  }

  return <span className="text-gray-700">Recorder Ready</span>
}

export default RecorderHeader
```

Файл src\components\features\ultrasonic\UltrasonicControl.tsx

```
import { Switch } from '@/components/ui/switch'
import {
  Card,
  CardHeader,
  CardTitle,
  CardDescription,
} from '@/components/ui/card'
import { useUltrasonicControl } from '@/hooks/useUltrasonicControl'
import { Spinner } from '@/components/ui/spinner'

const UltrasonicControl = () => {
  const { isUltrasonic, toggle, isLoading, isError } = useUltrasonicControl()

  return (
    <Card className="p-4">
      <CardHeader className="p-0 flex flex-row items-center justify-between">
        <div className="flex flex-col gap-2">
          <CardTitle>Ultrasonic</CardTitle>
          <CardDescription>
            Control the ultrasonic signal. Turn it on or off as needed.
          </CardDescription>
          {isError && (
            <p className="text-sm text-red-500">Failed to toggle
ultrasonic.</p>
          )}
        </div>
        <div className="flex items-center gap-2">
          {isLoading && (
            <Spinner size={'small'} className="animate-spin text-gray-500" />
          )}
          <Switch
            checked={isUltrasonic}
            onCheckedChange={toggle}
            disabled={isLoading}
          />
        </div>
      </CardHeader>
    </Card>
  )
}

export default UltrasonicControl
```

Файл .gitignore

```
.pio
.vscode/.browse.c_cpp.db*
.vscode/c_cpp_properties.json
.vscode/launch.json
.vscode/ipch
```

Файл platformio.ini

```
; PlatformIO Project Configuration File
;
;   Build options: build flags, source filter
;   Upload options: custom upload port, speed and extra flags
;   Library options: dependencies, extra library storages
;   Advanced options: extra scripting
;
; Please visit documentation for the other options and examples
; https://docs.platformio.org/page/projectconf.html

[env:esp32dev]
platform = espressif32
board = esp32dev
framework = arduino
monitor_speed = 115200
board_build.filesystem = littlefs
build_flags =
      -DCORE_DEBUG_LEVEL=3
      -DBOARD_HAS_PSRAM=1
      -mfix-esp32-psram-cache-issue
lib_deps =
      SD
      thingpulse/ESP8266 and ESP32 OLED driver for SSD1306 displays@^4.6.1
      esp32async/ESPAsyncWebServer@^3.6.0
      esp32async/AsyncTCP@^3.3.2
      lorol/LittleFS_esp32@1.0.5
      esphome/ESP32-audioI2S@^2.0.7
      tzapu/WiFiManager@^2.0.17
```

Файл src\main.cpp

```
#include "sd/sd_card.h"
#include "audio/audio.h"
#include "display/display.h"
#include "config/config.h"
#include "events/events.h"
#include "server/server.h"



void setup() {
  Serial.begin(115200);
  setupSDCard();
  setupAudio();
  setupDisplay();
  setupEvents();
  serverSetup();

  xTaskCreatePinnedToCore(
    taskButtonEvent,
    "taskButtonEvent",
    10000,
    NULL,
    1,
    NULL,
    0);

}

void loop() {
```

```
  vTaskDelay(1000);
  serverLoop();
}
```

Файл src\audio\audio.cpp

```
#include "audio.h"
#include "config/config.h"
#include "wav/wav_utils.h"
#include <SD.h>
#include <driver/i2s.h>
#include <Arduino.h>
#include <stdio.h>
#include <freertos/FreeRTOS.h>
#include <I2SMEMSSampler.h>
#include <ADCSampler.h>
#include <I2SOutput.h>
#include <WAVFileReader.h>
#include <WAVFileWriter.h>

// Глобальні змінні
static I2SSampler *input = nullptr;
static Output *output = nullptr;
static TaskHandle_t playbackTaskHandle = nullptr;
static TaskHandle_t recordingTaskHandle = nullptr;
bool isPlaying = false;
bool isRecording = false;
bool isUltrasonic = false;

//
-------------------------------------------------------------------------------
// Ініціалізація I2S
void setupAudio() {
    Serial.begin(115200);
    input = new ADCSampler(ADC_UNIT_1, ADC1_CHANNEL_7, i2s_adc_config);
    output = new I2SOutput(I2S_NUM_0, i2s_speaker_pins);
    gpio_set_direction(GPIO_BUTTON, GPIO_MODE_INPUT);
    gpio_set_pull_mode(GPIO_BUTTON, GPIO_PULLDOWN_ONLY);
    ledcSetup(0, ULTRASOUND_FREQUENCY, 8); // Канал 0, частота 40 кГц,
розрядність 8 біт
    ledcAttachPin(ULTRASOUND_PIN, 0);
}

void startUltrasound() {
    // Встановлення заповнення 50% для ШІМ
    ledcWrite(0, 128); // 128 із 255 (50%)
}

void stopUltrasound() {
    // Зупинка ШІМ
    ledcWrite(0, 0);
}

//
-------------------------------------------------------------------------------
// Функція для запису
void recordTask(void *param) {
    const char *filename = (const char *)param;
    int16_t *samples = (int16_t *)malloc(sizeof(int16_t) * 1024);
    FILE *fp = fopen(filename, "wb");
    if (!fp) {
        Serial.println("Failed to open file for recording.");
        free(samples);
        vTaskDelete(nullptr);
```

```
        return;
    }

    WAVFileWriter writer(fp, input->sample_rate());
    input->start();
    uint32_t startTime = millis();

    while (gpio_get_level(GPIO_BUTTON) == 1 && (millis() - startTime) <
RECORD_DURATION_SECONDS * 1000) {
        int samplesRead = input->read(samples, 1024);
        writer.write(samples, samplesRead);
    }

    input->stop();
    writer.finish();
    fclose(fp);
    free(samples);
    isRecording = false;
    Serial.println("Recording finished.");
    vTaskDelete(nullptr);
}

void startRecording(const char *filename) {
    if (isRecording) {
        Serial.println("Recording is already in progress.");
        return;
    }
    isRecording = true;
    xTaskCreatePinnedToCore(recordTask, "RecordTask", 4096, (void *)filename,
1, &recordingTaskHandle, 1);
}

void stopRecording() {
    if (isRecording) {
        gpio_set_level(GPIO_BUTTON, 0); // Симуляція відпускання кнопки
        isRecording = false;
    }
}

//
_____________________________________________________________________________
// Функція для програвання
void playbackTask(void *param) {
    const char *filename = (const char *)param;
    int16_t *samples = (int16_t *)malloc(sizeof(int16_t) * 1024);
    Serial.print(filename);
    FILE *fp = fopen("/sdcard/900yearsold_8.wav", "rb");
    if (!fp) {
        Serial.println("Failed to open file for playback.");
        free(samples);
        vTaskDelete(nullptr);
        return;
    }

    WAVFileReader reader(fp);
    output->start(reader.sample_rate());

    while (true) {
        int samplesRead = reader.read(samples, 1024);
        if (samplesRead == 0) break;
        output->write(samples, samplesRead);
    }

    output->stop();
```

```
    fclose(fp);
    free(samples);
    isPlaying = false;
    Serial.println("Playback finished.");
    vTaskDelete(nullptr);
}

void startPlayback(const char *filename) {
    Serial.println(filename);
    if (isPlaying) {
        Serial.println("Playback is already in progress.");
        return;
    }
    isPlaying = true;
    xTaskCreatePinnedToCore(playbackTask, "PlaybackTask", 4096, (void
*)filename, 1, &playbackTaskHandle, 1);
}

void stopPlayback() {
    if (isPlaying) {
        isPlaying = false;
        vTaskDelete(playbackTaskHandle);
        playbackTaskHandle = nullptr;
        output->stop();
    }
}

//
─────────────────────────────────────────────────────────────────────────────
// Логіка кнопки
void waitForButtonPress() {
    while (gpio_get_level(GPIO_BUTTON) == 0) {
        vTaskDelay(pdMS_TO_TICKS(100));
    }
}

//
─────────────────────────────────────────────────────────────────────────────
// Основний цикл
void mainTask(void *param) {
    while (true) {
        waitForButtonPress();
        startRecording("/test_8.wav");
        waitForButtonPress();
        stopRecording();
        startPlayback("/test_8.wav");
        waitForButtonPress();
        stopPlayback();
    }
}

void startMainTask() {
    xTaskCreatePinnedToCore(mainTask, "MainTask", 4096, nullptr, 1, nullptr,
1);
}
```

Файл src\audio\audio.h

```
#pragma once

#include <Arduino.h>

void setupAudio();
```

```
void startPlayback(const char *filename);
void stopPlayback();
void startRecording(const char *filename);
void stopRecording();

extern bool isPlaying;
extern bool isRecording;
extern bool isUltrasonic;
```

Файл src\config\config.cpp

```
#include "config.h"

//
// ─── I2S CONFIGURATION STRUCTS ──────────────────────────────────
//


i2s_config_t i2s_adc_config = {
    .mode = (i2s_mode_t)(I2S_MODE_MASTER | I2S_MODE_RX |
I2S_MODE_ADC_BUILT_IN),
    .sample_rate = SAMPLE_RATE,
    .bits_per_sample = I2S_BITS_PER_SAMPLE_16BIT,
    .channel_format = I2S_CHANNEL_FMT_ONLY_LEFT,
    .communication_format = I2S_COMM_FORMAT_STAND_I2S,
    .intr_alloc_flags = ESP_INTR_FLAG_LEVEL1,
    .dma_buf_count = 4,
    .dma_buf_len = 1024,
    .use_apll = false,
    .tx_desc_auto_clear = false,
    .fixed_mclk = 0
};

// i2s config for reading from I2S
i2s_config_t i2s_mic_Config = {
    .mode = (i2s_mode_t)(I2S_MODE_MASTER | I2S_MODE_RX),
    .sample_rate = SAMPLE_RATE,
    .bits_per_sample = I2S_BITS_PER_SAMPLE_32BIT,
    .channel_format = I2S_CHANNEL_FMT_ONLY_LEFT,
    .communication_format = I2S_COMM_FORMAT_STAND_I2S,
    .intr_alloc_flags = ESP_INTR_FLAG_LEVEL1,
    .dma_buf_count = 4,
    .dma_buf_len = 1024,
    .use_apll = false,
    .tx_desc_auto_clear = false,
    .fixed_mclk = 0
};

// i2s speaker pins
i2s_pin_config_t i2s_speaker_pins = {
    .bck_io_num = I2S_SPEAKER_BCK_IO_NUM,    // Bit clock
    .ws_io_num = I2S_SPEAKER_WS_IO_NUM, // Word select (LRCK)
    .data_out_num = I2S_SPEAKER_DATA_OUT_NUM,   // Data output to speaker
    .data_in_num = I2S_PIN_NO_CHANGE           // Not used (output only)
};
```

Файл src\config\config.h

```
#include <driver/i2s.h>

//
```

```
// ─── BUTTONS ─────────────────────────────────────────────────────────────────
//

// Button GPIO pins
#define BUTTON_YELLOW 13  // Yellow button
#define BUTTON_GREEN  12  // Green button
#define BUTTON_BLUE   14  // Blue button
#define BUTTON_RED    27  // Red button

// Button masks for bitwise operations
#define YELLOW_MASK 0b0001  // Mask for yellow button
#define GREEN_MASK  0b0010  // Mask for green button
#define BLUE_MASK   0b0100  // Mask for blue button
#define RED_MASK    0b1000  // Mask for red button
#define BUTTON_MASK (YELLOW_MASK | GREEN_MASK | BLUE_MASK | RED_MASK)  //
Combined mask for all buttons

// Button states
#define BUTTON_PRESSED LOW   // Logic level when button is pressed
#define BUTTON_RELEASED HIGH // Logic level when button is released

// Button timing settings
#define BUTTON_DEBOUNCE_DELAY 50     // Debounce delay in milliseconds
#define BUTTON_LONG_PRESS_DELAY 1000 // Long press threshold in milliseconds

//
// ─── LEDS ────────────────────────────────────────────────────────────────────
//

// LED GPIO pins
#define LED_YELLOW 15 // Yellow LED
#define LED_GREEN   2 // Green LED
#define LED_BLUE    4 // Blue LED
#define LED_RED    16 // Red LED

//
// ─── AUDIO SAMPLING ──────────────────────────────────────────────────────────
//

// Global sample rate
#define SAMPLE_RATE 16000 // 16 kHz sample rate

//
// ─── I2S SPEAKER CONFIGURATION ───────────────────────────────────────────────
//

// I2S speaker pin configuration
#define I2S_SPEAKER_BCK_IO_NUM     GPIO_NUM_32 // Bit clock (BCLK)
#define I2S_SPEAKER_WS_IO_NUM GPIO_NUM_26 // Word select (LRCK)
#define I2S_SPEAKER_DATA_OUT_NUM      GPIO_NUM_27 // Data out

// Ultrasound speaker
#define ULTRASOUND_PIN GPIO_NUM_25
#define ULTRASOUND_CHANNEL 0
#define ULTRASOUND_FREQUENCY 21000
#define ULTRASOUND_RESOLUTION 1

//
// ─── SD CARD CONFIGURATION ───────────────────────────────────────────────────
```

```
//

// SPI pins for SD card module
#define PIN_NUM_MISO GPIO_NUM_19
#define PIN_NUM_SCK  GPIO_NUM_18
#define PIN_NUM_MOSI GPIO_NUM_23
#define PIN_NUM_CS   GPIO_NUM_5

//
// ——— I2S MICROPHONE CONFIGURATION
————————————————————————————————————————
//

// Microphone
#define ANALOG_MIC_PIN ADC1_CHANNEL_0
#define RECORD_DURATION_SECONDS 15 // Duration for recording in seconds
#define BITS_PER_SAMPLE 16          1

#define NUM_BYTES_TO_READ_FROM_FILE 1024
#define I2S_PORT I2S_NUM_0

//
// ——— I2S CONFIG OBJECTS
————————————————————————————————————————
//

// I2S config for internal ADC input
// i2s config for using the internal ADC
extern i2s_config_t i2s_adc_config;
// i2s config for reading from of I2S
extern i2s_config_t i2s_mic_Config;
// i2s microphone pins
extern i2s_pin_config_t i2s_mic_pins;
// i2s speaker pins
extern i2s_pin_config_t i2s_speaker_pins;

// are you using an I2S microphone - comment this out if you want to use an
analog mic and ADC input
#define USE_I2S_MIC_INPUT
// are you using an I2S amplifier - comment this out if you want to use the
built in DAC
#define USE_I2S_SPEAKER_OUTPUT
#define GPIO_BUTTON GPIO_NUM_27
```


Файл src\display\display.cpp

```
#include <SSD1306Wire.h>

SSD1306Wire display(0x3c, SDA, SCL);

void setupDisplay() {
  display.init(); // Initialize the OLED display
  display.clear(); // Clear the display buffer
  display.flipScreenVertically();
  display.setFont(ArialMT_Plain_24);
  display.setTextAlignment(TEXT_ALIGN_LEFT);
  // display.setPixelColor(0, 5, BLACK); // Set a pixel at (0, 0)
  display.setColor(WHITE);
  display.drawString(0, 0, "Hello!");
  display.display(); // Actually display all of the above
}

void drawMessage(String message) {
  display.clear();
```

```
  display.drawString(0, 0, message);
  display.display();
}
```

Файл src\display\display.h

```
#pragma once
#include <WString.h>

void setupDisplay();
void drawMessage(String message);
```

Файл src\events\events.cpp

```
#include <Arduino.h>
#include "config/config.h"
#include "display/display.h"
#include "sd/sd_card.h"
#include "audio/audio.h"
#include "server/server.h"
#include <vector>
#include <string>

int currentFileIndex = 0;

//
// ─── UTILS ──────────────────────────────────────────────────────────────
//
using EventCallback = void (*)();

void drawCurrentFile()
{
  std::vector<String> files = getListFiles();
  if (files.empty())
    return; // Перевірка, чи список файлів порожній
  if (currentFileIndex >= files.size())
    return; // Перевірка, чи currentFileIndex виходить за межі списку
  drawMessage(String(currentFileIndex + 1) + ": " + files[currentFileIndex]);
}

void toggleRecording()
{
  if (isRecording)
  {
    // stopRecording();
    drawMessage("Saving...");
    sleep(1);
    drawCurrentFile();
  }
  else
  {
    String newFileName = "/recording_" + String(millis()) + ".wav";
    // startRecording(newFileName.c_str());
    drawMessage("Recording...");
  }
  isRecording = !isRecording;
}

void togglePlayback()
{
  if (isPlaying)
  {
    stopPlayback();
    drawMessage("Stopped...");
```

```
    digitalWrite(LED_BLUE, LOW);
    sleep(1);
    drawCurrentFile();
  }
  else
  {
    std::vector<String> files = getListFiles();
    if (currentFileIndex >= files.size())
    {
      drawMessage("Invalid file index");
      return;
    }
    String currentFile = files[currentFileIndex];
    digitalWrite(LED_BLUE, HIGH);
    startPlayback(('/' + currentFile).c_str());
    drawMessage("> " + currentFile);
  }
  isPlaying = !isPlaying;
}

//
// ─── EVENTS ─────────────────────────────────────────────────────────────
//

void recordingEvent(String filename)
{
  toggleRecording();
  digitalWrite(LED_RED, isPlaying ? HIGH : LOW);
}

void audioEvent(int id)
{
  currentFileIndex = id;
  togglePlayback();
  digitalWrite(LED_BLUE, isPlaying ? HIGH : LOW);
}

void ultrasonicEvent(bool toggle)
{
  isUltrasonic = toggle;
}

//
// ─── BUTTON HANDLER
─────────────────────────────────────────────────────────
//

void handleNavigateToNextFile()
{
  std::vector<String> files = getListFiles();
  currentFileIndex = (currentFileIndex + 1) % files.size();
  drawCurrentFile();
}

void handleNavigateToPreviousFile()
{
  std::vector<String> files = getListFiles();
  currentFileIndex = (currentFileIndex + files.size() - 1) % files.size();
  drawCurrentFile();
}

void handleTogglePlayback()
{
  togglePlayback();
```

```
  updateActiveAudioId(isPlaying ? -1 : currentFileIndex);
}

void handleRecordingToggle()
{
  toggleRecording();

  if(!isRecording) {
    updateAudioFiles(getListFiles());
  }

  updateIsRecording(isRecording);
}
//
// ─── BUTTON CONFIG ──────────────────────────────────────────────────────────
//

struct ButtonConfig
{
  uint8_t buttonPin;
  uint8_t mask;
  const char *message;
  uint8_t ledPin;
  EventCallback onClick;
};

ButtonConfig BUTTONS_CONFIG[4] = {
    {BUTTON_YELLOW, YELLOW_MASK, "", LED_YELLOW,
handleNavigateToPreviousFile},
    {BUTTON_GREEN, GREEN_MASK, "", LED_GREEN, handleNavigateToNextFile},
    {BUTTON_BLUE, BLUE_MASK, "", LED_BLUE, handleTogglePlayback},
    {BUTTON_RED, RED_MASK, "", LED_RED, handleRecordingToggle}};
constexpr size_t BUTTON_COUNT = sizeof(BUTTONS_CONFIG) /
sizeof(BUTTONS_CONFIG[0]);

//
// ─── EVENTS ─────────────────────────────────────────────────────────────────
//

constexpr size_t MAX_EVENTS = BUTTON_COUNT;

void *event_list[MAX_EVENTS] = {nullptr}; // Список подій
uint8_t event_masks[MAX_EVENTS] = {0};    // Відповідні маски кнопок

void register_event(EventCallback event, uint8_t button_mask)
{
  for (size_t i = 0; i < MAX_EVENTS; ++i)
  {
    if (event_list[i] == nullptr)
    {
      event_list[i] = reinterpret_cast<void *>(event);
      event_masks[i] = button_mask;
      break;
    }
  }
}

void run_events(uint8_t button_mask)
{
  for (size_t i = 0; i < MAX_EVENTS; ++i)
  {
    if ((event_list[i] != nullptr) && (event_masks[i] & button_mask))
```

```
    {
      auto callback = reinterpret_cast<EventCallback>(event_list[i]);
      callback();
    }
  }
}

void turnOffAllLeds()
{
  for (const auto &config : BUTTONS_CONFIG)
  {
    digitalWrite(config.ledPin, LOW);
  }
  stopPlayback();
}

//
// ─── BUTTON TASK ────────────────────────────────────────────────────────────
//

void taskButtonEvent(void *pvParameters)
{
  for (const auto &config : BUTTONS_CONFIG)
  {
    pinMode(config.buttonPin, INPUT_PULLUP);
    pinMode(config.ledPin, OUTPUT);
    digitalWrite(config.ledPin, LOW);
  }

  uint8_t lastButtonStateMASK = BUTTON_MASK;
  uint8_t buttonStateMASK = BUTTON_MASK;

  for (;;)
  {
    buttonStateMASK = 0;
    for (size_t i = 0; i < BUTTON_COUNT; ++i)
    {
      buttonStateMASK |= (digitalRead(BUTTONS_CONFIG[i].buttonPin) << i);
    }

    if (lastButtonStateMASK != buttonStateMASK)
    {
      for (size_t i = 0; i < BUTTON_COUNT; ++i)
      {
        const auto &config = BUTTONS_CONFIG[i];
        if (
            (buttonStateMASK & config.mask) != (lastButtonStateMASK &
config.mask) &&
            digitalRead(config.buttonPin) == BUTTON_PRESSED)
        {
          if (config.message != "")
          {
            drawMessage(config.message);
          }
          turnOffAllLeds();
          digitalWrite(config.ledPin, HIGH);
          if (config.onClick)
            config.onClick(); // Викликається тут, але також можна з
run_events()
        }
      }

      lastButtonStateMASK = buttonStateMASK;
```

```
    }

    vTaskDelay(10);
  }
}

//
// ─── INIT EVENTS ────────────────────────────────────────────────────────────
//

void setupEvents()
{
  for (const auto &config : BUTTONS_CONFIG)
  {
    register_event(config.onClick, config.mask);
  }

  drawCurrentFile();
}
```

Файл src\events\events.h

```
#pragma once
#include <cstddef>
#include <cstdint>

void setupEvents();
void run_event(uint8_t button_mask);
void register_event(void (*event)(), uint8_t button_mask);
void turnOffAllLeds();
void taskButtonEvent(void *pvParameters);

void recordingEvent(String filename);
void audioEvent(int id);
void ultrasonicEvent(bool value);

struct ButtonConfig {
  uint8_t buttonPin;
  uint8_t mask;
  const char* message;
  uint8_t ledPin;
  void (*onClick)();
};

extern int currentFileIndex;
```

Файл src\sd\sd_card.cpp

```
#include "config/config.h"
#include <SD.h>
#include <vector>
#include <SDCard.h>
#include <Arduino.h>

std::vector<String> files;

std::vector<String> listFiles() {
    std::vector<String> fileList;

    File root = SD.open("/");
    if (!root || !root.isDirectory()) {
```

```
        Serial.println("Failed to open root directory or not a directory.");
        return fileList;
    }

    File file = root.openNextFile();
    while (file) {
        if (!file.isDirectory()) {
            fileList.push_back(String(file.name()));
        }
        file.close();
        file = root.openNextFile();
    }
    if(!file) file.close();
    root.close();
    return fileList;
}

void setupSDCard() {
    pinMode(PIN_NUM_CS, OUTPUT);
    digitalWrite(PIN_NUM_CS, HIGH);

    if (!SD.begin(PIN_NUM_CS)) {
        Serial.println("Failed to initialize SD card.");
        while (true);
    }
    Serial.println("SD card initialized.");
    files = listFiles();
    // new SDCard("/sdcard", PIN_NUM_MISO, PIN_NUM_MOSI, PIN_NUM_SCK,
PIN_NUM_CS);
}


std::vector<String> getListFiles() {
    return files;
}
```

Файл src\sd\sd_card.h

```
#pragma once

#include <vector>
#include <Arduino.h>

void setupSDCard();
std::vector<String> getListFiles();
```

Файл src\server\server.cpp

```
#include <LITTLEFS.h>
#include <AsyncTCP.h>
#include <WiFiManager.h>
#include <ESPAsyncWebServer.h>
#include <WiFi.h>
#include "../config/config.h"
#include "../audio/audio.h"
#include "../sd/sd_card.h"
#include "../display/display.h"
#include "../events/events.h"
#include <driver/i2s.h>

const char *ssid = "ESP32-AP";
const char *password = "12345678";
```

```
IPAddress local_ip(192, 168, 4, 1);
IPAddress gateway(192, 168, 4, 1);
IPAddress subnet(255, 255, 255, 0);

DNSServer dnsServer;

AsyncWebServer server(80);
AsyncEventSource events("/events");
bool ultrasonicMode = false; // Стан ультразвукового режиму

struct AppState
{
    int activeAudioId;              // ID активного аудіофайлу, або -1 якщо
нічого не програється
    std::vector<String> audioFiles; // Список аудіофайлів
    bool isRecording;               // Чи йде запис
    bool isUltrasonic;              // Чи активний ультразвуковий режим
};

AppState getAppState()
{
    AppState state;

    // Отримання списку файлів
    state.audioFiles = getListFiles();

    // Визначення активного аудіофайлу
    state.activeAudioId = isPlaying ? currentFileIndex : -1;

    // Чи йде запис
    state.isRecording = isRecording;

    // Чи активний ультразвуковий режим
    state.isUltrasonic = ultrasonicMode;

    return state;
}

// Функція для обробки запиту на отримання списку файлів
String getFiles()
{
    String json = "[";
    std::vector<String> files = getListFiles();
    for (size_t i = 0; i < files.size(); ++i)
    {
        if (i > 0)
            json += ",";
        json += "{\"id\":" + String(i) + ",\"name\":\"" + files[i] + "\"}";
    }
    json += "]";
    return json;
}

// Функція для обробки запиту на відтворення файлу за id
void handleTogglePlayback(AsyncWebServerRequest *request)
{
    if (!request->hasParam("id", true))
    {
        request->send(400, "text/plain", "Missing 'id' parameter");
        return;
    }

    int id = request->getParam("id", true)->value().toInt();
    std::vector<String> files = getListFiles(); // Отримання списку файлів
```

```
    if (id >= 0 && id < files.size())
    {
        audioEvent(id);
        request->send(200, "text/plain", "Playing file...");
    }
    else
    {
        request->send(404, "text/plain", "File not found");
    }
}

 // Функція для обробки запиту на зупинку відтворення
 void handleStopFile(AsyncWebServerRequest *request)
 {
          Serial.print("Stop play file");
     request->send(200, "text/plain", "Playback stopped.");
// }

void handleDeleteFile(AsyncWebServerRequest *request)
{
    if (request->url().startsWith("/delete-audio/"))
    {
        String idStr = request->url().substring(String("/delete-
audio/").length());
        int id = idStr.toInt();

        std::vector<String> files = getListFiles(); // Отримання списку
файлів
        request->send(200, "text/plain", "File deleted: " + files[id %
files.size()]);

    }
    else
    {
        request->send(400, "text/plain", "Invalid request");
    }
}

// Функція для обробки запиту на запис файлу
void handleToggleRecording(AsyncWebServerRequest *request)
{
    if (!request->hasParam("filename", true))
    {
        request->send(400, "text/plain", "Missing 'filename' parameter");
        return;
    }

    String filename = request->getParam("filename", true)->value();
    Serial.print("Recording started: " + filename);
    recordingEvent(filename);
    // startRecording(filename.c_str()); // Виклик функції для запису
    request->send(200, "text/plain", "");
}

// Функція для обробки запиту на перемикання ультразвукового режиму
void handleToggleUltrasonic(AsyncWebServerRequest *request)
{
    if (!request->hasParam("isActive", true))
    {
        request->send(400, "text/plain", "Missing 'isActive' parameter");
        return;
    }
```

```
    String isActiveParam = request->getParam("isActive", true)->value();
    ultrasonicMode = (isActiveParam == "true");

    if (ultrasonicMode)
    {
        Serial.println("Ultrasonic mode activated");
    }
    else
    {
        Serial.println("Ultrasonic mode deactivated");
            }

    ultrasonicEvent(ultrasonicMode);

    request->send(200, "text/plain", "Ultrasonic mode set to: " +
isActiveParam);
}

void handleGetState(AsyncWebServerRequest *request)
{
    AppState state = getAppState();

    String json = "{";
    json += "\"activeAudioId\": ";
    if (state.activeAudioId == -1)
    {
        json += "null";
    }
    else
    {
        json += String(state.activeAudioId);
    }
    json += ",";
    json += "\"audioFiles\": ";
    json += getFiles();
    json += ",";

    json += "\"isRecording\": " + String(state.isRecording ? "true" :
"false") + ",";
    json += "\"isUltrasonic\": " + String(state.isUltrasonic ? "true" :
"false");
    json += "}";

    request->send(200, "application/json", json);
}

void updateIsRecording(bool isRecording) {
    events.send(isRecording ? "true" : "false", "is-recording");
}

void updateActiveAudioId(int audioId) {
    events.send(String(audioId).c_str(), "active-audio-id");
}

void updateIsUltrasonic(bool isUltrasonic) {
    events.send(isUltrasonic ? "true" : "false", "is-ultrasonic");
}

void updateAudioFiles(const std::vector<String>& audioFiles) {
    String json = getFiles();
    events.send(json.c_str(), "audio-files");
}

// Основна функція для налаштування сервера
```

```
void serverSetup()
{
    WiFi.softAPConfig(local_ip, gateway, subnet);
    WiFi.softAP(ssid, password);
    Serial.println("WiFi точка створена");
    Serial.println(WiFi.softAPIP());
    Serial.println("Connected to WiFi!");
    Serial.print("IP Address:");
    Serial.println(WiFi.softAPIP());

    dnsServer.start(53, "*", WiFi.softAPIP());

    if (!LITTLEFS.begin())
    {
        Serial.println("Помилка ініціалізації LittleFS");
        return;
    }

    DefaultHeaders::Instance().addHeader("Access-Control-Allow-Origin", "*");
    // server.on("/delete-audio", HTTP_DELETE, handleDeleteFile);
    server.on("/toggle-playback", HTTP_POST, handleTogglePlayback);
    server.on("/toggle-recording", HTTP_POST, handleToggleRecording);
    server.on("/toggle-ultrasonic", HTTP_POST, handleToggleUltrasonic);

    // API endpoint to get the current application state
    server.on("/state", HTTP_GET, handleGetState);

    server.serveStatic("/", LITTLEFS, "/").setDefaultFile("index.html");

    server.on("/generate_204", HTTP_GET, [](AsyncWebServerRequest *request)
              { request->redirect("/"); });
    server.on("/hotspot-detect.html", HTTP_GET, [](AsyncWebServerRequest
*request)
              { request->redirect("/"); });
    server.on("/redirect", HTTP_GET, [](AsyncWebServerRequest *request)
              { request->redirect("/"); });
    server.on("/ncsi.txt", HTTP_GET, [](AsyncWebServerRequest *request)
              { request->redirect("/"); });
    server.onNotFound([](AsyncWebServerRequest *request)
                      { request->redirect("/"); });

    server.addHandler(&events);

    server.begin();
}

void serverLoop()
{
    dnsServer.processNextRequest();
}
```

Файл src\server\server.h

```
#include <vector>
#include <string>
#include <Arduino.h>

void serverSetup();
void serverLoop();
void updateIsRecording(bool isRecording);
void updateActiveAudioId(int audioId);
void updateIsUltrasonic(bool isUltrasonic);
void updateAudioFiles(const std::vector<String>& audioFiles);
```

Файл src\wav\wav.cpp

```
#include "wav_utils.h"
#include "config/config.h"

void PrintData(const char* Data, uint8_t NumBytes) {
    for (uint8_t i = 0; i < NumBytes; i++)
        Serial.print(Data[i]);
    Serial.println();
}

bool isValidWav(WavHeader_Struct* Wav) {
    if (memcmp(Wav->RIFFSectionId, "RIFF", 4) != 0) {
        Serial.print("Invalid data - Not RIFF format");
        return false;
    }
    if (memcmp(Wav->RIFFFormat, "WAVE", 4) != 0) {
        Serial.print("Invalid data - Not Wave file");
        return false;
    }
    if (memcmp(Wav->FormatSectionId, "fmt", 3) != 0) {
        Serial.print("Invalid data - No format section found");
        return false;
    }
    if (memcmp(Wav->DataSectionId, "data", 4) != 0) {
        Serial.print("Invalid data - data section not found");
        return false;
    }
    if (Wav->FormatId != 1) {
        Serial.print("Invalid data - format Id must be 1");
        return false;
    }
    if (Wav->FormatSize != 16) {
        Serial.print("Invalid data - format section size must be 16.");
        return false;
    }
    if ((Wav->NumChannels != 1) & (Wav->NumChannels != 2)) {
        Serial.print("Invalid data - only mono or stereo permitted.");
        return false;
    }
    if (Wav->SampleRate > 48000) {
        Serial.print("Invalid data - Sample rate cannot be greater than
48000");
        return false;
    }
    if ((Wav->BitsPerSample != 8) & (Wav->BitsPerSample != 16)) {
        Serial.print("Invalid data - Only 8 or 16 bits per sample
permitted.");
        return false;
    }
    return true;
}

void dumpWavHeader(WavHeader_Struct* Wav) {
    if (memcmp(Wav->RIFFSectionId, "RIFF", 4) != 0) {
        Serial.print("Not a RIFF format file - ");
        PrintData(Wav->RIFFSectionId, 4);
        return;
    }
    if (memcmp(Wav->RIFFFormat, "WAVE", 4) != 0) {
        Serial.print("Not a WAVE file - ");
        PrintData(Wav->RIFFFormat, 4);
        return;
    }
    if (memcmp(Wav->FormatSectionId, "fmt", 3) != 0) {
```

```
        Serial.print("fmt ID not present - ");
        PrintData(Wav->FormatSectionId, 3);
        return;
    }
    if (memcmp(Wav->DataSectionId, "data", 4) != 0) {
        Serial.print("data ID not present - ");
        PrintData(Wav->DataSectionId, 4);
        return;
    }

    Serial.println("-----------------------------------");
    Serial.println("");
    Serial.print("Total size :"); Serial.println(Wav->Size);
    Serial.print("Format section size :"); Serial.println(Wav->FormatSize);
    Serial.print("Wave format :"); Serial.println(Wav->FormatId);
    Serial.print("Channels :"); Serial.println(Wav->NumChannels);
    Serial.print("Sample Rate :"); Serial.println(Wav->SampleRate);
    Serial.print("Byte Rate :"); Serial.println(Wav->ByteRate);
    Serial.print("Block Align :"); Serial.println(Wav->BlockAlign);
    Serial.print("Bits Per Sample :"); Serial.println(Wav->BitsPerSample);
    Serial.print("Data Size :"); Serial.println(Wav->DataSize);
    Serial.println("");
    Serial.println("-----------------------------------");
}

// Розмір заголовка — завжди 44 байти
static const uint32_t WAV_HEADER_SIZE = 44;

void prepareWavHeader(WavHeader_Struct* h, uint32_t dataSize) {
    // RIFF chunk descriptor
    memcpy(h->RIFFSectionId, "RIFF", 4);
    h->Size       = 36 + dataSize;                 // 4 + (8 + SubChunk1Size)
+ (8 + SubChunk2Size)
    memcpy(h->RIFFFormat, "WAVE", 4);

    // fmt sub-chunk
    memcpy(h->FormatSectionId, "fmt ", 4);
    h->FormatSize     = 16;                         // PCM
    h->FormatId       = 1;                          // PCM = 1
    h->NumChannels    = 1;                          // моно (1) або стерео
(2)
    h->SampleRate     = SAMPLE_RATE;                // ваша константа
    h->BitsPerSample  = 16000;            // ваша константа
    h->ByteRate       = h->SampleRate
                        * h->NumChannels
                        * h->BitsPerSample / 8;
    h->BlockAlign     = h->NumChannels
                        * h->BitsPerSample / 8;

    // data sub-chunk
    memcpy(h->DataSectionId, "data", 4);
    h->DataSize       = dataSize;
}
```

Файл src\wav\wav_utils.h

```
#pragma once

#include <Arduino.h>

struct WavHeader_Struct {
    char RIFFSectionId[4];
    uint32_t Size;
    char RIFFFormat[4];
```

```
    char FormatSectionId[4];
    uint32_t FormatSize;
    uint16_t FormatId;
    uint16_t NumChannels;
    uint32_t SampleRate;
    uint32_t ByteRate;
    uint16_t BlockAlign;
    uint16_t BitsPerSample;
    char DataSectionId[4];
    uint32_t DataSize;
};

bool isValidWav(WavHeader_Struct* header);
void dumpWavHeader(WavHeader_Struct* header);
void prepareWavHeader(WavHeader_Struct* header, uint32_t dataSize);
```